Pós-Graduação em Ciência da Computação

"UTILIZANDO O PROTOCOLO BITCOIN PARA CONDUÇÃO DE COMPUTAÇÕES MULTILATERAIS SEGURAS E JUSTAS"

Por

Márcio Barbosa de Oliveira Filho

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE/2016

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MÁRCIO BARBOSA DE OLIVEIRA FILHO

"UTILIZANDO O PROTOCOLO BITCOIN PARA CONDUÇÃO DE COMPUTAÇÕES MULTILATERAIS SEGURAS E JUSTAS"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADORA: ANJOLINA GRISI DE OLIVEIRA
CO-ORIENTADOR: RUY J. GUERRA B. DE QUEIROZ

RECIFE/2016

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

O48u Oliveira Filho, Márcio Barbosa de
Utilizando o protocolo Bitcoin para condução de computações multilaterais seguras e justas / Márcio Barbosa de Oliveira Filho. – 2016.
180 p.: il., fig., tab.

Orientador: Anjolina Grisi de Oliveira.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2016.
Inclui referências e apêndices.

1. Ciência da computação. 2. Segurança da informação. 3. Teoria da computação I. Oliveira, Anjolina Grisi de (orientadora). II. Título.

004 CDD (23. ed.) UFPE- MEI 2016-019

Márcio Barbosa de Oliveira Filho

Utilizando o Protocolo Bitcoin para Condução de Computações Multilaterais Seguras e Justas

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

_____

Profa. Dra. Anjolina Grisi de Oliveira
Orientadora do Trabalho de Dissertação

Aprovado em: 02/02/2016

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Gustavo Soares da Fonseca
Centro de Informática / UFPE

_____

Prof. Dr. Jeroen Antonius Maria van de Graaf
Departamento de Ciência da Computação/ UFMG

_____

Prof. Dr. Ruy José Guerra Barretto de Queiroz (Co-Orientador)
Centro de Informática / UFPE

AGRADECIMENTOS

Primeiramente agradeço à minha família por todo o carinho, paciência e força que me deram durante esse longo período de estudos e trabalho duro. Também ao amigo Anselmo pelas proveitosas discussões sobre segurança computacional, seriados e filmes.

Agradeço a todos os professores do Centro de Informática que contribuíram para a minha formação acadêmica. Foram muitos e em momentos variados. Como Silvo Melo, que me apresentou aos espaços vetoriais, Anjolina, a primeira a me falar de conjuntos não-enumeráveis, Ruy, mostrando de forma tão simples o problema da parada, Liliane, me orientando na maratona, Katia, me orientando nos momentos finais da graduação, Castor, Paulo, Armando e muitos, muitos, outros. Muito obrigado a todos. Em especial, aos meus orientadores, pela disposição em me guiar durante este trabalho.

Aproveito também para registrar minha gratidão a todos os autores que compartilharam seus conhecimentos e, dessa forma, possibilitaram a realização deste trabalho.

Deep in the human unconscious is a pervasive need for a logical universe that makes sense. But the real universe is always one step beyond logic.

—FRANK HERBERT, *DUNE*

RESUMO

Suponha que dois milionários desejam descobrir quem é o mais rico sem que, para isso, um descubra o valor da fortuna do outro. Esse problema pode ser facilmente resolvido se ambos concordarem sobre um juiz a quem eles possam confiar a tarefa de calcular a resposta sem quebrar a privacidade de nenhum dos dois. O desafio, no entanto, é substituir esse juiz por uma interação multilateral, ou protocolo, que alcance o mesmo grau de segurança. Isso significa conduzir uma *computação multilateral segura*.

Esse exemplo é conhecido como *o problema dos milionários de Yao* e foi proposto por Andrew Yao em um dos primeiros esforços para desenvolver uma forma geral de se conduzir computações multilaterais seguras. Desde lá, na década de 1980, vários avanços foram feitos nesse sentido e, hoje, já é um resultado bem estabelecido que qualquer função multilateral pode ser computada de maneira segura.

Esse importante resultado, no entanto, vem com uma importante ressalva: a *justiça* não pode ser alcançada de maneira geral. Entende-se por justiça a garantia de que, no final de uma computação, ou todos os participantes recebem suas respostas ou nenhum deles recebe.

Motivadas por esse resultado de impossibilidade, várias definições alternativas de justiça foram concebidas. Uma delas considera uma computação como sendo justa se os participantes que agirem desonestamente forem *monetariamente* penalizados. Ou seja, se alguns participantes receberem o resultado da computação e privarem os demais de receberem, então esse conluio é penalizado e os demais participantes, compensados.

Com essa definição, uma computação justa pode ser vista como o cumprimento de um contrato, no qual cada participante se compromete a agir honestamente ou a pagar uma multa. Sob essa perspectiva, trabalhos recentes mostram como *criptomoedas* podem ser utilizadas para escrever esses contratos e, portanto, servir para condução de computações multilaterais seguras e monetariamente justas.

Uma criptomoeda é um sistema monetário que se baseia em princípios criptográficos para alcançar segurança e controlar a taxa de emissão de novas moedas. O *Bitcoin*, criado em 2008 por Satoshi Nakamoto, foi a primeira realização dessa ideia. Ele se baseia em uma estrutura de dados pública chamada *blockchain*, que armazena o histórico de todas as transações já realizadas. A segurança da *blockchain*, incluindo sua não-maleabilidade, é garantida pela dificuldade da *prova de trabalho* exigida para que novas informações sejam adicionadas nessa estrutura.

Cada transferência de moedas no *Bitcoin* é feita de maneira que um usuário só pode recebê-las mediante a apresentação de uma entrada que satisfaça um *script* especificado pelo remetente. Contratos escritos através desses *scripts* se fazem cumprir sem a necessidade de intervenção de uma parte confiável externa. Essa característica é o que faz o *Bitcoin* ser adequado e atrativo para se conferir justiça para computações multilaterais seguras.

Um dos nossos objetivos neste trabalho é a realização de um estudo sobre os resultados que permitem a computação segura de uma função qualquer e dos que estabelecem a impossi-

bilidade de se alcançar justiça de maneira geral. Tentaremos manter o rigor matemático para evitar uma apresentação estritamente panorâmica. Além disso, analisaremos criticamente uma construção proposta para utilizar o *Bitcoin* como plataforma para condução de computações multilaterais seguras e justas. Por fim, a partir dos pontos positivos e negativos levantados, apresentaremos uma proposta nossa com a mesma finalidade.

Palavras-chave: Computação Multilateral Segura. Computação Justa. Bitcoin. Criptomoeda. Blockchain. Criptografia

ABSTRACT

Suppose that two millionaires want to find out which one is the richest, but without revealing how much their fortunes are worth in the process. This problem can be easily solved if they trust a judge to compute the desired answer without compromising their privacy. The challenge, however, is to replace such a judge by a multiparty interaction, or protocol, that achieves the same level of security. That is, the challenge is to carry out a *secure multiparty computation*.

The previous example is known as *Yao's millionaires' problem* and was stated by Andrew Yao in one of the first efforts to develop a general algorithm to carry out secure multiparty computations. Since then, in the 1980s, several advances were made to achieve this goal and, nowadays, it is a well-established result that any multiparty function can be securely computed.

This important result, however, comes with an important restriction: *fairness* cannot be achieved in general. Fairness is the guarantee that, at the end of a computation, either all parties receive their outputs or none of them does.

Motivated by this impossibility, several alternative definitions of fairness have been proposed. One of them considers a computation to be fair if the dishonest parties are *monetarily* penalized and the honest ones are *monetarily* compensated.

According to this definition, a fair computation can be viewed as the enforcement of a contract, in which the parties agree on paying a penalty if they misbehave. Recent works have shown how *cryptocoins* can be used to write those contracts and, therefore, to be used for carrying out secure and monetarily fair multiparty computations.

A cryptocoin is a monetary system that relies on cryptographic principles for achieving security and controlling the coin issuing rate. *Bitcoin*, created in 2008 by Satoshi Nakamoto, was the first practical realization of this idea. In its core, there is a publicly maintained data structure called *blockchain*, which works as a ledger and stores every transaction ever made. The security of the blockchain, including its non-malleability, is guaranteed by the difficulty of the *proof of work* required to add new data to it.

A Bitcoin transaction can only be redeemed by a user that presents an input satisfying a *script* specified by the issuer of the transaction. Contracts written on these scripts are enforced without an external trusted third-party to intervene. This makes Bitcoin suitable and interesting to confer monetary fairness to multiparty computations.

One of the goals of this work is to present the results that allow any function to be securely computed and the ones that show the impossibility of achieving fairness in general. We will try to present these results with the appropriate mathematical rigor to avoid giving just an overview of them. We will also analyze a recently proposed construction that uses Bitcoin as a platform to carry out fair multiparty computations. Finally, based on its positive and negative points, we will propose a construction with the same goal.

Keywords: Secure Multiparty Computation. Fair Computations. Bitcoin. Cryptocoin. Blockchain. Cryptography

LISTA DE FIGURAS

2.1 Decomposição da *blockchain* em cadeias a partir de suas bifurcações 28
2.2 Ilustração da promoção de cadeias secundárias 29
2.3 O problema do gasto-duplo 30
2.4 Exemplo de transação com uma saída 34
2.5 Exemplo de transação com uma entrada 35
2.6 Exemplo de transação com múltiplas entradas e saídas 36

7.1 Os depósitos escada e cobertura feitos na construção escada 120
7.2 A Transação $Txn^{cr}_{i\to j}$ 141
7.3 A Transação $Txn^{cred}_{i\to j}$ 142
7.4 A Transação $Txn^{reem}_{i\to j}$ 143

8.1 A Transação Txn^{dep}_{i} 161
8.2 A Transação Txn^{comp} 162
8.3 A Transação Txn^{resg}_{i} 163
8.4 A Transação $Txn^{pen}_{i\to j}$ 163

LISTA DE TABELAS

2.1 Campos que compõem as entradas das transações 21
2.2 Campos que compõem as saídas das transações 21
2.3 Campos que compõem as transações 22
2.4 Campos que compõem os blocos 26

4.1 Valores relacionados à computação de Π'_{and} 79

8.1 As transações *Bitcoin* envolvidas na implementação da funcionalidade f_{cmm}. . . 160

A.1 Estratégias $A^n_{i,0}$ e $A^n_{i,1}$ 178
A.2 Estratégias $B^n_{i,0}$ e $B^n_{i,1}$ 178

LISTA DE ACRÔNIMOS

TPC Terceira Parte Confiável 63

F_2 *corpo finito com dois elementos* 72

AuthBA *Authenticated Byzantine Agreement* 89

$EVCS_{m,m}$ (m,m)*-esquema verificável de compartilhamento de segredo* 114

MTI *Máquina de Turing Interativa* 43

ECDSA *Elliptic Curve Digital Signature Algorithm* (Algoritmo de Assinatura Digital de Curvas Elípticas) 23

SUMÁRIO

1 Introdução **16**

1.1 Objetivos 18

2 Bitcoin **20**

2.1 Funcionamento do *Bitcoin* 20

2.1.1 Visão Geral 20

2.1.2 Transações 21

2.1.3 *Scripts* 22

2.1.4 Assinatura Digital 23

2.1.5 Ponteiros *Hash* e Árvores de Merkle 24

2.1.6 Blocos 25

2.1.7 Prova de Trabalho 25

2.1.8 Juntando as Peças: *Blockchain* e Consenso 27

2.2 O Problema da Maleabilidade das Transações 31

2.2.1 Proposta 1: ANDRYCHOWICZ et al. (2014a) 32

2.2.2 Proposta 2: KUMARESAN e BENTOV (2014) 33

2.2.3 Proposta 3: ANDRYCHOWICZ et al. (2015) 33

2.3 Notação e Nomenclatura 34

3 Definições **38**

3.1 Funções Desprezíveis 38

3.2 Indistinguibilidade 38

3.3 Funções Unidirecionais, Arapucas Aprimoradas e Predicados Núcleo-duro . . . 39

3.4 Sistemas de Provas Fortes de Conhecimento 42

3.4.1 Visão Geral 42

3.4.2 Definições Preliminares 43

3.4.3 Sistema de provas fortes de conhecimento 44

3.4.4 Conhecimento Zero 46

3.5 Esquemas de Comprometimento 47

3.5.1 Esquemas de Comprometimento Equívocos 49

3.6 Esquemas de Compartilhamento de Segredo 50

3.7 Funções *Hash* 52

3.8 Esquemas de Criptografia Assimétricos 55

3.8.1 Construção a partir de Funções Arapuca 58

3.9 Assinaturas Digitais 59

4 Computação Multilateral Segura **62**
4.1 Definições Preliminares . . . 62
4.1.1 Segurança . . . 62
4.1.2 Restrição sobre as Funcionalidades . . . 64
4.1.3 Modelo de Segurança . . . 64
4.1.4 Segurança Contra Adversários Semi-Honestos . . . 65
4.1.5 Segurança Contra Adversários Maliciosos . . . 68
4.2 Adversário Semi-Honesto . . . 71
4.2.1 Representação das Funcionalidades . . . 72
4.2.2 Composição e Redução de Funcionalidades . . . 73
4.2.3 Transferência Oblívia . . . 74
4.2.4 Computação Privada de um Circuito . . . 76
4.2.5 Funcionalidades Probabilísticas . . . 83
4.2.6 Protocolos Canônicos . . . 85
4.3 Adversário Malicioso . . . 86
4.3.1 Canal de Comunicação . . . 87
4.3.2 Pré-Compilador . . . 88
4.3.3 Pós-Compilador . . . 89
4.3.4 Composição e Redução de Funcionalidades . . . 91
4.3.5 Funcionalidades Auxiliares . . . 92
4.3.5.1 *Broadcast* privado . . . 92
4.3.5.2 Transmissão de Imagem . . . 94
4.3.5.3 Computação Multilateral Autenticada . . . 95
4.3.5.4 Geração de Fonte de Aleatoriedade . . . 97
4.3.5.5 Comprometimento com uma Entrada . . . 99
4.3.6 O Compilador . . . 100
4.4 Considerações Finais . . . 105

5 Computação Justa **106**
5.1 Impossibilidade Geral . . . 106
5.2 Definições Relaxadas de Justiça . . . 108
5.3 Definição Adotada de Justiça . . . 109

6 Protocolos Justos com *Bitcoin* **111**
6.1 Modelo de Computação . . . 111
6.2 Computação Justa Através da Reconstrução Justa . . . 113
6.2.1 Visão Geral . . . 113
6.2.2 Definições Preliminares . . . 114
6.2.3 Juntando as Peças: a Redução . . . 116

7 Reconstrução Justa Através da Construção Escada **118**
7.1 Visão Geral . . . 118
7.2 A funcionalidade f_{cr} . . . 120
7.3 Construção Escada no Modelo f_{cr}-híbrido . . . 122
7.4 A Funcionalidade f_{rec}^{esc} para Reconstrução Justa . . . 127
7.5 Implementação de f_{cr} com *Bitcoin* . . . 140
7.5.1 Visão Geral . . . 140
7.5.2 As Transações $Txn_{i\to j}^{cr}$ . . . 141
7.5.3 As Transações $Txn_{i\to j}^{reem}$ . . . 142
7.5.4 Juntando as Peças: o Protocolo Π_{cr} . . . 143
7.6 Análise da Construção Escada e sua Implementação *Bitcoin* . . . 144

8 Reconstrução Justa Através de Comprometimento Monetário Multilateral **146**
8.1 Visão Geral . . . 146
8.2 A Funcionalidade f_{rec}^{cmm} para Reconstrução Justa . . . 147
8.3 A Funcionalidade f_{cmm} . . . 149
8.4 O Protocolo Π_{rec}^{cmm} no Modelo f_{cmm}-híbrido . . . 151
8.5 Implementação de f_{cmm} com *Bitcoin* . . . 159
8.5.1 Visão Geral . . . 159
8.5.2 A Transação Txn_i^{dep} . . . 160
8.5.3 A Transação Txn^{comp} . . . 161
8.5.4 A Transação Txn_i^{resg} . . . 162
8.5.5 A Transação $Txn_{i\to j}^{pen}$ . . . 162
8.5.6 Juntando as Peças: o Protocolo Π_{cmm} . . . 163
8.6 Análise de Π_{rec}^{cmm} e sua Implementação *Bitcoin* . . . 167

9 Considerações Finais e Trabalhos Futuros **170**
9.1 Considerações Finais . . . 170
9.2 Trabalhos Futuros . . . 171

Referências **173**

A Prova da Impossibilidade de CLEVE (1986) **176**

1 INTRODUÇÃO

O *Bitcoin* pode ser definido como um sistema de pagamento distribuído e baseado em técnicas de criptografia. Proposto por NAKAMOTO (2008), o *Bitcoin* teve uma boa aceitação inicial e, em decorrência disso, uma grande valorização comercial. Sua natureza aberta, descentralizada e sem autoridade reguladora fez com que uma grande comunidade se formasse em torno do *Bitcoin*, inclusive para usá-lo como base na criação de outras moedas, as chamadas *altcoins*. Diversas dessas *altcoins* foram criadas, a exemplo do *Dogecoin*, *Litecoin* e *Peercoin*, cada uma tendo objetivos e protocolos diferentes.

As peças fundamentais do *Bitcoin* são as *transações*. Elas especificam a transferência de valores entre dois conjuntos de usuários, geralmente referenciados de forma pseudo-anônima através de chaves públicas. Essas chaves, por sua vez, são comumente conhecidas como *endereços* e podem ser geradas de forma autônoma e em qualquer quantidade pelo usuário.

O histórico de todas as transações já realizadas é armazenado em uma estrutura pública chamada *blockchain*, a qual é comumente comparada a um livro-razão. A *blockchain* é formada por vários *blocos*, os quais contém uma ou mais transações. Uma transação só é considerada válida depois de armazenada em algum bloco publicado na *blockchain*. Por isso, com o auxílio da *blockchain*, é possível identificar e rejeitar transações indevidas, que tentem *gastar o mesmo valor duas vezes*. Esse problema é conhecido como *double-spending* e é uma das principais questões que devem ser resolvidas na concepção de sistemas monetários digitais.

Certamente a manutenção da *blockchain* por uma terceira parte central confiável, como um banco por exemplo, eliminaria o problema do *double-spending*. No entanto, a introdução de entidades centrais gera efeitos indesejáveis a um sistema, principalmente relacionados a sua disponibilidade, segurança e resistência a falhas. Uma das características mais atraentes do *Bitcoin* é justamente o fato de que a *blockchain* é mantida de forma totalmente distribuída. Para isso, os usuários se conectam através de uma rede ponto-a-ponto para chegarem, continuamente, a um *consenso* sobre quais transações devem estar presentes na *blockchain*.

Para publicar um novo bloco de transações na *blockchain*, deve-se exibir uma *prova de trabalho* calculada sobre esse bloco. De forma geral, uma prova de trabalho é a resposta para uma instância de um problema considerado computacionalmente difícil de se resolver, mas fácil de se conferir. No *Bitcoin*, a prova de trabalho consiste na manipulação do bloco a fim de que seu *hash* seja menor que um valor alvo publicamente conhecido. Para tanto, deve-se calcular repetidamente o *hash* do bloco, alterando alguns de seus campos, como *timestamp* e *nonce*, ou características, como a ordem das transações.

Para incentivar os usuários a investir recursos computacionais na validação de blocos, o protocolo *Bitcoin* estabelece uma recompensa financeira para cada prova de trabalho gerada. Essa recompensa é a única maneira pela qual novas "moedas" são criadas. Por essa razão, o processo de validar blocos, gerando provas de trabalho, é chamado de *mineração*. Os usuários que se dedicam a isso são conhecidos como *mineradores*.

A estrutura dos blocos foi concebida de modo a conter informações sobre o estado da *blockchain* no momento em que eles foram criados. Dessa forma, a publicação de um bloco pode ser encarada como um "voto" em uma determinada configuração da *blockchain*. A versão correta da *blockchain*, conhecida como *cadeia principal* (*main chain*), é a que possui mais blocos, ou seja, a que foi mais votada. Um bloco é rejeitado quando os usuários se recusam a estender sua versão da *blockchain*, e isso pode acontecer se ele possui transações inválidas ou não pertence à cadeia principal, por exemplo.

Devido à dificuldade do cálculo da prova de trabalho, a taxa de geração de blocos de um minerador é limitada por seu poder computacional. Dessa forma, sua influência na votação que estabelece o consenso sobre a *blockchain* é proporcional à sua fatia do poder computacional da rede. Uma consequência desse fato é que apenas um aumento no poder de processamento garante vantagem na geração de blocos e não, por exemplo, o controle de diversos endereços. Por isso, uma das hipóteses centrais do *Bitcoin* é a de que a maior parte do poder computacional da rede está sob controle de partes honestas. Se alguma entidade maliciosa possuísse mais de 50% desse poder, ela poderia manipular o consenso sobre a *blockchain* com alta probabilidade.

A natureza aberta e distribuída do *Bitcoin*, bem como o caráter programável de suas transações, são pontos atraentes para sua utilização como plataforma para resolução de problemas complexos. Por essa razão, trabalhos recentes como os de ANDRYCHOWICZ et al. (2014b), ANDRYCHOWICZ et al. (2014a), BENTOV e KUMARESAN (2014) e KUMARESAN e BENTOV (2014) têm explorado o *Bitcoin* para conferir *justiça* a computações multilaterais seguras.

Para explicar melhor conceitos importantes, como o de justiça, vamos utilizar o *problema dos milionários*, proposto por YAO (1982). Nele, dois milionários querem descobrir quem é mais rico sem que, para isso, o valor de suas riquezas seja revelado. Naturalmente, se um juiz confiável estivesse disponível, o valor das riquezas poderia ser informado somente a ele, e a correção do resultado decorreria diretamente da confiabilidade desse juiz. O desafio, no entanto, é conceber um protocolo multilateral que tenha o mesmo efeito de tal juiz, mas que não precise recorrer a nenhuma entidade central.

O sigilo das entradas, bem como a correção do resultado, são propriedades de uma computação multilateral *segura*. Importantes resultados anteriores, como os registrados por YAO (1982) e GOLDREICH (2004), mostram que é possível conduzir a computação segura de uma funcionalidade qualquer, mesmo na presença de uma maioria desonesta. Isso garante, por exemplo, a existência de um protocolo através do qual os milionários do problema de Yao podem interagir para *emular* um juiz confiável. Isso significa, informalmente, que esse protocolo "vaza" tanta informação sigilosa quanto a própria interação com um juiz, ou *terceira parte confiável*, que é o caso ideal.

Uma computação multilateral segura, no entanto, pode não ser *justa*. A justiça é a garantia de que ou todos os participantes recebem o resultado da computação ou nenhum deles recebe. No problema dos milionários, a falta de justiça pode fazer com que um dos milionários

descubra qual dos dois é o mais rico e decida então *abortar* a computação, de modo que o outro fique sem saber o resultado.

Infelizmente, nem toda funcionalidade admite uma computação multilateral justa sem a hipótese de que a maioria dos participantes é honesta. Esse resultado, inicialmente demonstrado por CLEVE (1986), é ainda mais forte quando aplicado ao caso particular de funcionalidades com apenas dois participantes, como no problema de Yao. Nelas, a propriedade de justiça só pode ser alcançada sob a hipótese de que *todos* os participantes são honestos, o que, obviamente, não é uma hipótese razoável.

Diante desse resultado de impossibilidade, definições relaxadas de justiça são utilizadas para que pelo menos algum grau de justiça possa ser alcançado de forma geral. Pode-se aceitar, por exemplo, que a justiça seja quebrada, mas apenas com uma probabilidade *desprezível*. Outra possibilidade, utilizada no decorrer deste trabalho, é o de penalizar *monetariamente* os participantes que quebram a justiça (abortam), compensando os demais.

Para alcançar a noção de *justiça monetária*, é necessário que cada participante "assine" uma espécie de contrato se comprometendo a pagar uma penalidade caso se comporte desonestamente. É preciso ainda que esse contrato se faça cumprir mesmo no caso em que o participante aborte. Mais uma vez, uma terceira parte confiável, como um banco em que se pudesse fazer um depósito como garantia, seria suficiente para implementar essa ideia. No entanto, pode-se eliminar a dependência de uma autoridade central através da utilização do *Bitcoin* para escrever esses contratos.

Nosso trabalho se baseia nas estratégias de BENTOV e KUMARESAN (2014), ANDRYCHOWICZ et al. (2014a) e ANDRYCHOWICZ et al. (2014b) de utilização do *Bitcoin* para conduzir computações justas. De maneira geral, elas conferem justiça monetária a uma computação através do uso de protocolos para *reconstrução justa de um segredo*. Para isso, construções baseadas em transações *Bitcoin* para implementar contratos são apresentadas.

Por fim, a partir da análise do trabalho de BENTOV e KUMARESAN (2014), nós propomos um protocolo baseado no *Bitcoin* para conduzir reconstrução justa de um segredo. As construções propostas tornam esse protocolo mais eficiente e escalável que os analisados.

1.1 Objetivos

Definimos como objetivo geral deste trabalho a análise do uso do *Bitcoin* para conduzir computações multilaterais seguras e justas. Para alcançá-lo, definimos os seguintes objetivos específicos:

- Estudo da possibilidade geral de se conduzir computações seguras,
- Estudo da impossibilidade geral de se conduzir computação justas,
- Estudo de definições alternativas de justiça e, em particular, da justiça monetária,

- Análise de estratégias de utilização do *Bitcoin* para conferir justiça (monetária) a computações multilaterais,
- A partir da análise anterior, propor uma estratégia eficiente.

2 BITCOIN

Neste capítulo temos como objetivo apresentar as características gerais do *Bitcoin* e discutir com maior profundidade aquelas que são mais importantes para o restante deste trabalho. Fazemos isso em duas etapas. Inicialmente apresentaremos as peças fundamentais do *Bitcoin* e como elas se relacionam. Por fim, apresentaremos a notação que será utilizada para representar as características do *Bitcoin* relevantes no decorrer deste trabalho.

2.1 Funcionamento do *Bitcoin*

2.1.1 Visão Geral

O *Bitcoin* é uma criptomoeda proposta em NAKAMOTO (2008) por Satoshi Nakamoto. Desde seu lançamento, o *Bitcoin* vem adquirindo cada vez mais adeptos devido a características como flexibilidade, descentralização e não regulação. Essa criptomoeda tem três componentes fundamentais: uma rede ponto-a-ponto de usuários construída sobre a *Internet*, um conjunto de regras especificadas inicialmente pela implementação de Satoshi Nakamoto, mas que vêm evoluindo ao longo do tempo e, por fim, uma estrutura de dados mantida de maneira distribuída chamada *blockchain*.

A rede ponto-a-ponto serve para que os usuários recebam e enviem diversos dados, como transações, blocos de transações e informações sobre a *blockchain*. O conjunto de regras, por sua vez, especifica diversos parâmetros do funcionamento do *Bitcoin* que servem, por exemplo, para determinar a validade de transações e blocos. Algumas regras, como o cálculo da prioridade dada às transações ou o algoritmo utilizado para roteamento de blocos, podem ser alteradas por cada usuário de maneira independente. Outras, no entanto, precisam ser adotadas por *todos* para que haja uma versão consistente do *Bitcoin*. É o caso, por exemplo, do tamanho máximo dos blocos e dos comandos que são permitidos nos *scripts* das transações.

Tanto a rede ponto-a-ponto quanto as regras que os usuários seguem têm como objetivo a manutenção da *blockchain*. A *blockchain* é uma estrutura de dados pública que armazena, em blocos, todas as transações que já ocorreram. Uma transação só é considerada válida se estiver na *blockchain*. Por isso, o ponto central do *Bitcoin* é o estabelecimento de um *consenso* sobre quais informações devem estar na *blockchain*. A solidez desse consenso, por sua vez, se baseia, principalmente, nas propriedades da *prova de trabalho* utilizada pelo *Bitcoin*. É ela que torna difícil a manipulação do consenso estabelecido pelos usuários e, consequentemente, é o que torna o *Bitcoin* seguro.

Além da prova de trabalho, o *Bitcoin* também tira proveito de outras construções criptográficas, como assinaturas digitais e funções *hash*.

Vamos convencionar chamar de *Bitcoin*, ou de protocolo *Bitcoin*, todo o sistema composto por usuários, rede ponto-a-ponto, regras de interação e *blockchain*. Chamaremos de *bitcoin* (minúsculo) a unidade monetária desse sistema.

Tabela 2.1: Campos que compõem as entradas das transações

Campo	Descrição
Transação Anterior	O *id* da transação de origem dos *bitcoins*.
Índice da Saída Referenciada	O índice da saída da transação referenciada que fornece os *bitcoins*.
Script (*scriptSig*)	Fornece as entradas para satisfazer o *script* da saída especificada e, assim, permitir que os *bitcoins* sejam gastos.
Tamanho do *script*	Inteiro de 9 *bytes*, representando o tamanho, em *bytes*, do *scriptSig*.

Fonte: Tabela criada pelo autor a partir de informações contidas em WIKI (2015a) e FOUNDATION (2015a).

Tabela 2.2: Campos que compõem as saídas das transações

Campo	Descrição
Valor	A quantidade de *bitcoins* disponibilizados pela saída.
Script (*ScriptPubKey*)	*Script* que deve ser satisfeito para a que a saída possa ser reivindicada.
Tamanho do *script*	Inteiro de 9 *bytes*, representando o tamanho, em *bytes*, do *ScriptPubKey*.

Fonte: Tabela criada pelo autor a partir de informações contidas em WIKI (2015a) e FOUNDATION (2015b).

2.1.2 Transações

As peças fundamentais do protocolo *Bitcoin* são as transações, que especificam transferências de *bitcoins* entre dois conjuntos de usuários. O conjunto de destino é definido pelas *saídas* da transação. Cada uma delas disponibiliza determinada quantidade de *bitcoins* e especifica um *script*, o *ScriptPubKey*. Os *bitcoins* de uma saída só podem ser reivindicados através do fornecimento de entrada apropriada para satisfazer o *ScriptPubKey*.

A origem dos *bitcoins*, por sua vez, é definida pelas *entradas* da transação. Uma entrada reivindica *bitcoins* de uma saída contida em alguma transação preexistente. Para isso, ela deve especificar um *script*, conhecido por *ScriptSig*, que, ao ser executado, produza entrada apropriada para satisfazer o *ScriptPubKey* da saída referenciada.

A Tabela 2.1 traz uma visão geral dos campos que compõem uma entrada. A Tabela 2.2 faz uma breve descrição dos campos que compõem as saídas.

Além das entradas e saídas, um campo importante das transações é o *locktime*. Ele especifica o instante a partir do qual a transação pode ser considerada finalizada (*final*). Antes disso, ela pode ser distribuída na rede ponto-a-ponto, mas não pode ser incluída na *blockchain*. O valor do *locktime* pode ser definido tanto em termos da quantidade de blocos na *blockchain*, medida conhecida como altura (*height*), quanto em forma de estampa de tempo (*timestamp*).

Os campos de uma transação são sumarizados na Tabela 2.3.

Tabela 2.3: Campos que compõem as transações

Campo	Descrição
Id	O valor *hash* da transação calculado pela função *double SHA256* (a função *SHA256* composta com ela própria).
Número de Versão	Atualmente assume valor 1.
Quantidade de Entradas	A quantidade de entradas contidas na transação.
Lista de Entradas	As entradas propriamente ditas.
Quantidade de Saídas	A quantidade de saídas contidas na transação.
Lista de Saídas	As saídas propriamente ditas.
Locktime	Campo que especifica a partir de quando a transação poderá ser considerada válida. Pode ser especificado através de uma estampa de tempo ou da altura da *blockchain*.

Fonte: Tabela criada pelo autor a partir de informações contidas em WIKI (2015a) e FOUNDATION (2015b).

2.1.3 *Scripts*

A linguagem de *script* utilizada no *Bitcoin* é baseada em pilha e possui instruções voltadas para a manipulação das primitivas criptográficas mais utilizadas. Ela não é Turing completa e, em particular, não possui laços, dificultando ataques de negação de serviço e a criação de programas maliciosos.

Os *scripts* possuem exclusivamente a função de autorizar a transferência de *bitcoins*. Como mencionamos na Seção 2.1.2, saídas e entradas de uma transação precisam especificar *scripts*. Em particular, o *script* de uma entrada precisa satisfazer o *script* da saída que ela referencia. Nesse contexto, dizemos que um *script* S_1 satisfaz outro *script* S_2 se a execução do *script* resultante da concatenação dos dois ($S_1||S_2$) não gera erros e deixa um valor diferente de zero na pilha de execução. De maneira equivalente, podemos pensar que S_1 satisfaz S_2 se, ao final de sua execução, S_1 retorna valores tais que, quando fornecidos como entrada, fazem S_2 executar sem erros e gerar uma saída não nula.

Existem dois tipos mais comuns de transações, os quais definem padrões de *scripts* de entrada e saída. O primeiro deles é o *Pay to Pubkey Hash*, e se baseia em um esquema de assinaturas digitais[1]. Nesse tipo de transação, os *bitcoins* são transferidos para quem apresentar uma assinatura feita com uma determinada chave secreta. Naturalmente essa chave é especificada através da chave pública correspondente.

Nas transferências do tipo *Pay to Pubkey Hash*, o *script ScriptPubKey* (o da saída) espera ser alimentado com (i) uma chave pública que tenha um determinado valor *hash*, comumente chamado de *endereço*, e (ii) uma assinatura gerada com a chave secreta correspondente ao endereço especificado em (i). Essa assinatura é fornecida pelo *ScriptSig* da entrada que reivindica

[1]Esquemas de assinaturas digitais são apresentados na Definição 3.27.

os *bitcoins* e deve ser gerada sobre a *versão simplificada* da transação que contém essa entrada.

A versão simplificada de uma transação é gerada com respeito a uma de suas entradas I. Nessa versão, todas as demais entradas são removidas e o *script* de I é substituído pelo *script* da saída referenciada por I. De fato, os *scripts* das entradas não podem fazer parte das versões simplificadas porque eles próprios podem conter assinaturas dessas versões. Além disso, o *script* de I é substituído pelo da saída referenciada para que toda entrada gere uma versão simplificada diferente[2]. Apesar de uma versão simplificada estar sempre atrelada a uma entrada específica, não vamos mencionar essa entrada quando ela estiver clara no contexto.

Outro tipo comum de transferência é o *Pay to Script Hash*. Nesse tipo de transferência, o *script* de uma saída precisa apenas especificar um valor *hash*. Para reivindicar essa saída, uma entrada precisa conter um *script* que execute sem erros e que tenha valor *hash* igual ao especificado. Com esse esquema de transferência é possível criar transações mais complexas, para implementar, por exemplo, contratos inteligentes. Alguns exemplos desses contratos podem ser encontrados nas documentações *online* do *Bitcoin*, como em WIKI (2015b) e FOUNDATION (2015c). Em particular, boa parte das construções propostas no decorrer deste trabalho só podem ser implementadas através da utilização desse tipo de transferência.

2.1.4 Assinatura Digital

O esquema de assinatura digital utilizado pelo *Bitcoin* é o *Elliptic Curve Digital Signature Algorithm* (Algoritmo de Assinatura Digital de Curvas Elípticas) (ECDSA). Esse esquema se baseia na chamada criptografia de curvas elípticas e, de maneira geral, alcança bons níveis de segurança com chaves relativamente pequenas.

O uso de assinaturas digitais é um dos ingredientes da *pseudo-anonimidade* que o *Bitcoin* provê aos seus usuários. De fato, o tipo mais comum de transação, o *Pay to Pubkey Hash*, identifica os emissores e receptores de transações através de seus *endereços*, que nada mais são que valores *hash* de chaves públicas do ECDSA. Uma mesma pessoa pode criar um número arbitrário de endereços. De fato, recomenda-se que os usuários criem um par de chaves, uma pública e outra secreta, para cada transação que emitirem como forma de proteger sua identidade e minimizar os danos causados por um possível roubo. Os endereços podem ser anunciados publicamente, como o fazem instituições que recebem doações em *bitcoins*, por exemplo. Já as chaves secretas devem, naturalmente, ter seu sigilo protegido.

As transações *Pay to Pubkey Hash* se baseiam na *autenticidade* que o esquema de assinatura oferece. De fato, porque o ECDSA é *inforjável*[3], uma transação que transfere *bitcoins* para o endereço $hash(pk)$ pode ser reivindicada apenas pelo portador da chave secreta sk correspondente à pk. Um atacante não consegue forjar uma assinatura feita com sk, mesmo que

[2] Isso é verdade mesmo que os *scripts* das saídas referenciadas sejam iguais. Nesse caso as versões simplificadas seriam diferenciadas apenas pela posição da entrada que é não nula.

[3] Por hipótese. A inforjabilidade do esquema depende da dificuldade do problema no qual ele se baseia, a saber, o do logaritmo discreto em curvas elípticas.

ele tenha acesso a várias dessas assinaturas válidas. Tais "exemplos de assinatura" podem estar disponíveis se *sk* já tiver sido usada antes para reivindicar outras transações. Esse tipo de ataque é o mesmo do descrito no experimento de inforjabilidade da Definição 3.28.

Outro benefício decorrente do uso de assinaturas digitais é a *integridade* das transações. Os campos das transações que são assinados, i.e., aqueles que fazem parte da versão simplificada, são "protegidos" pela assinatura gerada. Ou seja, um atacante não pode adulterá-los porque não conseguiria gerar outra assinatura válida. Isso é mais uma consequência da inforjabilidade do ECDSA.

Ressaltamos, no entanto, que não só as assinaturas digitais contribuem para a integridade das transações. Os valores *hash* das transações e a prova de trabalho calculada sobre os blocos também contribuem para isso.

A definição de esquemas de assinatura e a discussão de suas propriedades são feitas na Seção 3.9 do Capítulo 3.

2.1.5 Ponteiros *Hash* e Árvores de Merkle

Um *ponteiro* é uma estrutura de dados que referencia (ou *aponta* para) algum valor, permitindo acessá-lo de forma eficiente. Ele é especialmente utilizado para se compartilhar o acesso a alguma estrutura de dados sem a necessidade de se fazer cópias dela. Várias das estruturas utilizadas pelo *Bitcoin* fazem uso de ponteiros, mas em uma forma estendida: o *ponteiro hash*.

Um ponteiro *hash* pode ser definido como um par *(ptr, H)*, em que *ptr* é um ponteiro tradicional e H é o *hash* do valor que está sendo apontado.

No *Bitcoin* as estruturas de dados principais, como as transações e os blocos, são referenciadas a partir de seus identificadores (*ids*). Eles, por sua vez, são simplesmente o valor *hash* da estrutura correspondente. Por essa razão, as referências feitas pelas estruturas no *Bitcoin* são ponteiros *hash* (prt, H) tais que $prt = H$.

Os ponteiros *hash* possuem pelo menos uma vantagem sobre os ponteiros comuns. Com eles é possível verificar a integridade de uma estrutura apontada, com base na propriedade anticolisão da função *hash* utilizada. Ou seja, para testar se uma estrutura E é a apontada por (prt, H), pode-se calcular o *hash* H_E de E e então verificar se $H_E = H$. Caso os *hashes* coincidam, pode-se aceitar que E é a estrutura apontada por (prt, H) com alta probabilidade. Pois se E' fosse a estrutura apontada por (prt, H) e $E \neq E'$ mas $H = H_{E'} = H_E$, então o par (E, E') causaria uma colisão na função de *hash* utilizada. Se essa função for resistente à colisão, tal par só pode ser encontrado com probabilidade desprezível.

Essa propriedade dos ponteiros *hash* é particularmente útil para o *Bitcoin* porque possibilita que usuários com menos capacidade de armazenamento guardem apenas ponteiros *hash* ao invés das estruturas apontadas. Se alguma estrutura tornar-se necessária, o usuário pode requisitá-la através da rede ponto-a-ponto e conferir a integridade da estrutura recebida utilizando o *hash* contido no ponteiro armazenado.

Uma importante aplicação dos ponteiros *hash* para o *Bitcoin* é a *árvore de Merkle*. Ela é, basicamente, uma árvore binária na qual os ponteiros comuns são substituídos por ponteiros *hash*. Ela é particularmente útil porque permite que a integridade de suas *provas de pertinência* seja verificada. A prova de pertinência de um determinado valor consiste de todos os nós do caminho ligando a raiz ao nó que contém o valor em questão.

Para validar essa prova é necessário apenas (i) conferir se a raiz é válida e (ii) verificar se cada nó intermediário corresponde de fato à estrutura apontada pelo seu predecessor no caminho (que é seu pai na árvore). Para conduzir (i) o verificador precisa ter armazenado um ponteiro *hash* para a raiz da árvore. A etapa (ii) é conduzida a partir dos nós e seus ponteiros *hash* recebidos. A integridade deles segue diretamente da integridade da raiz garantida por (i). De fato, se um dos nós recebidos, que não a raiz, tivesse sido alterado, seu *hash* teria mudado e, para manter a árvore consistente, o ponteiro *hash* do seu pai teria que ser mudado também. Essa necessidade de atualização de ponteiros seria propagada até a raiz da árvore. Mas isso seria detectado, com alta probabilidade, pela etapa (i).

O tamanho dessa prova é proporcional à altura da árvore. Essa altura, por sua vez, é limitada por uma função logarítmica do número de folhas se a árvore for balanceada.

2.1.6 Blocos

Um bloco é uma estrutura de dados cuja função é armazenar transações, organizadas em uma árvore de *Merkle*. Além de uma árvore de transações, um bloco possui outros campos, como sumarizado na Tabela 2.4. Um dos que se destacam é o *nonce*, um inteiro de 32 *bits* utilizado no cálculo da prova de trabalho, como explicado na Seção 2.1.7.

Todo bloco possui uma transação especial, sempre na primeira posição da lista de transações, conhecida como *coinbase*. Ela é responsável por recolher a "recompensa" pela geração do bloco junto com quaisquer "gorjetas" dadas através das transações. Mais sobre essas recompensas será discutido na seção Seção 2.1.7.

2.1.7 Prova de Trabalho

Uma prova de trabalho é, em geral, a resposta para um problema que acredita-se ser difícil, só podendo ser resolvido com emprego de uma quantidade considerável de recursos computacionais. Provas de trabalho podem ser utilizadas, por exemplo, para evitar ataques de negação de serviço e o envio de *spam*. Para tal, é um requisito fundamental que a prova de trabalho tenha natureza *assimétrica*. Isto é, enquanto seu cálculo deve ser difícil, a verificação de sua validade precisa ser conduzida eficientemente. A prova de trabalho tem papel fundamental para a segurança do *Bitcoin*, como veremos mais adiante.

No *Bitcoin*, um bloco só pode ser considerado válido se seu *hash*, quando calculado pela função *double SHA256*,[4] for numericamente menor que um *alvo* globalmente conhecido.

[4] A função *double SHA256* é a composição da função *SHA256* com ela própria.

Tabela 2.4: Campos que compõem os blocos

Campo	Descrição
Versão	A versão das regras de validação que o bloco segue.
Id	O *hash* do bloco, calculado com a função *double SHA256* (a função *SHA256* composta com ela própria).
Quantidade de Transações	A quantidade de transações no bloco.
Lista de Transações	As transações propriamente ditas.
Bloco Anterior	O *id* do bloco imediatamente anterior (ponteiro *hash*). Esse campo cria uma estrutura acíclica chamada *blockchain*.
Raiz da Árvore de Transações	O valor *hash* da raiz da árvore de Merkle que contém as transações do bloco (ponteiro *hash*).
Estampa de Tempo	Data e hora da criação do bloco. O tempo é especificado em segundos desde a meia noite de 1/1/1970 UTC (*Coordinated Universal Time*).
Valor Alvo	Valor alvo que regula a dificuldade da prova de trabalho no momento da criação do bloco.
nonce	Campo numérico relacionado ao cálculo da prova de trabalho.

Fonte: Tabela criada pelo autor a partir de informações contidas em WIKI (2015c,d).

Essa função, por sua vez, é considerada pseudo-aleatória e, portanto, acredita-se não haver um algoritmo eficiente capaz de predizer sua saída. Dessa forma, o único algoritmo disponível para encontrar uma configuração válida para um bloco é a busca exaustiva. Isto é, o algoritmo que consiste de sucessivas modificações na estrutura do bloco até que uma delas resulte em uma configuração com *hash* válido. Dessa forma, a prova de trabalho no *Bitcoin* é a própria geração de um bloco válido[5]. Nesse caso, o recurso computacional explorado é o poder de processamento.

Mais precisamente, para se gerar uma prova de trabalho, altera-se sucessivamente o campo *nonce* de um bloco. Como esse campo é representado por 32 *bits*, esse procedimento garante 2^{32} tentativas de se encontrar uma prova de trabalho. Se nenhuma dessas tentativas for exitosa, pode-se modificar outros campos, como a estampa de tempo do bloco, e então tentar novamente a sorte com o *nonce*. O processo de se encontrar uma prova de trabalho pode ser modelado como um experimento aleatório, como argumenta NAKAMOTO (2008).

O valor alvo que regula a prova de trabalho no *Bitcoin* separa os números de 256 *bits*[6] em dois conjuntos disjuntos: (i) o composto pelos números menores que o alvo e (ii) o composto pelos demais números. Apenas blocos que possuem *hash* pertencente ao conjunto (i) podem ser considerados provas de trabalho válidas. Por isso, quanto menor o valor do alvo, mais restrito o conjunto (i) se torna e, como consequência, mais difícil é de se encontrar uma prova de trabalho.

[5]Nos referimos à validade com respeito apenas ao valor do *hash*. Naturalmente várias outras característica são necessárias para que um bloco seja considerado válido, como seu tamanho e a validade das transações nele contidas.

[6]A saída da função *double SHA256* tem tamanho fixo de 256 *bits*.

Por isso, o tamanho do conjunto (i) está diretamente relacionado com a taxa global com a qual provas de trabalho são encontradas.

O protocolo *Bitcoin* estabelece que o tamanho do conjunto (i) seja reajustado de modo que um bloco seja criado, em média, a cada 10 minutos. Esse reajuste é feito a cada 2016 blocos, levando-se em conta a proporção entre o tempo real de geração desses blocos e o tempo ideal, que é de duas semanas. O protocolo *Bitcoin* especifica ainda que o valor alvo não deve ser ajustado por um fator maior que quatro, para evitar mudanças bruscas na dificuldade da prova de trabalho.

Enquanto o valor alvo tem um influência *global* na taxa com a qual provas de trabalhos são geradas, o poder computacional de cada usuário interfere na capacidade *individual* de cálculo dessas provas. Isso decorre naturalmente do fato de que um maior poder de processamento proporciona uma também maior velocidade no cálculo de *hashes*, o que está diretamente relacionado com o cálculo de provas de trabalho.

A geração de blocos, em decorrência da prova de trabalho, exige uma série de investimentos, como a aquisição de *hardware* especializado e o uso intensivo de energia elétrica. Para incentivar os usuários a incorrer nesses gastos, o protocolo *Bitcoin* define uma recompensa para todo usuário que gerar um bloco válido. Essa recompensa, inicialmente de 50 *bitcoins*, cai pela metade a cada 210.000 blocos, o que equivale a aproximadamente quatro anos. Essa é a única maneira pela qual *bitcoins* são criados. Por essa razão, o processo de geração de blocos é conhecido como *mineração* e os usuários que o conduzem, *mineradores*.

Em adição à recompensa prevista pelo protocolo, os mineradores também podem receber gorjetas, ou taxas (*fees*), através das transações contidas no bloco minerado. A gorjeta dada por uma transação corresponde à quantidade de *bitcoins* não gasta por aquela transação[7]. Embora o pagamento de gorjetas não seja obrigatório, os mineradores são livres para decidir quando e quais transações incluir em um bloco, e o não pagamento de gorjetas influencia negativamente a prioridade dada a uma transação. A priorização de transações é descrita em WIKI (2015e). Naturalmente, com a gradual diminuição da recompensa, as gorjetas vão assumir papel cada vez mais importante no incentivo à mineração de blocos.

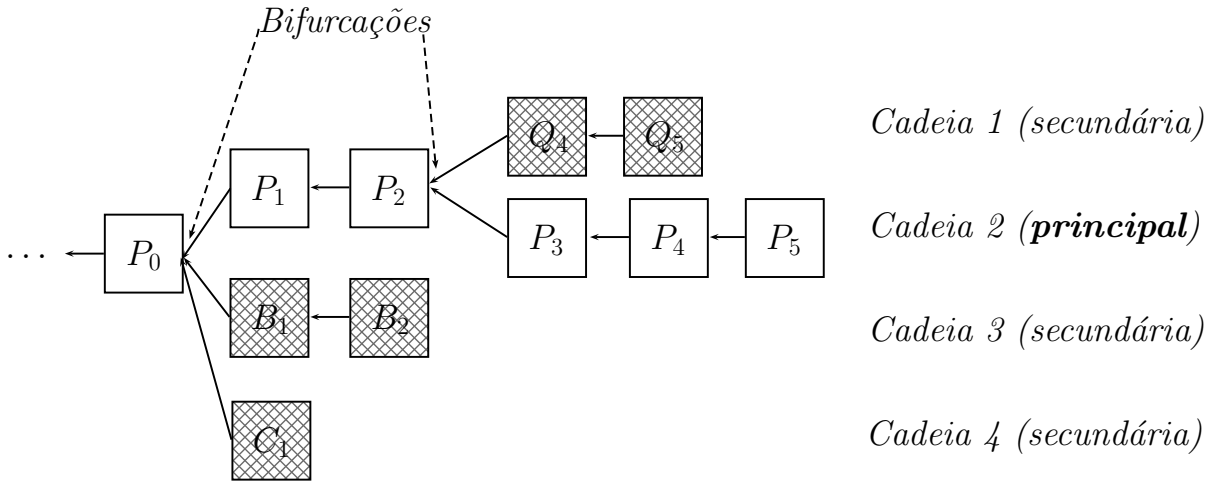
2.1.8 Juntando as Peças: *Blockchain* e Consenso

Como mostrado na Tabela 2.4, um dos campos contidos nos blocos é um ponteiro *hash* que referencia outro bloco anteriormente gerado. A estrutura de dados formada pelos blocos e seus ponteiros é chamada de *blockchain*. Ela é responsável por manter o histórico de todas as transações do *Bitcoin* e permite, por exemplo, a verificação da validade de transações e o cálculo do saldo dos usuários.

No *Bitcoin* não há uma entidade central mantenedora da *blockchain*. Ao invés disso, cada usuário possui uma cópia local, que é atualizada com os blocos recebidos através da rede

[7] Ou seja, a soma de todas as entradas menos a soma de todas as saídas.

Figura 2.1: Decomposição da *blockchain* em cadeias a partir de suas bifurcações



Fonte: Elaborada pelo autor.

ponto-a-ponto e os gerados pelo próprio usuário, caso ele seja um minerador. Essa natureza distribuída pode gerar problemas de consistência, fazendo com que diferentes versões da *blockchain* coexistam entre os usuários. Resolver esses problemas de consistência e garantir a segurança das informações na *blockchain* estão entre os principais objetivos do *Bitcoin*.

A *blockchain* é uma estrutura em forma de árvore, com diversas bifurcações (*forks*). Uma bifurcação caracteriza uma divergência, possivelmente já superada, dos usuários do *Bitcoin* sobre quais blocos deveriam estar na *blockchain*. De maneira concreta, essa divergência se manisfesta quando um mesmo bloco é referenciado por mais de um ponteiro. Ilustramos o cenário de bifurcações na *blockchain* na Figura 2.1.

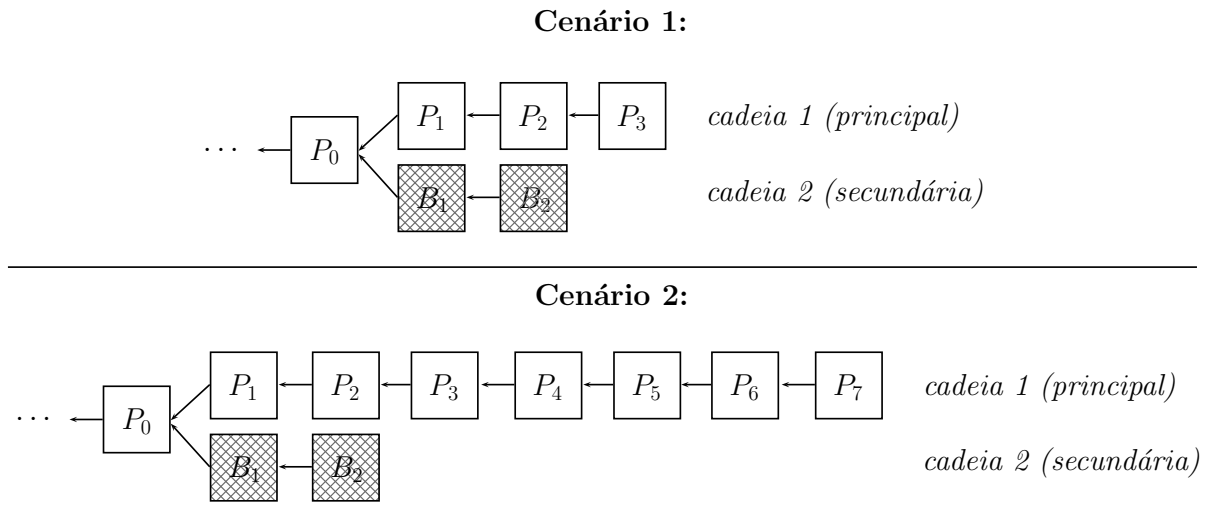
Como mostra a Figura 2.1, as bifurcações dividem a *blockchain* em várias *cadeias*. O objetivo central do protocolo *Bitcoin* é estabelecer um *consenso* entre seus participantes sobre qual dessas cadeias é a *principal*. Apenas os blocos e, consequentemente, as transações, contidos na cadeia principal são considerados válidos[8]. A regra estabelecida pelo protocolo *Bitcoin* é a de que a cadeia principal é sempre a de maior extensão. Na Figura 2.1, a cadeia 2, formada pelos blocos $P_0, P_1, P_2, P_3, P_4, P_5$, é a principal e as demais, secundárias. Diferenciamos graficamente os blocos das cadeias secundárias para explicitar que eles não são considerados válidos.

O comportamento esperado de um minerador é o de sempre adicionar seus blocos na cadeia principal, isto é, gerar blocos que referenciem o bloco mais recente da cadeia principal. Não só porque essa é a especificação do protocolo, mas principalmente porque blocos em cadeias secundárias são considerados inválidos e, portanto, não geram recompensas.

No entanto, uma cadeia secundária pode tornar-se a principal se for suficientemente estendida. Em geral, se os mineradores estão agindo honestamente, isso ocorre quando essas cadeias (a principal e a secundária) têm tamanhos muito semelhantes. Considere, por exemplo,

[8]Os blocos e transações nas cadeias secundárias são *bem-formados*, pois, em algum momento, foram publicados na *blockchain*. Mas, para todos os efeitos, é como se eles não "existissem".

Figura 2.2: Ilustração da promoção de cadeias secundárias



Fonte: Elaborada pelo autor.

o cenário 1 da Figura 2.2. Se um usuário não receber o bloco P_3, devido a um atraso na rede, por exemplo, ele pode considerar que a cadeia 2 é a principal. Dessa forma, bastaria que ele adicionasse rapidamente 2 blocos à cadeia 2 para *promovê-la* a principal.

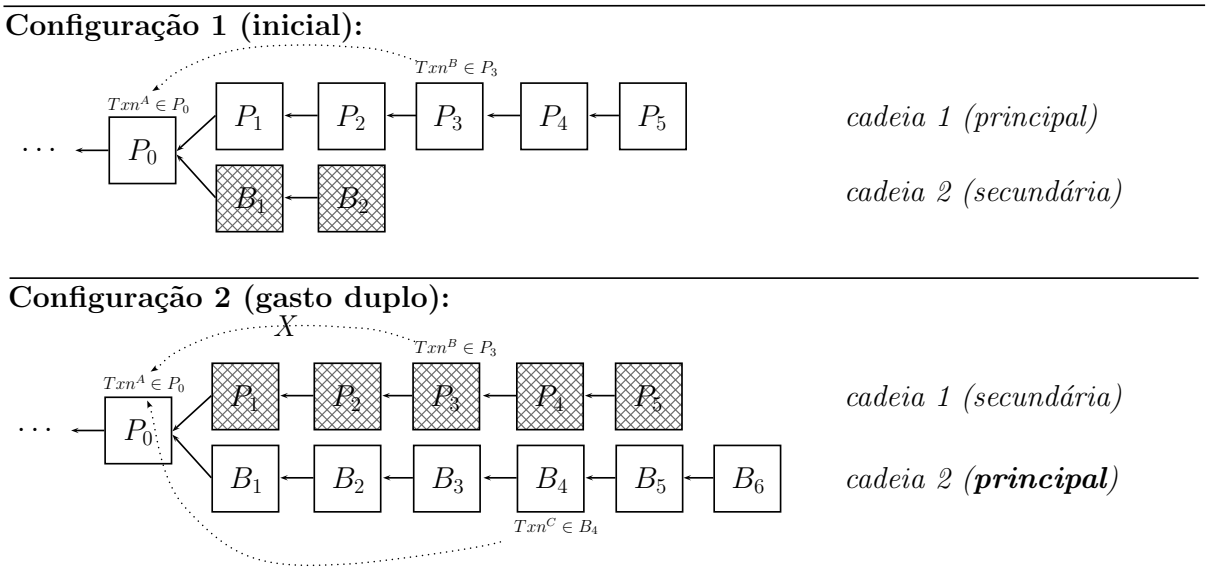
Estender uma cadeia secundária pode ser, no entanto, interesse de uma entidade maliciosa. Uma possível motivação para isso é a de invalidar transações contidas na cadeia principal, causando o problema conhecido como gasto-duplo (*double-spending*). Ele ocorre quando uma mesma transação é reivindicada mais de uma vez, como se os *bitcoins* disponibilizados em suas saídas fossem duplicados.

Exemplificamos o gasto-duplo na Figura 2.3. Inicialmente, na configuração 1, temos que a transação Txn^B reivindica os *bitcoins* da transação Txn^A. Ambas estão na cadeia principal e os blocos B_1 e B_2, na secundária (sendo, portanto, inválidos).

Considere agora que um atacante tenha forçado o cenário da configuração 2 a ocorrer. Isto é, ele minerou blocos de maneira suficientemente rápida e promoveu a cadeia 2 a principal. Nesse caso, os blocos de P_1 à P_5 se tornam inválidos. Em particular, P_3 se torna inválido e, com ele, a transação Txn^B. Com isso, torna-se possível a publicação da transação Txn^C, que também reivindica Txn^A. Ou seja, tudo se passa como se os *bitcoins* de Txn^A tivessem sido gastos duas vezes. Em particular, a transação Txn^B, que foi *revertida*, poderia ser o pagamento de um serviço já realizado. Nesse caso, o prestador do serviço, destinatário de Txn^B, fica no prejuízo.

O problema do gasto-duplo está diretamente ligado ao consenso estabelecido sobre quais blocos devem formar a *blockchain*. De fato, se fosse fácil manipular esse consenso, promovendo cadeias secundárias arbitrariamente, então o *Bitcoin* seria totalmente vulnerável ao gasto-duplo. No entanto, esse consenso não é manipulado facilmente. Quanto mais "certeza" os usuários têm de que um determinado bloco deve pertencer à cadeia principal, mais difícil é promover uma cadeia secundária que exclua esse bloco.

Figura 2.3: O problema do gasto-duplo



Fonte: Elaborada pelo autor.

Para ilustrar isso, vamos utilizar o cenário 2 da Figura 2.2 como exemplo. Para que a cadeia 2 se iguale em tamanho à cadeia principal, é necessário que seja estendida em 6 blocos. Isso significa que o atacante precisa gerar 6 provas de trabalho e, portanto, está sujeito à limitação imposta pelo valor alvo, como discutido na Seção 2.1.7. Enquanto isso, o restante da rede continua estendendo a cadeia 1. Por essa razão, o atacante só teria probabilidade não desprezível de ser bem sucedido se sua taxa de geração de provas de trabalho fosse superior à taxa do restante da rede *em conjunto*.

O objetivo do exemplo dado é ressaltar que a probabilidade de um bloco deixar a cadeia principal da *blockchain* diminui na medida em que o tempo passa e sua profundidade (na *blockchain*) aumenta. Note que a profundidade é essencialmente o que difere os cenários 1 e 2 da Figura 2.2 e é também o que torna um possível e outro improvável.

Segundo a modelagem feita em NAKAMOTO (2008), a probabilidade de um bloco deixar a cadeia principal diminui *exponencialmente* de acordo com o aumento da quantidade de provas de trabalho que precisam ser calculadas. Por isso, cada bloco adicionado a uma cadeia é chamado de *confirmação*, pois ele funciona como um "voto" para confirmar determinada cadeia como a principal. Na prática, a comunidade *Bitcoin* considera desprezível a probabilidade de um bloco deixar a cadeia principal se ele recebeu pelo menos 6 confirmações, como descrito em WIKI (2015f)[9].

Outro risco a que o consenso formado sobre a *blockchain* precisa resistir é o da *maleabilidade* dos blocos. Se fosse possível adulterar um bloco da cadeia principal, ou de qualquer outra cadeia secundária, sem que ninguém percebesse, o *Bitcoin* seria inseguro. Isso permitiria, por exemplo, que um atacante modificasse uma transação já aceita pelos participantes do *Bitcoin*

[9]Essa análise leva em conta um adversário com menos de 10% do poder computacional da rede.

sem levantar suspeitas. Mais uma vez, é a dificuldade da prova de trabalho utilizada pelo *Bitcoin* que previne esse tipo de ataque. De fato, como a função *SHA256* é considerada pseudo-aleatória, qualquer modificação em um bloco, mesmo que de apenas alguns *bits*, tem um grande impacto no seu valor *hash*. Consequentemente, é esperado que qualquer modificação feita em um bloco o torne uma prova de trabalho inválida.

Com isso, um atacante precisa recalcular a prova de trabalho do bloco B que ele pretende adulterar. Mas não *apenas* dele. Como é preciso atualizar o ponteiro *hash* do bloco B_{prox} que o referencia, o próprio *hash* do bloco B_{prox} muda e, com isso, ele também perde sua validade como prova de trabalho. Ou seja, a necessidade de recálculo de provas de trabalho é propagada. De fato, todos os blocos desde B até o mais recente da cadeia principal precisam ter suas provas de trabalho recalculadas. Podemos, com isso, tirar duas conclusões. A primeira delas é que não se pode fazer alterações *locais* em um bloco sem propagá-las. A outra é que, devido a essa propagação, a adulteração de um bloco equivale ao ataque anteriormente citado de se promover uma cadeia secundária a principal. Ou seja, adulterar um bloco B, transformando-o em B', significa promover a primária a cadeia da *blockchain* iniciada em B'. Dessa forma, esse ataque possui as mesmas dificuldades argumentadas anteriormente. Isto é, a probabilidade de se adulterar um bloco B com sucesso cai exponencialmente com o aumento da quantidade de confirmações recebidas por B.

2.2 O Problema da Maleabilidade das Transações

Uma fragilidade presente no protocolo *Bitcoin* é a chamada *maleabilidade das transações*. Ela consiste na possibilidade de se criar duas transações *semânticamente* equivalentes mas com *ids* diferentes. O *id* de uma transação é o valor *hash* calculado sobre seus campos, incluindo, em particular, suas entradas. Isso contrasta, por exemplo, com o modo pelo qual as assinaturas adicionadas nas entradas são geradas. Como mencionamos, tais assinaturas são geradas sobre a versão simplificada da transação, a qual não possui todas as entradas.

Para entender o efeito da maleabilidade das transações, considere o seguinte cenário. Alice e Bob firmam um contrato no qual Alice se compromete a fazer um pagamento a Bob depois de 1 mês. Para garantir que esse pagamento vai acontecer, Bob pede que Alice crie a transação $T_{A\to B}$ que transfere a quantia combinada para ele. Essa transação só pode ser reivindicada através da apresentação de duas assinaturas: uma de Alice e outra de Bob. Bob então cria uma transação T_B e pede para Alice assiná-la. Ela então confere se o campo *locktime* está configurado corretamente e, se estiver, gera sua assinatura.

De fato, T_B garante a Alice que Bob não vai reivindicar $T_{A\to B}$ antes do prazo determinado, pois Bob não pode alterar o *locktime* de T_B sem invalidar a assinatura de Alice. Isso porque o campo *locktime* faz parte da versão simplificada da transação. Por isso, Alice precisa apenas confiar no funcionamento do *Bitcoin*, e não em Bob.

Por outro lado, Bob pode ser prejudicado por Alice. Imagine que Alice modifique uma

entrada de $T_{A\to B}$ adicionando instruções que não lhe alterem o funcionamento[10] *depois de* T_B *ter sido criada*. Ao mudar uma entrada, Alice muda também o *id* de $T_{A\to B}$. Mas note que T_B usa esse *id* para referenciar $T_{A\to B}$. Note também que Bob não pode simplesmente corrigir o *id* porque isso invalidaria a assinatura de Alice em T_B. A conclusão é que Bob não poderia mais reivindicar $T_{A\to B}$.

Esse cenário, no entanto, pode receber uma complicação extra. Imagine que Alice não tenha a intenção de prejudicar Bob. Ela publica a transação $T_{A\to B}$ do jeito que foi criada. No entanto, antes de aparecer na *blockchain*, essa transação é publicada na rede *Bitcoin*, onde será recebida por mineradores, adicionada em algum bloco, o qual terá sua prova de trabalho calculada. Só então ela aparecerá na *blockchain*. Nesse meio tempo, a maleabilidade da transação $T_{A\to B}$ poderia ser explorada por uma terceira parte externa, Charle, por exemplo. Ele poderia alterar uma das entradas de $T_{A\to B}$ criando $T'_{A\to B}$ e invalidando a transação T_B.

Esses cenários em que a maleabilidade das transações é explorada podem parecer, à primeira vista, "artificiais". Isso porque eles podem ser evitados se Bob, por exemplo, exigisse que a transação criada por Alice aparecesse na *blockchain* para só então criar T_B. De fato, para a maior parte dos usuários do *Bitcoin*, que utilizam as transações da maneira mais comum, a maleabilidade não deveria ser um problema. Como mostrado em ANDRYCHOWICZ et al. (2015), esses usuários só são atingidos pelo problema da maleabilidade se o *software* utilizado por eles para se conectar com o *Bitcoin* for defeituoso.

Para continuar nossa discussão, vamos considerar três aspectos que contribuem para a maleabilidade das transações:

1. O *id* de uma transação é calculado sobre todos os seus campos, inclusive as entradas,
2. Uma transação T_B referencia uma transação $T_{A\to B}$ que ainda não apareceu na *blockchain*,
3. A transação T_B precisa de uma assinatura feita com uma chave secreta não acessível a seu criador e sobre um conteúdo *mutável*.

Como mencionamos, os usuários mais comuns do *Bitcoin* não são afetados pela maleabilidade das transações porque evitam o aspecto 2. No entanto, esse aspecto é essencial para a criação de contratos utilizando o *Bitcoin*. De fato, várias das transações que serão apresentadas no Capítulo 6 referenciam outras ainda não presentes na *blockchain*.

A seguir, apresentaremos três propostas de solução para o problema da maleabilidade. As duas primeiras atacam o aspecto 1, enquanto que a última se concentra no aspecto 3.

2.2.1 Proposta 1: ANDRYCHOWICZ et al. (2014a)

Uma solução relativamente simples para o problema da maleabilidade das transações é proposta por ANDRYCHOWICZ et al. (2014a). Como vimos, esse problema decorre do fato de

[10] Como uma instrução *push* seguida de uma *pop*. Ou Alice poderia simplesmente assinar a transação novamente, gerando outra assinatura igualmente válida.

que o *id* das transações é calculado sobre *todos* os campos da transação, inclusive suas entradas. Então uma solução é fazer com que o *id* de uma transação seja calculado sobre sua versão simplificada, da mesma maneira com que as assinaturas são calculadas. Dessa forma, toda vez que alguma modificação alterar o *id* de uma transação, ela também vai invalidar as assinaturas da transação.

2.2.2 Proposta 2: KUMARESAN e BENTOV (2014)

Como argumentado por KUMARESAN e BENTOV (2014), a Proposta 1 pode acarretar em uma perda de expressividade que as transações atualmente possuem. Isso porque, de acordo com essa proposta, os *ids* das transações deixariam, naturalmente, de refletir as informações contidas nas entradas, as quais podem ser importantes para a criação de contratos.

Para resolver o problema, KUMARESAN e BENTOV (2014) propõem adotar a Proposta 1 parcialmente. Ou seja, as transações passariam, na prática, a possuir dois *ids*. O calculado sobre sua versão simplificada serviria para ser referenciado por outras transações, como proposto em ANDRYCHOWICZ et al. (2014a). Já o outro identificador, calculado sobre toda a transação, como ocorre atualmente, seria utilizado para referenciar a transação na árvore de Merkle que a contém dentro de um bloco.

2.2.3 Proposta 3: ANDRYCHOWICZ et al. (2015)

A modificação da proposta 3 não afeta o *Bitcoin*, mas sim a maneira pela qual os contratos são escritos.

Vamos considerar o exemplo das transações $T_{A\to B}$ e T_B dado anteriormente. A proposta de ANDRYCHOWICZ et al. (2015) é remover a necessidade de se ter a assinatura de Alice em T_B. Mais especificamente, $T_{A\to B}$ deve ser reivindicada mediante a apresentação de uma assinatura de Bob e um valor *secreto* conhecido por Alice.

Inicialmente, para criar $T_{A\to B}$, Alice escolhe aleatoriamente um valor s e entrega para Bob o valor $h \stackrel{def}{=} H(s)$, em que H é uma função *hash*. Para reivindicar essa transação, Bob precisa colocar em T_B uma assinatura sua e o valor s, que ele ainda não conhece. Como o acordo entre eles é que $T_{A\to B}$ só pode ser reivindicada após 1 mês, Alice precisa esperar esse tempo para entregar s para Bob. Note que assim que Bob conhecer s ele poderá reivindicar $T_{A\to B}$[11].

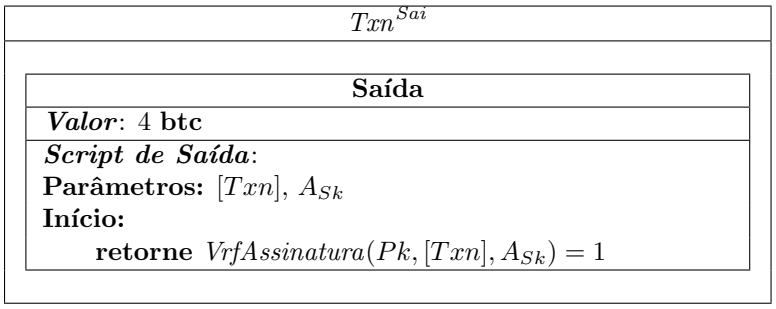
O problema da maleabilidade é resolvido porque $T_{A\to B}$ pode ser prontamente publicada na *blockchain*. Nesse caso, T_B não precisa referenciar uma transação "escondida". A preocupação de Bob agora é outra: como garantir que Alice vai de fato revelar o valor s correto?

Para resolver isso, os autores se baseiam em uma construção apresentada por eles em ANDRYCHOWICZ et al. (2014b). De maneira geral, eles propõem a utilização de um esquema de comprometimento baseado no *Bitcoin*[12]. Ele funciona como a seguir. Alice cria uma transação

[11] A validade dessa ideia se baseia na propriedade de não-colisão de H, a qual garante que é *difícil* encontrar $s' \neq s$ tal que $H(s') = H(s)$. Apresentamos esse conceito na Definição 3.22.

[12] Apresentamos a ideia de esquema de comprometimento na Definição 3.14.

Figura 2.4: Exemplo de transação com uma saída



Fonte: Elaborada pelo autor.

Commit que transfere a mesma quantidade de moedas que $T_{A \to B}$ e pode ser reivindicada de duas maneiras:

1. Apresentando-se uma assinatura de Alice e o valor *s* que gera *h*. Ou seja, Alice coloca o valor *h* no *script* de saída de *Commit*. Isso serve para Alice se *comprometer* com *s*,

2. Apresentando-se uma assinatura de Alice e outra de Bob.

A ideia é que Alice reivindique *Commit* em até 1 mês utilizando o método 1. Dessa forma Bob receberá *s* e conseguirá reivindicar $T_{A \to B}$. Se Alice não o fizer, então Bob reivindicará *Commit* através do método 2. Ele não estará apto a reivindicar $T_{A \to B}$ mas receberá a mesma quantidade de moedas.

Para que esse esquema funcione, Alice cria uma transação *Fuse* que referencia *Commit*, coloca sua assinatura nela e a envia para Bob. O campo *locktime* de *Fuse* deve ser superior a 1 mês. Através de *Fuse* Bob poderá reivindicar *Commit*. Note que *Fuse* não sofre com a maleabilidade porque referencia uma transação já presente na *blockchain* (*Commit*).

Com esse esquema, Bob consegue detectar intenções desonestas de Alice, caso *Commit* ou $T_{A \to B}$ não sejam publicadas, *Fuse* não seja recebida ou qualquer uma dessas transações estiverem incorretas. Mas, caso essas transações sejam criadas corretamente, Bob terá a garantia de receber suas moedas e Alice, a de que a transferência só ocorrerá no tempo acordado.

2.3 Notação e Nomenclatura

Ao longo deste trabalho, precisaremos lidar apenas com um subconjunto das funcionalidades do *Bitcoin*. Mais especificamente, teremos que representar transações, com suas entradas, saídas e, quando necessário, seus campos *locktime*. Representaremos as transações como nas Figuras 2.4, 2.5 e 2.6.

Sempre incluiremos um nome na representação de uma transação para facilitar a referência textual. Isso é equivalente a fazer referência a uma transação por meio de seu campo *id*.

Figura 2.5: Exemplo de transação com uma entrada

Txn^{Ent}
Entrada
Saída Referenciada: a única saída de Txn^{Sai}
Script de Entrada: **Início:** **forneça** $[Txn^{Ent}], Assina(Sk, [Txn^{Ent}])$ para a saída referenciada.

Fonte: Elaborada pelo autor.

Uma parte que merece atenção especial na representação de uma transação é a implementação dos *scripts* de suas entradas e saídas. Fizemos a opção por apresentar esses *scripts* na forma de um pseudo-código ao invés de utilizar uma linguagem semelhante à definida pelo *Bitcoin*. Pretendemos, com isso, tornar a apresentação mais legível e intuitiva, poupando o leitor da necessidade de interpretar programas escritos em uma linguagem baseada em pilha. Essa decisão aproxima nossa apresentação daquela contida em ANDRYCHOWICZ et al. (2014a).

Com o objetivo de mostrar como as saídas e seus *scripts* serão representados, ilustramos, na Figura 2.4, a transação Txn^{Sai}. Ela possui uma única saída que disponibiliza 4 *bitcoins* (**btc**) sob a condição de que seu *script* seja satisfeito. Na nossa representação isso significa que o *script* precisa retornar 1 ao fim de sua execução. De maneira geral, os *scripts* das saídas retornam o resultado de uma operação lógica calculada sobre seus *parâmetros* e sobre constantes introduzidas no código no momento da criação da transação.

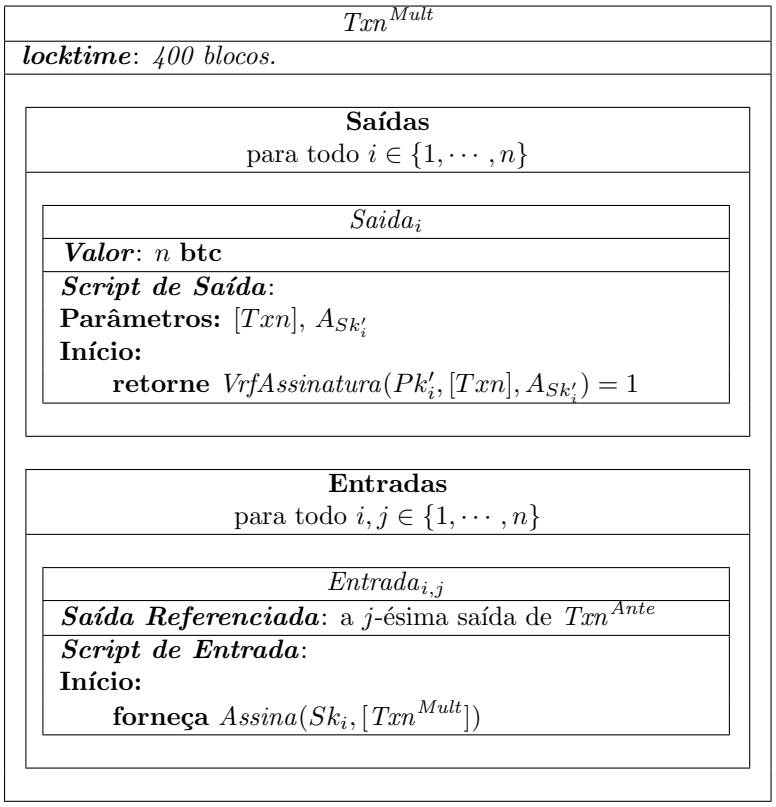
Um parâmetro comumente presente nos *scripts* das saídas é a versão simplificada da transação que reivindica essa saída. Denotamos a versão simplificada de uma transação Txn por $[Txn]$.

Como mencionamos anteriormente, um dos tipos mais comuns de transações é aquele em que a transferência de *bitcoins* é feita para o portador de uma determinada chave secreta[13] *Sk*. Para que um usuário comprove ser tal portador, ele precisa criar uma transação T contendo uma assinatura digital A_T gerada com *Sk* sobre $[T]$. Por essa razão, é comum que nos *scripts* das saídas se precise verificar a validade de uma assinatura. Para isso utilizamos a função *VrfAssinatura*. Adiantando a apresentação da Definição 3.27, definimos que a função *VrfAssinatura* recebe como entrada a chave pública *Pk* (correspondente à *Sk*), a mensagem original $[T]$ e uma possível assinatura A_T e retorna 1 se a assinatura for válida e 0 caso contrário.

Juntando as peças até aqui, podemos entender que a transação Txn^{Sai} da Figura 2.4 transfere 4 **btc** para o portador da chave secreta correspondente à chave pública *Pk*. A origem desses *bitcoins* não é especificada porque omitimos a entrada (ou entradas) de Txn^{Sai}. Sempre que fizermos isso será para deixar claro que não impomos restrições sobre a origem dos *bitcoins*

[13] É o *Pay to Pubkey Hash*.

Figura 2.6: Exemplo de transação com múltiplas entradas e saídas



Fonte: Elaborada pelo autor.

de uma transação.

Na Figura 2.5 representamos uma transação que reivindica Txn^{Sai}. Isso fica claro a partir da referência feita à saída de Txn^{Sai} na única entrada de Txn^{Ent}. O *script* dessa entrada fornece os argumentos necessários e na ordem correta para satisfazer o *script* da saída de Txn^{Sai}. Um desses argumentos é uma assinatura da versão simplificada da própria Txn^{Ent} feita com a chave secreta Sk. Para gerar essa assinatura, usamos a função *Assina*. Mais uma vez, adiantando a Definição 3.27, definimos que a função $Assina(V, Sk)$ serve para gerar uma assinatura válida de V feita com Sk. Ou seja, se Pk é a chave pública correspondente a Sk, $VrfAssinatura(V, Pk, Assina(V, Sk)) = 1$.

De maneira semelhante ao que fizemos com as entradas de Txn^{Sai}, omitimos as saídas de Txn^{Ent}. Com isso queremos deixar claro que não impomos restrições sobre como Txn^{Ent} disponibiliza os *bitcoins* que reivindica.

Quando uma transação contiver várias entradas ou, de maneira semelhante, várias saídas, elas possuirão índices para facilitar a referência textual. A representação de uma transação com múltiplas entradas e saídas é ilustrada na Figura 2.6. Definimos nos títulos das entradas e das saídas quais valores os índices podem assumir.

A transação Txn^{Mult} da Figura 2.6 possui n^2 entradas. A $Entrada_{i,j}$, para todo $i, j \in \{1, \cdots, n\}$, reivindica *bitcoins* da j-ésima saída de uma transação Txn_i^{Ante} através do fornecimento de uma assinatura gerada com a chave secreta Sk_i. Esses *bitcoins* são disponibilizados em n

saídas. Cada $Saida_i$ disponibiliza *n bitcoins* para o portador de uma chave secreta correspondente à chave pública Pk'_i.

Como o campo *locktime* de Txn^{Mult} tem valor igual a 400, essa transação só pode ser publicada quando a *blockchain* contiver, pelo menos, 400 blocos. Quando esse campo for omitido, a transação pode ser prontamente publicada.

Por fim, ressaltamos que o objetivo da notação apresentada aqui não é ser *completa*, no sentido de abranger todas as possibilidades de transações do *Bitcoin*. Ela foi pensada para ser simples, mas expressiva o suficiente para representar as variações de transações apresentadas no decorrer deste trabalho.

3 DEFINIÇÕES

Neste capítulo definiremos conceitos importantes que serão utilizados ao longo deste trabalho. Nosso objetivo não é apresentar uma discussão profunda e exaustiva de cada um deles, mas apenas registrar o necessário para o entendimento de futuras referências. Por essa razão, não nos preocupamos em descrever realizações práticas nem em demonstrar formalmente certas propriedades.

As definições apresentadas neste capítulo foram adaptadas de GOLDREICH (2004), GOLDREICH (2001), KATZ e LINDELL (2014) e BENTOV e KUMARESAN (2014). Deixamos explícita a fonte de cada definição.

3.1 Funções Desprezíveis

Uma função é desprezível (*negligible*) se seu valor é limitado, em módulo, pelo inverso de um valor polinomialmente ligado à sua entrada. Esse tipo de função é extensamente utilizado para limitar superiormente outras funções, expressando o qual pequeno são seus valores.

Definição 3.1 (Função desprezível)**.** *Uma função $\mu : \mathbb{N} \to \mathbb{R}$ é desprezível se, para todo inteiro positivo $c > 1$, existe um inteiro N_c tal que, para todo $x > N_c$:*

$$|\mu(x)| < \frac{1}{x^c}$$

Dizemos que uma *probabilidade é desprezível* se a função que a calcula é limitada por uma função desprezível. O fato de uma probabilidade ser desprezível ressalta o quão raramente um determinado evento ocorre.

3.2 Indistinguibilidade

A noção de *indistinguibilidade* é fundamental para o projeto e análise de segurança de várias construções criptográficas. A ideia geral para se determinar se dois objetos são indistinguíveis é conduzir uma espécie de experimento.

Nele, consideramos um procedimento para distinção (e.g., um algoritmo ou circuito) que recebe um objeto e dá como resultado um valor binário. O experimento consiste então em alimentar esse procedimento com o primeiro objeto e depois com o segundo. Dizemos que o procedimento foi *bem sucedido* em distinguir os objetos se as suas saídas foram distintas. Ou seja, se para o primeiro objeto ele deu como resultado o valor b e para o segundo, $1-b$, onde b é um *bit*. De maneira mais geral, se o procedimento é probabilístico, então ele é bem sucedido se suas saídas forem distintas com alta probabilidade.

Dizemos então que dois *objetos* são indistinguíveis se nenhum procedimento *eficiente* pode ser bem sucedido no experimento definido acima com probabilidade *não desprezível*.

Dizemos que um procedimento é eficiente se ele executa em tempo polinomial no tamanho de sua entrada (ou de algum outro parâmetro explicitamente definido).

Um *objeto*, em geral, pode ser definido como um *ensemble* de variáveis aleatórias indexado por algum conjunto contável. Registramos essa definição a seguir.

Definição 3.2 (*Ensemble* probabilístico - Definição 3.2.1 de GOLDREICH (2001))**.** *Seja I um conjunto contável de índices. Um* ***ensemble indexado por*** *I é uma sequência de variáveis aleatórias indexadas por I. Em símbolos, se para todo $i \in I$, X_i é uma variável aleatória, então $X = \{X_i\}_{i \in I}$ é um ensemble indexado por I.*

A definição a seguir formaliza a noção de indistinguibilidade que discutimos.

Definição 3.3 (Indistinguibilidade por circuitos de tamanho polinomial - Definição 3.2.7, variação 2, de GOLDREICH (2001))**.** *Seja S um conjunto contável de cadeias. Dois objetos (ensembles indexados por S) $X \stackrel{def}{=} \{X_w\}_{w \in S}$ e $Y \stackrel{def}{=} \{Y_w\}_{w \in S}$ são* ***indistinguíveis por circuitos de tamanho polinomial****, ou simplesmente* ***indinstinguíveis****, se, para toda família $\{C_n\}_{n \in \mathbb{N}}$ de circuitos de tamanho polinomial, todo polinômio positivo $p(\cdot)$ e todo w suficientemente grande é verdade que:*

$$|Pr[C_{|w|}(X_w) = 1] - Pr[C_{|w|}(Y_w) = 1]| < \frac{1}{p(|w|)}$$

.

Por fim, vamos definir a notação que utilizaremos ao longo deste trabalho.

Definição 3.4 (Notação de Indinstinguibilidade)**.** *Se X, Y são dois ensembles indistinguíveis por circuitos de tamanho polinomial, conforme a Definição 3.3, dizemos apenas que eles são* ***indistinguíveis*** *e denotamos essa relação por $X \stackrel{c}{\equiv} Y$.*

3.3 Funções Unidirecionais, Arapucas Aprimoradas e Predicados Núcleo-duro

Uma função é dita *unidirecional* se ela é fácil de calcular mas difícil de inverter. Uma função *arapuca* (*trapdoor*), por sua vez, é um tipo especial de função unidirecional que pode ser facilmente invertida com o auxílio de um valor especial chamado "arapuca". Sem ele, no entanto, ainda é difícil invertê-la.

A dificuldade de inversão de uma função arapuca se baseia na incapacidade de qualquer procedimento (eficiente) em invertê-la com acesso apenas a um elemento aleatoriamente escolhido do seu contradomínio. Uma versão mais forte dessa noção é aquela em que a dificuldade de inversão se mantém mesmo que a fonte de aleatoriedade utilizada nessa amostragem do contradomínio seja conhecida. Nesse caso, essa função é chamada de arapuca *aprimorada*.

Para formalizar os conceitos discutidos acima, vamos começar definindo o que são funções unidirecionais.

Definição 3.5 (Funções unidirecionais - Definição 2.2.1 de GOLDREICH (2001))**.** *Uma função $f : \{0,1\}^* \to \{0,1\}^*$ é **unidirecional** se as seguintes condições são respeitadas:*

(i) (Facilidade na computação:) Existe um algoritmo (determinístico) de tempo polinomial A que, alimentado com x, computa $f(x)$. Isto é, $A(x) = f(x)$.

(ii) (Dificuldade de inversão:) Para todo algoritmo probabilístico de tempo polinomial A', todo polinômio positivo $p(\cdot)$ e todo $n \in \mathbb{N}$ suficientemente grande:

$$Pr[A'(f(U_n), 1^n) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}$$

em que U_n é um valor uniformemente escolhido de $\{0,1\}^n$.

Ressaltamos que, na Definição 3.5, a função f não é, necessariamente, injetora. Por essa razão, no item (ii), admite-se que $f^{-1}(\cdot)$ é um *conjunto* (não necessariamente unitário) de valores. De agora em diante, no entanto, as definições apresentadas tratarão apenas de *permutações*, ou seja, funções bijetoras.

Seguindo a apresentação de GOLDREICH (2001), vamos definir uma permutação arapuca de maneira diferente da qual definimos funções unidirecionais. A ideia geral é "simular" uma permutação arapuca a partir de um conjunto infinito de permutações e alguns algoritmos que operam sobre esse conjunto. A motivação para essa mudança na apresentação é puramente *prática*, ou seja, esse tipo de formulação alternativa facilita a criação de construções práticas.

Definição 3.6 (Coleção de permutações arapuca[1])**.** *Uma **coleção de permutações arapuca**, ou simplesmente **permutação arapuca**, é uma coleção de permutações $\{f_\alpha : D_\alpha \to D_\alpha\}$ acompanhada de quatro algoritmos probabilísticos de tempo polinomial (I, S, F, F^{-1}) tais que as seguintes condições se verificam:*

(i) (Facilidade de geração da arapuca:) O algoritmo I, sobre entrada 1^n, gera o par (α, τ). O valor α tem n bits e é o índice de uma permutação (f_α). Já τ é uma arapuca para f_α.

(ii) (Facilidade de amostragem:) O algoritmo S, sobre a entrada α, faz uma amostragem do domínio de f_α, retornando um elemento aleatoriamente escolhido.

(iii) (Facilidade de computação:) Dado um índice α e um valor $x \in D_\alpha$, o algoritmo F computa f_α. Isto é, $F(\alpha, x) = f_\alpha(x)$.

*(iv) (Facilidade de inversão **com** arapuca:) O algoritmo F^{-1} inverte f_α. Isto é, se (α, τ) é o par gerado por $I(1^n)$ e $y \in D_\alpha$, então $F^{-1}(\tau, y) = f_\alpha^{-1}(y)$.*

[1] Definição adaptada da Definição 2.4.4 de GOLDREICH (2001) e do Anexo C de GOLDREICH (2004)

*(v) (Dificuldade de inversão **sem** arapuca:) Para todo algoritmo probabilístico A' de tempo polinomial, todo polinômio positivo $p(\cdot)$ e todo $n \in \mathbb{N}$ suficientemente grande:*

$$Pr[A'(\alpha, f_\alpha(S(\alpha))) = S(\alpha)] < \frac{1}{p(n)}$$

em que α é o primeiro valor do par gerado por $I(1^n)$.

Os algoritmos (I, S, F, F^{-1}) podem ser transformados em determinísticos. Para isso, basta que a fonte de aleatoriedade utilizada por eles seja explicitamente recebida como mais uma entrada. Por exemplo, I passaria a ser invocado como $I(r, 1^n)$ e S, como $S(r, \alpha)$, com r gerado aleatoriamente.[2]

O item (v) da Definição 3.6 caracteriza a dificuldade de inversão de uma permutação arapuca. Um algoritmo "inversor" A' tem acesso ao índice α da permutação e do valor $f_\alpha(S(\alpha))$ escolhido *aleatoriamente* de seu contradomínio pelo algoritmo S. Esse algoritmo S, como mencionamos na Definição 3.6, pode ser tornado determinístico se sua fonte de aleatoriedade r for explicitada. Se nenhum algoritmo A' for capaz de inverter f_α tendo acesso a r como informação adicional, então dizemos que a coleção de permutações é *aprimorada*. Registramos essa definição a seguir.

Definição 3.7 (Coleção de permutações arapuca aprimoradas - Definição C.1.1 de GOLDREICH (2004))**.** *Seja $f_\alpha : D_\alpha \to D_\alpha$ e (I, S, F, F^{-1}) uma permutação arapuca como na Definição 3.6. Chamamos essa coleção de **coleção de permutações arapuca aprimoradas**, ou simplesmente de **permutação arapuca aprimorada**, se o item (v) da Definição 3.6 for substituído pelo seguinte:*

*(v) (Dificuldade de inversão **sem** arapuca:) Para todo algoritmo probabilístico A' de tempo polinomial, todo polinômio positivo $p(\cdot)$ e todo $n \in \mathbb{N}$ suficientemente grande:*

$$Pr[A'(\alpha, r, f_\alpha(S'(r, \alpha))) = S'(r, \alpha)] < \frac{1}{p(n)}$$

em que α é o primeiro valor do par gerado por $I(1^n)$, S' é o algoritmo S com sua fonte de aleatoriedade explícita e r é um valor aleatoriamente gerado com a mesma distribuição de S.

Vimos que, dado uma função unidirecional f e um valor $f(x)$ em seu contradomínio, é difícil se recuperar x.[3] No entanto, não há qualquer restrição à obtenção de informações *parciais* sobre x. Considere, por exemplo, a função $g(x) \stackrel{def}{=} (x \; mod \; 2)$. Embora seja difícil invertê-la, é possível aprender um *predicado* de x com acesso apenas a $g(x)$: a sua paridade.

[2] Utilizamos essa notação modificada no Capítulo 4.

[3] Ou qualquer valor x' com $f(x') = f(x)$ se f não for injetora

Um *predicado* de uma função é uma informação *binária* calculada sobre os elementos de seu domínio. Dizemos que um predicado b de uma função f é *núcleo-duro* se é fácil calcular $b(x)$ dado x mas difícil se apenas $f(x)$ for dado. Apresentamos a definição formal a seguir.

Definição 3.8 (Predicado núcleo-duro - Definição 2.5.1 de GOLDREICH (2001))**.** *Um predicado computável em tempo polinomial $b : \{0,1\}^* \to \{0,1\}$ de uma função f é* ***núcleo-duro*** *se, para todo algoritmo probabilístico A' de tempo polinomial, todo polinomial positivo $p(\cdot)$ e todo $n \in \mathbb{N}$ suficientemente grande:*

$$Pr[A'(f(U_n)) = b(U_n)] < \frac{1}{2} + \frac{1}{p(n)}$$

em que U_n é um valor uniformemente escolhido do conjunto $\{0,1\}^n$.

Naturalmente a probabilidade de se adivinhar o valor de um predicado é pelo menos $\frac{1}{2}$, já que ele pode assumir apenas dois valores: 0 ou 1. Mas, se o predicado é núcleo-duro, então essa probabilidade aumenta, no máximo, de maneira desprezível ($+\frac{1}{p(n)}$).

3.4 Sistemas de Provas Fortes de Conhecimento

3.4.1 Visão Geral

Um *sistema de prova* permite que um *provador* convença um *verificador* de que uma determinada afirmação é verdadeira. Mais especificamente, se R_L é uma relação binária e L uma linguagem definida como $L \stackrel{def}{=} \{x \mid \exists y \text{ t.q } (x,y) \in R_L\}$, então um sistema de prova permite que afirmações do tipo $x \in L$ sejam provadas. A relação R_L é chamada de *relação testemunha* e y, de *testemunha*.

Um *prova* pode ser encarada como uma interação entre os computadores P (provador) e V (verificador). A saída de V (e.g., 0 ou 1) é o que determina se a prova foi aceita ou não.

Dizemos que uma prova é de *conhecimento zero* se, depois de interagir com P, o verificador V não "adquire nenhum conhecimento adicional". De maneira mais formal, isso significa que a saída de V após a interação com P sobre uma entrada x (da afirmação $x \in L$), denotada por $\langle P,V\rangle(x)$, pode ser simulada por um computador M que *não contacta* P. Dessa forma fica explícito que V não aprende nada de P além da validade da afirmação.

De maneira um pouco diferente, um sistema de *prova de conhecimento* permite a P provar que *conhece* um y tal que $(x,y) \in R_L$. Ou seja, enquanto em um sistema de prova, P convence V de que *existe* uma testemunha y para x, em uma prova de conhecimento, P convence V de que *conhece* uma tal testemunha y.

Além de V e P, um sistema de prova de conhecimento define um terceiro computador K, chamado de *extrator*. É através do extrator que se "atesta" o conhecimento de P. O objetivo do extrator é, a partir da interação com um oráculo para P, construir a testemunha y que P alega conhecer.

Em um sistema de prova de conhecimento, o tempo esperado de execução do extrator é inversamente proporcional à probabilidade do verificador V aceitar uma prova de P. Intuitivamente isso pode ser interpretado como "quanto mais fácil for de extrair o conhecimento de P, mais fácil é de se convencer V".

Uma definição mais *forte* para um sistema de prova de conhecimento exige que K execute sempre em tempo polinomial, embora possa falhar, com probabilidade desprezível, em extrair de P uma testemunha válida. Com essa definição, as provas são chamadas de *provas fortes* e o sistema como um todo, de *sistema de provas fortes de conhecimento*. A mesma noção de conhecimento zero se aplica a esses sistemas.

A teoria de sistemas de prova e de prova de conhecimento é vasta e de grande interesse para a criptografia de maneira geral. Não pretendemos cobrí-la totalmente nesta seção. Procuraremos apenas definir o necessário para o entendimento do nosso Capítulo 4. Isso significa que nos concentraremos em definir o conceito de *sistema de provas fortes de conhecimento de conhecimento zero*.

3.4.2 Definições Preliminares

Algumas definições preliminares são necessárias para o prosseguimento da nossa exposição. Por exemplo, mencionamos os "computadores" V, P e K. Essa noção de computador pode ser capturada por uma *Máquina de Turing Interativa* (MTI). Fazemos essa definição a seguir.

Definição 3.9 (Máquina de Turing interativa - Definição 4.1.1 de GOLDREICH (2001))**.** *Uma máquina de Turing interativa é uma máquina de Turing que possui fitas de entrada (somente leitura), de entrada auxiliar (somente leitura), de fonte de aleatoriedade (somente leitura), de trabalho (leitura e escrita), de saída (somente escrita), de comunicação (uma somente leitura e outra somente escrita) e de estado (leitura e escrita, consistindo de apenas uma célula).*

Cada MTI possui um bit $\sigma \in \{0,1\}$ chamado de sua identidade. Uma MTI está ativa se o conteúdo de sua fita de estado é igual à sua identidade. Caso contrário ela está inativa. Enquanto uma MTI está inativa, toda a sua configuração, i.e., a posição e conteúdo de suas fitas, é preservada.

Quando uma MTI escreve na sua fita de comunicação (somente escrita) dizemos que ela envia uma mensagem. A mensagem é, naturalmente, o conteúdo escrito na fita. De maneira semelhante, ao ler o conteúdo da sua fita de comunicação (somente leitura), dizemos que ela recebeu uma mensagem.

A definição de MTI se completa com a de *computação conjunta*, feita a seguir.

Definição 3.10 (Computação conjunta de duas MTIs - Definição 4.2.1 de GOLDREICH (2001))**.** *Dizemos que duas MTIs estão conectadas se (i) elas têm identidades complementares, (ii) suas fitas de entrada e de estado coincidem e (iii) a fita de comunicação (somente leitura) de uma coincide com a fita de comunicação (somente escrita) da outra e vice-versa.*

*Uma **computação conjunta** entre duas MTIs conectadas é uma sequência de pares representando a configuração de cada uma delas. Cada par pode ser visto como uma "rodada" em que apenas uma das máquinas está ativa. A rodada de uma máquina acaba quando ela escreve o complemento de sua identidade na fita (compartilhada) de estado. É aí que a outra máquina se torna ativa. Se, no entanto, uma máquina pára enquanto o conteúdo da fita de estado ainda é a sua identidade, dizemos que ambas as máquinas conectadas param e, portanto, a computação se encerra. É nesse momento que as saídas de ambas são determinadas.*

Denotamos por $\langle A(y), B(z)\rangle(x)$ a variável aleatória representando a saída de B após a computação conjunta com A sobre a entrada compartilhada x e as entradas auxiliares y (de A) e z (de B).

O objetivo das definições de MTI e de computação conjunta é definir de maneira mais precisa o que significa um provador P e um verificador V interagirem. De maneira geral, uma prova (de conhecimento) é uma computação conjunta entre P e V. Nessa computação, P quer provar que conhece uma testemunha para a entrada compartilhada x. Admitimos que a entrada auxiliar de P é tal testemunha ou, pelo menos, ajuda P durante a prova. Ao final da interação, após ambos trocarem mensagens, a saída de V (0 ou 1) indica se a prova de conhecimento foi aceita ou não. Ou seja, se P conseguiu convencer V. Vamos voltar a esse assunto mais adiante.

A última definição de que precisamos para prosseguirmos na nossa apresentação é a de *função de especificação de mensagens de uma MTI*. Em linhas gerais, essa função determina qual a mensagem que uma MTI P envia em reposta a uma determinada mensagem recebida.

Definição 3.11 (Função de especificação de mensagens - Definição 4.7.1 de GOLDREICH (2001))**.** *Seja $P_{x,y,r}$ uma função tal que $P_{x,y,r}(\bar{m})$ é a mensagem enviada por uma MTI P executando sobre entrada x, entrada auxiliar y e fonte de aleatoriedade r, após receber a mensagem $\bar{m}$. Dizemos que $P_{x,y,r}$ é a **função de especificação de mensagens** de P com entrada x, entrada auxiliar y e fonte de aleatoriedade r.*

3.4.3 Sistema de provas fortes de conhecimento

O problema que um sistema de provas (fortes) de conhecimento resolve é o seguinte. Um provador P quer convencer um verificador V de que possui uma testemunha s para x em uma relação R. Ou seja, que $(x,s) \in R$. Uma *prova* é uma computação conjunta entre P e V. O valor x é a entrada compartilhada por ambos P e V, ou seja, é o valor da fita de entrada compartilhada por eles. O provador P pode ter uma entrada auxiliar que o ajude na prova que, de forma geral, é o próprio s. Da mesma maneira, V também pode ter uma entrada auxiliar.

A ideia de que P conhece um valor s é capturada por uma MTI K chamada de *extrator*. Fixando-se a entrada x, fonte de aleatoriedade r e entrada auxiliar y, K tem acesso oracular à função $P_{x,y,r}$ de especificação de mensagens de P. O objetivo de K é usar $P_{x,y,r}$ como caixa-preta para, em tempo polinomial, obter s (tal que $(x,s) \in R$). Intuitivamente, K consegue obter s porque P, de fato, conhece esse valor.

A seguir vamos definir um sistema de provas fortes de conhecimento. Nessa definição, em particular, faremos restrições sobre o funcionamento de K.

Definição 3.12 (Sistema de provas fortes de conhecimento - Definição 4.7.13 de GOLDREICH (2001))**.** *Seja R uma relação binária. Dizemos que uma MTI V é um* ***verificador de conhecimento forte para*** *R se as seguintes condições se verificam:*

(i) (Não-trivialidade:) Existe uma MTI P tal que, para todo $(x,s) \in R$, todas as possíveis interações entre V e P em que x é a entrada compartilhada e s é a entrada auxiliar de P terminam em aceitação de V (i.e., a saída de V é 1).

(ii) (Validade forte:) Existe uma função desprezível $\mu : \mathbb{N} \to [0,1]$ e uma MTI probabilística de tempo polinomial K tal que, para toda MTI P e todo $x,y,r \in \{0,1\}^*$, a máquina K, executando sobre a entrada x e com acesso oracular a $P_{x,y,r}$, possui a seguinte propriedade:

> *Seja $p(x,y,r)$ a probabilidade de V aceitar a prova de P quando x é a entrada compartilhada e y é a entrada auxiliar de P. Se $p(x,y,r) > \mu(|x|)$, então a saída s de K é tal que $Pr[(x,s) \in R] \geq 1 - \mu(|x|)$.*[4]

A MTI K é chamada de ***extrator forte****.*

Se V é um verificador de conhecimento forte para uma relação R, e P é uma MTI que satisfaz a condição (i) de não-trivialidade (com respeito a R e V), então o par (P,V) de MTIs conectadas é chamado de ***sistema de provas fortes de conhecimento****.*

A propriedade de não-trivialidade da Definição 3.12 estabelece que o provador sempre consegue convencer o verificador se ele de fato estiver falando a verdade. Ou seja, ele sempre consegue provar que $(x,y) \in R$ quando de fato conhece y.

A propriedade de validade forte, por sua vez, define uma função desprezível μ como limite superior para a probabilidade de falha da prova. Note que, nesta propriedade, os valores x,y são tomados de $\{0,1\}^*$ e, portanto, não é necessariamente verdade que $(x,y) \in R$. Se $(x,y) \in R$, então, pela propriedade (i), temos que $p(x,y,r) = 1$. Nesse caso, K deveria sempre conseguir "extrair o conhecimento" de P. A probabilidade disso não acontecer é, no máximo, $\mu(|x|)$.

Se, por outro lado, não existe testemunha válida para x, isto é, não existe s' tal que $(x,s') \in R$, então P está tentando provar uma afirmação falsa. O extrator não pode encontrar tal testemunha e, portanto, sua saída s é tal que $Pr[(x,s) \in R] = 0$. Analisando a propriedade de validade forte na contrapositiva, temos que $Pr[(x,s) \in R] < 1 - \mu(|x|)$ implica em $p(x,y,r) \leq \mu(|x|)$. Portanto, combinando essa implicação com o fato de que $Pr[(x,s) \in R] = 0$, conclui-se que as chances de V aceitar uma prova inválida são limitadas superiormente por μ e, consequentemente, desprezíveis. Isso significa que a propriedade de validade forte garante a *corretude* (*soundness*) do sistema.

Por fim, ressaltamos que, segundo a nomenclatura definida em GOLDREICH (2001), não existe um sistema de provas "fracas" de conhecimento. Como mencionamos de maneira

[4] Ressaltamos que a saída de K é dada com base na interação com $P_{x,y,r}$. Ele não interage nem com V nem com P.

introdutória, o adjetivo "forte" se refere à restrição de que o verificador K executa sempre em tempo polinomial, embora com uma probabilidade de falha. Um sistema que não é forte é definido como na Definição 3.12, mas o tempo de execução do extrator varia com o inverso da probabilidade $p(x,y,r)$[5].

3.4.4 Conhecimento Zero

Vimos que um sistema de provas fortes de conhecimento é, pela sua própria definição, correto e não-trivial. No entanto, desejamos que ele tenha também a propriedade de *conhecimento zero*. Isto é, o verificador não deve aprender nada sobre a testemunha que o provador possui além do que suas entradas compartilhada e auxiliar revelam. Discutiremos extensamente o que significa *não aprender nada de uma execução* no Capítulo 4. Em linhas gerais, utilizamos o *paradigma de simulação* e dizemos que um sistema de prova de conhecimento tem conhecimento zero se a interação entre o provador e o verificador pode ser simulada por um algoritmo eficiente, chamado *simulador*.

Ao simulador é dado acesso somente à entrada compartilhada x e à entrada auxiliar z do verificador. Sua tarefa é, a partir apenas dessas informações, chegar a uma resposta "compatível" com a que V chegaria na verdadeira interação com P. Se isso é possível, então conclui-se que toda informação "vazada" na interação entre V e P pode ser computada sem que essa interação ocorra. Intuitivamente, isso é o máximo de segurança que uma interação pode ter.

Registramos como o conhecimento zero se aplica a sistemas de provas fortes de conhecimento na definição abaixo.

Definição 3.13 (Sistema de provas fortes de conhecimento de conhecimento zero (com entradas auxiliares) - Definição 4.3.10 de GOLDREICH (2001))**.** *Seja (P,V) um sistema de provas de conhecimento para uma relação R e $L \stackrel{def}{=} \{x \mid \exists y \text{ t.q } (x,y) \in R\}$ a linguagem definida por R. Para todo $x \in L$, denotamos por y_x a testemunha de x em R, ou seja, $(x,y_x) \in R$. Dizemos que (P,V) é um* ***sistema de provas fortes de conhecimento de conhecimento zero*** *se, para toda MTI V^* probabilística de tempo polinomial, existe um algoritmo probabilístico M^* de tempo polinomial (no tamanho da sua primeira entrada) tal que, para todo $x \in L$ e $z \in \{0,1\}^*$:*

$$\{\langle P(y_x), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*} \stackrel{c}{\equiv} \{M^*(x,z)\}_{x\in L, z\in\{0,1\}^*}$$

A definição acima formaliza o que discutimos anteriormente. A MTI M^*, sem acesso a P, gera saídas indistinguíveis das geradas por V após uma interação com P. Isso significa que V não obtém qualquer conhecimento ao interagir com P (além do que pode ser obtido da entrada x e da entrada auxiliar z). Por isso, um sistema de prova de conhecimento zero é aquele que preserva, durante a prova, o sigilo da testemunha conhecida por P .

Um detalhe importante a ser ressaltado é que o conhecimento zero se verifica mesmo que P seja "atacado" por uma máquina arbitrária V^*. Ou seja, P preserva o segredo da testemunha

[5] Ver Definição 4.7.2 de GOLDREICH (2001).

que possui mesmo que o verificador assuma uma estratégia (maliciosa) arbitrária durante a prova (interação).

Destacamos ainda que a definição de conhecimento zero é geral e não se aplica apenas a sistema de provas fortes de conhecimento. Ele também se aplica a sistemas que não são fortes e também a sistemas apenas de prova (sem ser de conhecimento).

Todas as relações da classe *NP* admitem sistemas de provas fortes de conhecimento zero. De maneira geral, tais sistemas podem ser obtidos a partir de sistemas já conhecidos para algumas linguagens *NP*-completas específicas, como a dos grafos 3-coloríveis e a dos grafos Hamiltonianos.

3.5 Esquemas de Comprometimento

Um esquema de comprometimento permite a um *emissor* se comprometer com um determinado valor sem torná-lo público. O esquema dá a possibilidade do emissor revelar, posteriormente, o tal valor secreto de modo que o *receptor* possa ter certeza de que o valor revelado é, de fato, aquele com o qual o emissor se comprometeu. Uma possível analogia é a de que o emissor tranca uma mensagem em um cofre e envia esse cofre ao receptor. Posteriormente o emissor pode revelar o segredo de abertura do cofre. Desse modo o receptor pode ter certeza de que a mensagem não foi alterada.

Duas propriedades são importantes para um esquema de comprometimento. A primeira delas, a do *sigilo*, estabelece que o receptor não pode aprender nada sobre o valor com o qual o emissor se comprometeu até que ele seja revelado. Ou seja, na analogia, o emissor não consegue inferir algo sobre a mensagem dentro do cofre (muito menos abrir o cofre).

A segunda propriedade, a da *não-ambiguidade*, estabelece que o emissor não pode revelar um valor diferente do qual ele se comprometeu sem que o receptor perceba. Na analogia, isso significa que, uma vez dentro do cofre, a mensagem não pode ser alterada pelo emissor.

Registramos, a seguir, a definição de um *esquema de comprometimento para um bit*. Utilizamos as definições 3.9 (*Máquina de Turing interativa*) e 3.10 (*Computação conjunta de duas MTIs*) como auxiliares.

Definição 3.14 (Esquema de comprometimento para um *bit* - Definição 4.4.1 de GOLDREICH (2001))**.** *Um* ***esquema de comprometimento para um bit*** *C é um par (S,R) de MTIs de tempo polinomial (S de sender e R de receiver) tais que:*

1. *(Entradas:) A entrada compartilhada por S e R é um parâmetro de segurança n (representado em unário). A entrada auxiliar de S é um bit v com o qual ele quer se comprometer.*

2. *(Sigilo:) O receptor R, mesmo adotando uma estratégia arbitrária, não consegue distinguir quando S se compromete com $v = 0$ de quando ele se compromete com $v = 1$.*

Ou seja, para toda MTI R^* de tempo polinomial interagindo com S:

$$\{\langle S(0), R^*\rangle(1^n)\}_{n\in\mathbb{N}} \stackrel{c}{\equiv} \{\langle S(1), R^*\rangle(1^n)\}_{n\in\mathbb{N}}$$

3. *(Não-ambiguidade:) Definições auxiliares:*

   - *A* ***visão*** $(r,\bar{m})$ *que o receptor tem da interação com o emissor consiste da fonte de aleatoriedade r utilizada pelo receptor e da sequência $\bar{m}$ das mensagens recebidas do emissor.*
   - *Seja $\sigma \in \{0,1\}$ um bit. Uma visão $(r,\bar{m})$ é um* ***comprometimento com*** σ *se existe uma cadeia $s \in \{0,1\}^*$ tal que $\bar{m}$ é a sequência das mensagens recebidas por R na interação com S quando S usa s como fonte de aleatoriedade e tem σ como entrada auxiliar .*
   - *A visão $(r,\bar{m})$ é* ***ambígua*** *se ela é tanto um comprometimento com 0 quanto um comprometimento com 1.*

   Um esquema é não-ambíguo se apenas uma quantidade desprezível de fontes de aleatoriedade $r \in \{0,1\}^n$ tornam ambígua a visão $(r,\bar{m})$ do receptor para alguma sequência $\bar{m}$ de mensagens recebidas.

A propriedade de sigilo estabelece que a interação com S quando ele executa sobre entrada auxiliar 0 é indistinguível daquela em que ele executa sobre entrada auxiliar 1. Ou seja, nenhuma estratégia arbitrária R^* consegue determinar, com probabilidade não desprezível, se S se comprometeu com 0 ou com 1. A estratégia é *arbitrária* para ressaltar que o sigilo da entrada de S não é garantido apenas na interação com a estratégia honesta de R, mas na interação com *qualquer* estratégia (eficiente).

As propriedades de sigilo e não-ambiguidade ressaltam a importância da fonte de aleatoriedade utilizada por S. De fato, se S não usasse uma fonte de aleatoriedade, ele se comprometeria com o *bit* 1 (ou com 0) sempre da mesma maneira. Dessa forma, o sigilo de tal *bit* seria trivialmente quebrado.

Por outro lado, o receptor R poderia temer que S se comprometesse com um *bit* σ e depois conseguisse achar uma fonte de aleatoriedade s que fizesse parecer que ele se comprometera com $1-\sigma$. A propriedade de não-ambiguidade garante que isso só é possível com uma probabilidade desprezível (no parâmetro de segurança).

Uma característica que vale ser notada na Definição 3.14 é que o comprometimento de um emissor é a *própria interação* com o receptor, no mesmo espírito de um sistema de prova de conhecimento. No entanto, é conveniente adotar uma definição mais "concreta", que aproxime a ideia de um esquema de comprometimento da sua implementação. Fazemos isso na definição a seguir.

Definição 3.15 (Esquema de comprometimento para um *bit* (revisitada))**.** *Seja $\sigma \in \{0,1\}$ um bit, 1^n um parâmetro de segurança e $s \in \{0,1\}^n$ uma cadeia. Um* ***esquema de comprometimento para um bit*** *é um algoritmo determinístico C de tempo polinomial tal que $C_s(\sigma)$ é o resultado de se comprometer com σ utilizando s como fonte de aleatoriedade.*

Existem duas abordagens para que o receptor verifique se $C_s(\sigma)$ foi de fato gerado sobre o par (s,σ) eventualmente revelado pelo emissor. Pode-se admitir a existência de um algoritmo *Verifica* tal que $\mathit{Verifica}(s,\sigma,C_s(\sigma)) = 1$ se e somente se $C_s(\sigma)$ foi gerado com o par (s,σ). Outra possível abordagem, adotada neste trabalho, é a de que o algoritmo C é conhecido pelo receptor. Portanto, ele pode executar C sobre as entradas (s,σ) e verificar se o resultado coincide com $C_s(\sigma)$.

Fazendo uma relação com a Definição 3.14, C é a estratégia do emissor S. Já $C_s(\sigma)$ é a visão que R tem da interação com S quando S usa s como fonte de aleatoriedade e σ como entrada auxiliar. A propriedade de sigilo significa que $C_s(\sigma)$ não revela ao receptor informação alguma sobre s ou σ. Já a não-ambiguidade significa que não existe $s' \in \{0,1\}^n$ tal que $C_{s'}(1-\sigma) = C_s(\sigma)$ (na verdade s' existe apenas com probabilidade desprezível).

Algo importante a ser ressaltado da Definição 3.15 é o fato de o algoritmo C ser determinístico. De fato, C é determinístico porque recebe *explicitamente* uma fonte de aleatoriedade (s). Essa característica é importante, por exemplo, para possibilitar que R verifique a validade do comprometimento a partir de s e σ.

De maneira geral, pode ser necessário ao emissor S se comprometer com uma *cadeia* de *bits* ao invés de com um único *bit*. Naturalmente isso pode ser alcançado através do comprometimento com cada *bit* da cadeia individualmente. Definimos a seguir a notação que utilizaremos nesse caso.

Definição 3.16 (Esquema de comprometimento para cadeias de *bits* (notação))**.** *Seja C um esquema de comprometimento para um bit como na Definição 3.15, 1^n um parâmetro de segurança e $x \in \{0,1\}^*$ uma cadeia. Para todo $i \in \{1,\cdots,|x|\}$, seja $s_i \in \{0,1\}^n$ uma cadeia. Definimos a cadeia $s \in \{0,1\}^{n\cdot|x|} \stackrel{def}{=} s_1||\cdots||s_{|x|}$ como a concatenação de todas as cadeias s_i. Definimos ainda x_i como o i-ésimo bit de x.*

Definimos $\overline{C}_s(x) \stackrel{def}{=} (C_{s_1}(x_1),\cdots,C_{s_{|x|}}(x_{|x|}))$ como o ***resultado de se comprometer com a cadeia*** *x* ***usando*** *s* ***como fonte de aleatoriedade****. Note que $\overline{C}_r(x)$ é a sequência dos resultados de se comprometer com cada bit de x.*

3.5.1 Esquemas de Comprometimento Equívocos

No decorrer deste trabalho estaremos também interessados em sistemas de comprometimento *equívocos*. Tais sistemas permitem que o emissor quebre a propriedade de não-ambiguidade sem que o receptor perceba. Isso é feito através de um par de algoritmos: um para gerar um comprometimento "coringa" e outro para gerar uma fonte de aleatoriedade compatível com tal "coringa".

Definimos esse tipo de comprometimento a seguir.

Definição 3.17 (Esquema de comprometimento equívoco - Definição 4 de BENTOV e KUMARESAN (2014))**.** *Seja C um esquema de comprometimento para um bit como na Definição 3.15. Dizemos que ele é* ***equívoco*** *se possui dois algoritmos determinísticos de tempo polinomial Equivoca e Desequivoca tais que:*

(i) ('Equivoca' gera comprometimento coringa:) Seja $z \in \{0,1\}^n$ uma cadeia. Então $Equivoca(z) = (coringa, info)$, com $info \in \{0,1\}^{poly(n)}$. O valor coringa é um comprometimento ambíguo, segundo a Definição 3.14. Ou seja, existem $s, s' \in \{0,1\}^n$ tais que coringa é um comprometimento com 0 ($C_s(0)$) e com 1 ($C_{s'}(1)$).

(ii) ('Desequivoca' gera fonte de aleatoriedade apropriada:) Seja $\sigma \in \{0,1\}$ um bit. Então $Desequivoca(info, \sigma) = s$, com $s \in \{0,1\}^n$.

(iii) (O receptor não consegue distinguir:) Para todo algoritmo probabilístico de tempo polinomial R^* e todo $\sigma \in \{0,1\}$ temos que:

$$\{R^*(1^n, \sigma, Desequivoca(info_z, \sigma), coringa_z)\}_{n\in\mathbb{N}, z\in\{0,1\}^n} \stackrel{c}{\equiv} \{R^*(1^n, \sigma, s, C_s(\sigma))\}_{n\in\mathbb{N}, s\in\{0,1\}^n}$$

em que $Equivoca(z) = (coringa_z, info_z)$.

O algoritmo *Equivoca* da Definição 3.17 gera um comprometimento ambíguo com base apenas em uma fonte de aleatoriedade z. Além disso, ele gera uma cadeia *info* que permite que *Desequivoca* encontre, de forma eficiente, uma fonte de aleatoriedade correspondente para desambiguar o comprometimento.

A propriedade (iii) estabelece que não há um algoritmo eficiente para se distinguir o resultado de se usar C "honestamente" e de se usar os algoritmos de equivocação. Em particular, o valor *coringa* é indistinguível de um $C_s(\sigma)$ honestamente gerado.

Note que um esquema equívoco ainda pode ser utilizado de forma "honesta", isto é, com a propriedade de não-ambiguidade preservada. É assim que utilizaremos esse tipo de esquema. Mais especificamente, utilizaremos um sistema equívoco em um contexto em que ele será *garantidamente* utilizado de forma honesta. A utilidade dele será a de permitir a prova de correção de algumas construções que apresentaremos.

3.6 Esquemas de Compartilhamento de Segredo

Um *esquema de compartilhamento de segredo* permite que um segredo z seja dividido em m pedaços. Isso é feito de modo que z só possa ser reconstruído a partir de pelo menos t desses pedaços. Naturalmente, qualquer subconjunto com menos de t pedaços não deve revelar nenhuma informação sobre z. Registramos essa definição a seguir.

Definição 3.18 (Esquemas de compartilhamento de segredo)**.** *Sejam t, m dois inteiros positivos com $t \leq m$. Um (t,m)-esquema de compartilhamento de segredo é um par de algoritmos (D,R) tais que:*

1. *D é um algoritmo probabilístico que mapeia um bit em uma sequência de m pedaços. Isto é, para todo $\sigma \in \{0,1\}$, a variável aleatória $D(\sigma)$ assume valores do conjunto $(\{0,1\}^*)^m$.*

2. *R é um algoritmo determinístico que recebe uma sequência de t pedaços e retorna o segredo (bit) que foi dividido. Em símbolos, $R(\bar{p}) = \sigma$, com $\bar{p} \in (\{0,1\}^*)^t$ e $\sigma \in \{0,1\}$.*

3. *(Condição de reconstrução:) Para todo bit $\sigma \in \{0,1\}$, toda sequência $(p_1, \cdots, p_m)$ do contradomínio de D e todo subconjunto $\{i_1, \cdots, i_t\} \subseteq [m]$, com $[m] \stackrel{def}{=} \{1, \cdots, m\}$, temos que:*

$$R(p_{i_1}, \cdots, p_{i_t}) = \sigma$$

4. *(Condição de sigilo:) Nenhum subconjunto com menos de t pedaços deve revelar informações sobre σ. Formalmente, seja $g_I(\sigma) \stackrel{def}{=} (p_{i_1} \cdots, p_{i_{t-1}})$ com $I \stackrel{def}{=} \{i_1, \cdots, i_{t-1}\} \subset [m]$ e $(p_1, \cdots, p_m) = D(\sigma)$. Então, para todo subconjunto I dessa forma (i.e., com $t-1$ elementos), $g_I(0)$ tem distribuição idêntica à de $g_I(1)$.*

Suponha que duas pessoas, Alice e Bob, vão participar do seguinte jogo. Um *bit* σ secreto foi dividido em dois pedaços p_A e p_B e só pode ser reconstruído a partir dos dois[6]. Alice só conhece p_A e Bob só conhece p_B. O jogo começa com Alice e Bob apostando sobre qual é o valor de σ. Em seguida eles revelam p_A e p_B, reconstroem σ e determinam quem venceu.

No entanto, se Bob quiser trapacear, ele pode esperar Alice divulgar p_A para reconstruir σ primeiro e só então decidir como agir: revelar p_B se sua aposta foi vencedora ou divulgar outro valor p'_B, caso contrário. Isso é possível porque não há nada que *comprometa* Bob com p_B. Isto é, Alice não tem como verificar se o que Bob revelar é de fato o pedaço p_B.

Usamos essa situação para motivar a extensão de um esquema de compartilhamento de segredo na qual os pedaços vêm acompanhados de um *comprometimento* calculado sobre eles. Vamos chamar esse comprometimento de *etiqueta*. No exemplo acima, no início do jogo, Bob e Alice publicariam suas etiquetas e_B e e_A. Com isso, Alice poderia verificar se o valor revelado por Bob é de fato aquele correspondente a e_B. A impossibilidade de Bob revelar um valor diferente de p_B no decorrer do jogo se baseia na não-ambiguidade da etiqueta e_B.

Chamamos essa extensão de *esquema verificável de compartilhamento de segredo* e a definimos a seguir.

Definição 3.19 (Esquemas verificáveis de compartilhamento de segredo)**.** *Seja C um esquema de comprometimento, m,t dois inteiros positivos, com $t \leq m$ e $\Gamma_{m,t} = (D,R)$ um esquema de compartilhamento de segredo como na Definição 3.18. Um (t,m)-esquema verificável de compartilhamento de segredo é um trio de algoritmos $(Divide, Reconstrói, Verifica)$ tais que:*

[6] O *bit* σ foi dividido com um $(2,2)$-esquema de compartilhamento de segredo.

1. *Divide é um algoritmo probabilístico que mapeia um bit* σ *em uma sequência* $((e_1,(p_1,w_1)),\cdots,(e_m,(p_m,w_m)))$ *em que* $(p_1,\cdots,p_m) = D(\sigma)$ *e* $C_{w_i}(p_i) = e_i$ *para todo* $i \in \{1,\cdots,m\}$.

2. *Verifica é um algoritmo determinístico que recebe* $(e,(p,w))$ *e retorna 1 se* $e = C_w(p)$. *Ou seja:*

$$Verifica(e,(p,w)) \stackrel{def}{=} \begin{cases} 1 & \text{se } C_w(p) = e, \\ 0 & \text{caso contrário.} \end{cases}$$

3. *Reconstrói é um algoritmo determinístico que recebe uma sequência de* t *pedaços e retorna o segredo (bit) que foi dividido. Em símbolos,* $Reconstrói(\bar{p}) = \sigma$, *com* $\bar{p} \in (\{0,1\}^*)^t$ *e* $\sigma \in \{0,1\}$.

4. *(Condição de reconstrução:) Para todo bit* $\sigma \in \{0,1\}$, *toda sequência* $((e_1,(p_1,w_1)),\cdots,(e_m,(p_m,w_m)))$ *do contradomínio de Divide e todo subconjunto* $I \stackrel{def}{=} \{i_1,\cdots,i_t\} \subseteq [m]$, *com* $[m] \stackrel{def}{=} \{1,\cdots,m\}$, *temos que:*

$$Reconstrói((e_{i_1},(p_{i_1},w_{i_1})),\cdots,(e_{i_t},(p_{i_t},w_{i_t}))) \stackrel{def}{=} \begin{cases} \sigma & \text{se } C_{w_i}(p_i) = e_i \text{ para todo } i \in I \\ \perp & \text{caso contrário.} \end{cases}$$

5. *(Condição de sigilo:) Nenhum subconjunto com menos de* t *pedaços deve revelar informações sobre* σ. *Formalmente, seja* $g_I(\sigma) \stackrel{def}{=} (p_{i_1}\cdots,p_{i_{t-1}})$ *com* $I \stackrel{def}{=} \{i_1,\cdots,i_{t-1}\} \subset [m]$ *e* $(p_1,\cdots,p_m) = D(\sigma)$. *Então, para todo subconjunto* I *dessa forma (i.e., com* $t-1$ *elementos),* $g_I(0)$ *tem distribuição idêntica à de* $g_I(1)$.

3.7 Funções *Hash*

De forma geral, uma função *hash* é uma função que comprime uma cadeia de tamanho arbitrário em uma de tamanho menor. Uma das motivações para o uso de uma função *hash* é sua aplicação na área de estruturas de dados, permitindo o acesso eficiente a valores previamente armazenados. Para isso, cada elemento é armazenado em uma tabela e indexado a partir de seu valor *hash*. Dessa maneira, elementos com *mesmo valor hash* acabam sendo armazenados em uma mesma célula da tabela.

Dois elementos que possuem o mesmo valor *hash* representam uma *colisão*. Como o domínio de uma função *hash* é, em geral, bem maior que seu contradomínio, podendo inclusive ser infinito, a existência de colisões é inevitável. Uma "boa" função *hash*, no entanto, é aquela em que essas colisões são "difíceis" de serem encontradas. Em outras palavras, uma boa função de *hash* é *resistente a colisões*.

Enquanto a resistência à colisão é um requisito desejável para que uma função *hash* seja utilizada em estruturas de dados, ela se torna essencial para o uso dessas funções em aplicações de criptografia. Nessa área, um exemplo importante de aplicação de funções *hash* resistentes a

colisões é a geração de códigos de autenticação. Esses códigos conferem integridade a cadeias arbitrárias e podem ser utilizados, por exemplo, para a criação de esquemas de criptografia com níveis mais altos de segurança.

Nosso interesse em funções *hash*, naturalmente, implica que tais funções sejam resistentes a colisões. Finalizamos essa breve discussão informal com as definições de função *hash* e resistência a colisões.

Definição 3.20 (Função *hash* - Definição 5.1 de KATZ e LINDELL (2014))**.** *Uma* ***função hash*** *é um par* (Gen, H) *de algoritmos probabilísticos de tempo polinomial tais que:*

(i) *Gen é um algoritmo probabilístico que recebe um parâmetro de segurança* 1^n *como entrada e tem como resultado uma fonte de aleatoriedade (ou chave)* $s \in \{0,1\}^{poly(n)}$.

(ii) *H é um algoritmo determinístico de tempo polinomial que recebe s e* $x \in \{0,1\}^*$ *como entradas e tem como resultado* $H_s(x) \in \{0,1\}^{poly(n)}$.

Utilizamos $H(x)$ *significando* $H_s(x)$ *para alguma chave s.*

A Definição 3.20 formaliza a discussão introdutória que fizemos. Vale ser ressaltado, no entanto, a existência do algoritmo *Gen* e o determinismo de H. Em geral, é importante que a função H tenha um comportamento probabilístico para que a saída de H não seja totalmente definida por sua entrada x. Isso é alcançado tornando explícita a fonte de aleatoriedade usada por H. As funções *hash* usadas na prática, entretanto, consistem apenas do algoritmo determinístico H. Dessa forma, cabe à aplicação de mais alto nível prover aleatoriedade para H.

Uma colisão para uma função *hash* (Gen, H) é um par (x, x') de cadeias tais que $x \neq x'$ e $H_s(x) = H_s(x')$ para algum s. Uma função *hash* é resistente a colisões se é "difícil" encontrar um desses pares. A noção de dificuldade é, naturalmente, capturada pela (não) existência de um algoritmo polinomial para encontrar uma colisão. Formalizamos a noção de resistência a colisões com base no experimento definido a seguir.

Definição 3.21 (Experimento de busca por colisões - KATZ e LINDELL (2014))**.** *Seja* $\Gamma = (Gen, H)$ *uma função hash, A um adversário e* $n \in \mathbb{N}$ *um parâmetro de segurança. Definimos o* ***experimento de busca por colisões****, denotado por* $Col_{A,\Gamma(n)}$*, como a seguir:*

1. *Uma chave* $s = Gen(1^n)$ *é gerada.*
2. *O adversário A recebe s como entrada. Ele deve gerar um par de cadeias* (x, x').
3. *O resultado do experimento é 1 se e somente se* $x \neq x$ *e* $H_s(x) = H_s(x')$*. Dizemos, neste caso, que A encontrou uma colisão.*

Com a Definição 3.21 podemos formalizar a noção de dificuldade em se encontrar uma colisão para uma determinada função *hash*.

Definição 3.22 (Resistência a colisões - Definição 5.2 de KATZ e LINDELL (2014))**.** *Uma função hash* (Gen, H) *é* ***resistente a colisões*** *se, para todo* $n \in \mathbb{N}$ *e todo algoritmo probabilístico A de tempo polinomial existe uma função* μ *desprezível (em n) tal que:*

$$Pr[Col_{A,\Gamma(n)} = 1] \leq \mu(n)$$

A Definição 3.22 estabelece que uma função *hash* resistente a colisões não admite um algoritmo eficiente A para encontrar colisões, i.e., pares (x, x'), com probabilidade não desprezível.

Um famoso ataque para se achar colisões em uma função *hash* é o *ataque do aniversário* (*birthday attack*). Ele se baseia no chamado *problema do aniversário*[7] (*birthday problem*): qual a probabilidade de quaisquer 2 pessoas, entre um total de q, fazerem aniversário no mesmo dia do ano? Assumindo que os aniversários são uniformemente distribuídos ao longo dos 365 dias de um ano (não bissexto), quando $q = 23$ essa probabilidade é superior a 50%. Note que o problema do aniversário lida, basicamente, com as chances de ocorrência de uma *colisão de datas de aniversário*.

O ataque do aniversário consiste em procurar por uma colisão entre um conjunto com $O\left(\sqrt{2^{l(n)}}\right)$ valores do contradomínio de H de uma função *hash* (Gen, H)[8]. Considerando que H é uma função aleatória[9], quando $O\left(\sqrt{2^{l(n)}}\right)$ valores são analisados, a chance de se encontrar uma colisão entre quaisquer dois deles é maior que 50%. Uma análise mais aprofundada é feita por KATZ e LINDELL (2014).

Do ponto de vista da complexidade assintótica, não existe diferença entre se conduzir um ataque de força bruta, analisando-se $2^{l(n)}$ valores, e o ataque do aniversário, onde são considerados "apenas" $O\left(\sqrt{2^{l(n)}}\right) = O\left(2^{l(n)/2}\right)$ valores. Do ponto de vista prático, no entanto, existe uma grande diferença. Considere, por exemplo, o caso em que $l(n) = 128$. Um ataque de força bruta é infactível, pois 2^{128} é uma quantidade demasiadamente grande de elementos para analisar. No entanto, o ataque do aniversário precisa analisar em torno de 2^{64} possibilidades, o que já não pode ser considerado infactível.

As funções *hash* utilizadas na prática, em geral, têm a chamada segurança *heurística*. Em particular, algumas não utilizam uma fonte de aleatoriedade (ou chave) como na Definição 3.20 (e.g., a função *SHA256*). Como consequência, uma colisão (x, x') pode inutilizar completamente a função. Em uma função *hash* com uso de chave, uma colisão (x, x') serve, a princípio, apenas para uma determinada chave.

Além disso, as funções *hash* usadas na prática normalmente têm um contradomínio de tamanho fixo. Isso significa que não existe a noção de parâmetro de segurança e, portanto, elas

[7] Também conhecido como *paradoxo* do aniversário. É chamado de "paradoxo" apenas porque a análise matemática resulta em uma resposta "não intuitiva" para o problema.

[8] A quantidade total de elementos no contradomínio de H é $2^{l(n)}$.

[9] Como destaca KATZ e LINDELL (2014), o caso da função ser aleatória é o pior caso para o atacante. Quanto mais a função se desviar de uma "aleatoriedade perfeita", mais fácil é encontrar uma colisão.

não podem seguir rigorosamente a Definição 3.22.

3.8 Esquemas de Criptografia Assimétricos

Um esquema de criptografia, de maneira geral, permite que uma mensagem m seja "embaralhada" por um emissor de modo que apenas o receptor dessa mensagem consiga "desembaralhá-la" corretamente. O processo de embaralhamento da mensagem é chamado *encriptação* e o de desembaralhamento, *decriptação*. Chamamos a mensagem original, antes de ser encriptada, de *purotexto* (*plaintext*). Já a versão encriptada de uma mensagem é chamada de *cifrotexto* (*ciphertext*).

Um cifrotexto não deve revelar nada sobre o purotexto original, de modo que emissor e receptor possam se comunicar sigilosamente mesmo sobre um canal inseguro. Na criptografia moderna, os algoritmos utilizados para encriptação e decriptação são publicamente conhecidos, de modo que a segurança dos cifrotextos gerados se baseia no sigilo de uma *chave*. A chave é utilizada para encriptar e decriptar as mensagens trocadas entre emissor e receptor.

Em um esquema criptográfico *simétrico* existe apenas uma chave[10]. Ela é utilizada tanto para encriptar quanto para decriptar e deve ser conhecida pelo emissor e pelo receptor. Aliás, dada a simetria desse esquema, os papéis de emissor e receptor não são bem definidos, uma vez que cada participante pode enviar e receber mensagens com a mesma chave.

Um dos problemas que precisam ser resolvidos para a utilização deste tipo de esquema é a maneira pela qual a chave é gerada e distribuída entre os participantes da comunicação. Isso porque o adversário que obtiver tal chave terá total acesso à comunicação.

Já em um esquema *assimétrico*, também chamado de *esquema de chave pública*, existem duas chaves relacionadas entre si, uma pública e outra secreta. A chave pública é usada para encriptar uma mensagem. Somente sua contraparte secreta pode decriptar o cifrotexto resultante. Dessa forma, o par de chaves deve ser gerado pelo *receptor* da mensagem. Ele também deve providenciar para que o emissor receba a chave pública. De maneira geral, a chave pública pode ser conhecida mesmo por quem não vai participar da comunicação. A chave secreta, no entanto, deve ser conhecida apenas pelo receptor.

Note que, em um esquema assimétrico, os papéis de receptor e emissor são bem definidos e não permutáveis entre si.

Neste trabalho estaremos interessados apenas em esquemas criptográficos assimétricos. Apresentamos essa definição a seguir.

Definição 3.23 (Esquema de criptografia assimétrico - Definição 11.1 de KATZ e LINDELL (2014))**.** *Um* ***esquema de criptografia assimétrico****, ou* ***esquema criptográfico assimétrico****, é um trio de algoritmos* (Gen, Enc, Dec) *probabilísticos de tempo polinomial tais que:*

1. *O algoritmo Gen recebe um parâmetro de segurança* 1^n *como entrada e dá como saída um par de chaves* (pk, sk)*. A chave pk é chamada de* ***chave pública*** *e sk, de* ***chave secreta****.*

[10] Ou, de maneira equivalente, a chave possui somente uma parte.

2. *O algoritmo Enc recebe como entrada uma chave pública pk e uma mensagem m de algum espaço de purotextos, o qual pode depender de pk. Ele dá como resultado um cifrotexto $c = Enc(pk, m)$.*

3. *O algoritmo Dec recebe como entrada uma chave secreta sk e um cifrotexto c e dá como saída uma mensagem m ou um símbolo especial $\perp$ simbolizando falha. Podemos assumir sem perder generalidade que Dec é determinístico*[11]*. Denotamos o resultado da operação de decriptação como $m = Dec(sk, c)$.*

Como requisito de correção, exige-se que, para todo n, todo par (pk, sk) gerado por $Gen(1^n)$ e toda mensagem m retirada do espaço de purotextos, a seguinte propriedade se verifique:

$$Dec(sk, Enc(pk, m)) = m$$

Fora a importante exigência de correção, a Definição 3.23 é basicamente sintática. No entanto, para que um esquema criptográfico seja útil, além de correto, ele precisa, naturalmente, ser *seguro*. Para definir segurança vamos utilizar um *experimento*. Nele, consideramos um adversário *abelhudo* (ou passivo), ou seja, um que tem apenas o poder de observar mensagens, e não o de alterá-las. Pedimos ao adversário que forneça duas mensagens à sua escolha, mas desde que tenham o mesmo tamanho. Sorteamos uma dessas mensagens e a encriptamos. Enviamos ao adversário o cifrotexto resultante e ele deve adivinhar qual mensagem foi encriptada. Definimos esse experimento a seguir.

Definição 3.24 (Experimento da indinstinguibilidade de cifrotextos na presença de um abelhudo)**.** *Seja $\Gamma = (Gen, Enc, Dec)$ um esquema criptográfico assimétrico, $n \in \mathbb{N}$ um parâmetro de segurança e A um adversário abelhudo. O procedimento a seguir, denotado por $Ind_{A,\Gamma}(n)$, é chamado de* ***experimento da indistinguibilidade de crifrotextos na presença de um abelhudo****.*

1. *$Gen(1^n)$ é executado para gerar as chaves (pk, sk).*

2. *O adversário A recebe pk e escolhe duas mensagens m_0, m_1 com $|m_0| = |m_1|$.*

3. *Um bit $b \in \{0,1\}$ é escolhido aleatoriamente. O cifrotexto $c = Enc(pk, m_b)$ é calculado e enviado para A. O cifrotexto c é chamado de cifrotexto-desafio.*

4. *O adversário A dá como saída um bit $b' \in \{0,1\}$.*

5. *O resultado do experimento é 1 se $b' = b$ e 0 caso contrário.*

O resultado do experimento $Ind_{A,\Gamma}(n)$ é 1 quando o adversário é bem sucedido em adivinhar qual das mensagens fornecidas foi encriptada. A probabilidade disso acontecer é

[11] Isso porque podemos assumir que qualquer fonte de aleatoriedade necessária para decriptar c está embutida no próprio c.

pelo menos $\frac{1}{2}$, pois o adversário pode simplesmente gerar uma resposta aleatória. Para que um esquema criptográfico tenha a propriedade de *indistinguibilidade de cifrotextos na presença de um abelhudo*, essa probabilidade não pode exceder $\frac{1}{2}$ em um valor não desprezível. Registramos essa definição a seguir.

Definição 3.25 (Indinstinguibilidade de cifrotextos na presença de um abelhudo - Definição 11.2 de KATZ e LINDELL (2014))**.** *Um esquema criptográfico assimétrico* $\Gamma = (Gen, Enc, Dec)$ *tem a propriedade de* ***indinstinguibilidade de cifrotextos na presença de um abelhudo*** *se, para todo adversário probabilístico A de tempo polinomial e todo parâmetro de segurança* $n \in \mathbb{N}$*, existe uma função* μ *desprezível (em n) tal que:*

$$Pr[Ind_{A,\Gamma}(n) = 1] \leq \frac{1}{2} + \mu(n)$$

A probabilidade acima leva em conta a aleatoriedade de A, assim como a utilizada para gerar as chaves, escolher b e encriptar m_b*.*

A Definição 3.25 estabelece que um adversário não deve conseguir extrair qualquer informação parcial do purotexto a partir do cifrotexto, mesmo que ele saiba os possíveis valores para o purotexto. De fato, nem mesmo se esses possíveis valores para o purotextos forem cuidadosamente escolhidos por ele próprio.

Apesar de simples, a Definição 3.25 é equivalente a outras formulações aparentemente mais fortes. Por exemplo, ela garante que Γ é seguro contra *ataques de purotextos escolhidos* (*chosen-plaintext attacks*). O experimento relacionado a esse ataque é semelhante ao da Definição 3.24, mas com a diferença de que o adversário A pode utilizar o algoritmo de encriptação como uma "caixa-preta" para realizar uma quantidade polinomial de consultas. Em cada uma dessas consultas ele obtém um cifrotexto correspondente a um purotexto de sua escolha sem que, naturalmente, a chave secreta de encriptação seja revelada.

A Definição 3.25 também garante a Γ a propriedade de indistinguibilidade de *múltiplos* cifrotextos. Essa propriedade se baseia em um experimento semelhante ao da Definição 3.24, mas com a diferença de que o adversário fornece dois *vetores* de mensagens. Intuitivamente isso significa que o adversário tem acesso a vários cifrotextos correspondentes a purotextos escolhidos por ele e gerados sobre a mesma chave. É notável que essa equivalência só se dê para esquemas *assimétricos*[12].

Por fim, a Definição 3.25 é também equivalente à chamada *segurança semântica*. Em linhas gerais, um esquema é semanticamente seguro se a probabilidade de um adversário calcular uma propriedade f de um purotexto dados o cifrotexto-desafio c e uma outra propriedade h for a

[12]Isso ocorre porque esquemas *simétricos* podem ter indistinguibilidade de cifrotextos mesmo que o método de encriptação utilizado seja determinístico. Mas, para se ter indinstinguibilidade de múltiplos cifrotextos, é necessário que o método de encriptação seja probabilístico. Um esquema assimétrico, por outro lado, precisa de um método probabilístico de encriptação já para alcançar a indistinguibilidade de cifrotextos. Para mais sobre essa diferença, ver o Teorema 10.10 e a Afirmação 3.20 de KATZ e LINDELL (2014).

mesma de quando o adversário não tem acesso a c. As propriedade f, h são funções polinomiais calculadas sobre o purotexto[13].

3.8.1 Construção a partir de Funções Arapuca

No Capítulo 4 utilizaremos o fato de que esquemas criptográficos assimétricos podem ser construídos a partir de funções arapuca. Deixamos registrado como isso pode ser feito na definição a seguir.

Definição 3.26 (Um esquema criptográfico assimétrico a partir de uma família de funções arapuca - Construção 5.3.13 de GOLDREICH (2004))**.** *Seja $\{f_\alpha : D_\alpha \to D_\alpha\}$ uma família de funções e (I, S, F, F^{-1}) os algoritmos de acordo com a Definição 3.6. Seja ainda b um predicado núcleo-duro para essa família de funções arapuca. Definimos a implementação dos algoritmos de um esquema criptográfico assimétrico $\Gamma = (Gen, Enc, Dec)$ como a seguir:*

1. *(Gen:) A geração das chaves pública e secreta consiste em invocar I para obter um índice α para uma função f_α e uma arapuca τ correspondente. O índice α é a chave pública e a arapuca τ, a secreta. Isto é, $Gen(1^n) = I(1^n)$.*

2. *(Enc:) Para se encriptar um bit σ, o algoritmo Enc seleciona um elemento $r \in D_\alpha$ aleatoriamente e define o cifrotexto como $(f_\alpha(r), \sigma \oplus b(r))$. Ou seja, $Enc(\alpha, \sigma) = (F(\alpha, r), \sigma \oplus b(r))$, em que $r = S(\alpha)$.*

3. *(Dec:) Para decriptar um cifrotexto (y, z), Dec calcula $z \oplus b(f_\alpha^{-1}(y))$, em que, naturalmente, f_α é invertida com o auxílio da arapuca τ. Ou seja, $Dec(\tau, (y, z)) = z \oplus b(F^{-1}(\tau, y))$.*

Podemos facilmente verificar que o esquema da Definição 3.26 atende ao requisito de corretude. De fato:

$$\begin{aligned} D(\tau, E(\alpha, \sigma)) &= D(\tau, (F(\alpha, S(\alpha)), \sigma \oplus b(S(\alpha))) \\ &= (\sigma \oplus b(S(\alpha))) \oplus b(F^{-1}(\tau, F(\alpha, S(\alpha)))) \\ &= \sigma \oplus b(S(\alpha)) \oplus b(f_\alpha^{-1}(f_\alpha(S(\alpha)))) \\ &= \sigma \oplus b(S(\alpha)) \oplus b(S(\alpha)) = \sigma \end{aligned} \tag{3.1}$$

A segurança (de acordo com a Definição 3.25) do esquema da Definição 3.26 se baseia na dificuldade de se inverter uma função arapuca sem a arapuca correspondente e, além disso, na dificuldade de se calcular $b(r)$ dado apenas $f_\alpha(r)$. Uma prova formal de que esse esquema tem indistinguibilidade de cifrotextos pode ser encontrada na Proposição 5.3.14 de GOLDREICH (2004).

[13] Definição 3.3 de KATZ e LINDELL (2014) ou Definição 5.2.2 de GOLDREICH (2004).

3.9 Assinaturas Digitais

Um esquema de assinatura digital permite que um emissor assine mensagens por meio de sua chave *secreta*, também chamada de chave de assinatura. O receptor pode utilizar a chave *pública* do emissor para verificar que a mensagem não foi adulterada e que ela de fato foi enviada pelo portador da chave secreta correspondente. Vale ressaltar que a proteção ao sigilo das mensagens não é um requisito de um esquema de assinatura.

Um esquema de assinaturas digitais é considerado seguro se ele gera assinaturas *inforjáveis*. Isto é, se não é possível gerar assinaturas válidas em relação a uma determinada chave secreta *sk sem a posse* dessa chave. Essa propriedade deve se verificar mesmo que um atacante tenha acesso a outras assinaturas (válidas) feitas com *sk*.

Um esquema inforjável confere *integridade* e *não-repúdio* às mensagens geradas. Integridade é a garantia de que o receptor não pode ser levado a aceitar uma mensagem diferente daquela enviada pelo emissor. Ou seja, ou a mensagem chega inalterada ou o receptor pode detectar que ela foi adulterada. A maneira de se detectar tal adulteração é através da verificação da assinatura. De fato, quando uma mensagem é adulterada por um atacante, sua assinatura torna-se inválida. Além disso, tal atacante não pode ser capaz de gerar uma assinatura válida para a mensagem adulterada. Se uma dessas afirmações não fosse verdadeira, i.e., se a assinatura não se tornasse inválida ou o atacante gerasse uma assinatura válida, então o esquema não seria inforjável.

O não-repúdio, por sua vez, diz respeito à impossibilidade de um emissor negar que assinou determinada mensagem. De fato, se existe uma assinatura validada por uma chave pública *pk*, então ou ela foi de fato gerada pelo portador da chave secreta correspondente a *pk* ou ela foi forjada, contradizendo a inforjabilidade do esquema.

Prosseguimos com a definição formal de esquema de assinatura digital.

Definição 3.27 (Esquemas de assinaturas digitais)**.** *Um* ***esquema de assinaturas digitais****, ou simplesmente* ***esquema de assinaturas****, é um trio de algoritmos* $(Gen, Assina, VrfAssinatura)$ *de tempo polinomial tais que:*

(i) O algoritmo probabilístico Gen recebe um parâmetro de segurança 1^n *e retorna um par de chaves* (pk, sk) *para verificação e assinatura, respectivamente.*

(ii) O algoritmo probabilístico Assina recebe uma chave secreta sk e uma cadeia $\alpha \in \{0,1\}^*$ *e produz uma assinatura* $A_{sk,\alpha}$*. Dizemos que* $A_{sk,\alpha}$ *é uma assinatura sobre* α *gerada com sk.*

(iii) O algoritmo determinístico VrfAssinatura recebe uma chave pública pk, uma cadeia α *e uma suposta assinatura* $A_{sk,\alpha}$*. Ele retorna* 1 *se* $A_{sk,\alpha}$ *é uma assinatura válida sobre* α *gerada com a chave secreta sk correspondente à pk. Caso contrário, ele retorna 0.*

(iv) Para todo par (pk, sk) *do domínio de* $Gen(1^n)$ *e toda cadeia* $\alpha \in \{0,1\}^*$*, os algoritmos*

Assina e *VrfAssinatura satisfazem a seguinte condição:*

$$Pr[VrfAssinatura(pk, \alpha, Assina(sk, \alpha)) = 1] = 1$$

em que essa probabilidade é calculada levando-se em conta as escolhas aleatórias de *Assina* e *VrfAssinatura*.

A definição acima é quase que exclusivamente sintática. Vale ressaltar, no entanto, o item (iv), semelhante ao requisito de correção dos esquemas de criptografia assimétricos. Ele estabelece que *Assina*, probabilístico, sempre gera assinaturas válidas, i.e., assinaturas que fazem *VrfAssinatura* retornar 1. No entanto, fica em aberto o que acontece quando assinaturas *inválidas* são apresentadas a *VrfAssinatura*. De fato, para que um esquema de assinaturas seja útil, é necessário que *VrfAssinatura* rejeite assinaturas inválidas, ou que aceite, no máximo, apenas uma quantidade desprezível delas. Esse requisito é capturado pela definição de inforjabilidade apresentada mais adiante.

A segurança, ou inforjabilidade, de um esquema de assinaturas pode ser definida com base em um experimento. Nele, consideramos um adversário A com acesso oracular ao algoritmo $Assina(sk, \cdot)$. Ou seja, ainda que ele não conheça a chave secreta sk utilizada no experimento, ele pode consultar o oráculo para obter assinaturas válidas geradas com essa chave sobre mensagens de sua escolha. O desafio proposto ao adversário é o de criar uma assinatura válida sobre uma mensagem que não tenha sido consultada. Definimos esse experimento a seguir.

Definição 3.28 (O experimento de inforjabilidade)**.** *Seja* $\Gamma \stackrel{def}{=} (Gen, Assina, VrfAssinatura)$ *um esquema de assinaturas,* A *um adversário e* $n \in \mathbb{N}$ *um parâmetro de segurança. O procedimento a seguir, denotado por* $Inforj_{A,\Gamma}(n)$*, é chamado de* ***experimento de inforjabilidade****.*

1. *Gen é executado para se obter as chaves* (sk, pk)*.*
2. *O adversário* A *possui acesso a* pk *e ao oráculo* $Assina(sk, \cdot)$*. Ele pode fazer uma quantidade polinomial (em* n*) de consultas oraculares. O adversário dá como resposta um par* (m, σ)*.*
3. *Seja* Q *o conjunto das mensagens que foram consultadas por* A *na etapa anterior. O resultado do experimento é* 1 *se* $m \notin Q$ *e* $VrfAssinatura(pk, m, \sigma) = 1$ *e* 0 *caso contrário.*

O experimento $Inforj_{A,\Gamma}(n)$ tem resultado 1 sempre que o adversário A consegue gerar uma assinatura válida sem acesso à chave sk. Naturalmente, um esquema de assinaturas é inforjável se qualquer adversário tem apenas uma probabilidade desprezível de ser bem sucedido em $Inforj_{A,\Gamma}(n)$. Registramos isso na definição a seguir.

Definição 3.29 (Esquema de assinatura inforjável - Definição 12.2 de KATZ e LINDELL (2014))**.** *Um esquema de assinatura* Γ *é* ***inforjável*** *se, para todo* $n \in N$ *e todo adversário probabilístico* A *de tempo polinomial, existe uma função* μ *desprezível (em* n*) tal que:*

$$Pr[Inforj_{A,\Gamma}(n) = 1] \leq \mu(n)$$

Como observação final, esquemas de assinaturas inforjáveis podem ser construídos a partir de funções unidirecionais e funções *hash* resistente a colisões. Esse resultado pode ser encontrado, por exemplo, no Corolário 6.4.10 de GOLDREICH (2004).

4 COMPUTAÇÃO MULTILATERAL SEGURA

O objetivo do capítulo atual é apresentar o resultado teórico, já bem conhecido na criptografia, que estabelece que qualquer funcionalidade multilateral pode ser computada de forma segura através de um protocolo distribuído. Esse resultado, por sua vez, é fundamental para o restante do trabalho.

Toda a apresentação do capítulo corrente se baseia no Capítulo 7 (*General Cryptographic Protocols*) de GOLDREICH (2004). Além de realizar mudanças gerais na notação e na organização da apresentação, também decidimos remover provas formais e substituí-las por argumentações mais intuitivas sobre a correção dos resultados e das construções. Consideramos que essas decisões facilitam a leitura e não comprometem o entendimento do leitor.

O capítulo está dividido da seguinte forma. Na Seção 4.1 definimos conceitos importantes para o entendimento do capítulo como um todo. Na Seção 4.2 mostraremos como construir um protocolo para computar *privadamente* uma funcionalidade multilateral, isto é, com segurança contra adversários semi-honestos (ou passivos). Na Seção 4.3 exibiremos um *compilador* para transformar um protocolo construído conforme os resultados da Seção 4.2 em um protocolo seguro contra adversários maliciosos, os quais são mais poderosos. Por fim, ainda na Seção 4.3, juntaremos todos os resultados anteriores para estabelecermos o nosso resultado final: um procedimento para computar uma funcionalidade qualquer de forma segura na presença de um adversário malicioso.

4.1 Definições Preliminares

Inicialmente, vamos definir o conceito de m-funcionalidade multilateral.

Definição 4.30 (m-funcionalidade multilateral)**.** *Uma m-funcionalidade multilateral f, ou apenas m-funcionalidade, é um processo probabilístico que mapeia uma sequência de entrada $\bar{x} \stackrel{def}{=} (x_1,\ldots,x_m) \in (\{0,1\}^*)^m$ em uma sequência de saída $f(\bar{x}) = f(x_1,\ldots,x_m) \stackrel{def}{=} (f_1(\bar{x}),\ldots,f_m(\bar{x}))$.*

Temos interesse no cenário em que m participantes P_i, com $1 \leq i \leq m$, desejam computar uma m-funcionalidade f de maneira distribuída. A sequência de entrada $\bar{x}$ é composta, nesse caso, pela junção das entradas locais x_i de cada um dos participantes P_i. Ao final da computação, cada participante P_i espera receber sua parte $f_i(\bar{x})$ do resultado $f(\bar{x})$. Queremos nos concentrar em como os participantes podem realizar essa computação de maneira segura.

4.1.1 Segurança

Vamos definir o conceito de segurança gradualmente, partindo de uma noção mais intuitiva até chegar na definição que utilizaremos de fato.

Inicialmente, podemos entender que a computação distribuída de uma m-funcionalidade f é segura se ela oferecer uma série de garantias aos seus participantes. Um exemplo de

garantia naturalmente desejável é a da *correção*, ou seja, a de que, ao final da computação, cada participante receberá uma resposta coerente com a execução de f sobre as entradas dadas. Outra garantia importante é a *privacidade* das entradas. Ela pode ser vista como a impossibilidade de um participante ter acesso a qualquer informação sobre uma entrada, ou saída, que não a sua própria.

Essa noção de segurança, no entanto, impõe algumas dificuldades ao seu tratamento formal. Considere, por exemplo, uma 2-funcionalidade t tal que $t(x_1,x_2) \stackrel{def}{=} (x_2,x_1)$. Essa funcionalidade especifica uma troca, ou seja, ao final de sua execução, cada participante aprende a entrada do outro. Nesse caso, a correção ainda é uma propriedade absolutamente necessária, porém a noção de privacidade já não faz mais sentido da maneira pela qual foi pensada anteriormente. Isso porque qualquer execução de t sempre teria sua privacidade quebrada e, portanto, não poderia ser segura. Uma abordagem para contornar esse problema seria adequar a definição de privacidade às características *particulares* de t. No entanto, isso não é desejável, uma vez que impossibilita uma definição geral de segurança. Além disso, possivelmente teríamos que estender esse tratamento especial a outras garantias.

Devido a esse tipo de dificuldade, a segurança da computação de uma m-funcionalidade f é definida com base no *paradigma de simulação*. Nele, dois modelos de computação de f são comparados. No primeiro deles, chamado de *modelo ideal*, os participantes computam f através de um protocolo trivial com o auxílio de uma Terceira Parte Confiável (TPC)[1]. Nesse protocolo, os participantes entregam suas entradas diretamente à TPC, a qual fica responsável por computar f sobre as entradas recebidas e distribuir as saídas de forma adequada. Note que, com o uso de uma TPC, cada participante aprende sobre as entradas e saídas dos demais apenas o que é possível extrair das suas próprias. Adicionalmente, os participantes recebem saídas corretas, uma vez que elas são computadas por uma parte honesta. Como o nome sugere, o modelo ideal é a forma mais segura de se computar f multilateralmente.

No *modelo real*, em oposição ao ideal, o protocolo conduzido pelos participantes tem como objetivo emular o comportamento de uma TPC, a qual não existe de fato. Essa interação entre os participantes proporcionada pelo protocolo (não trivial) possibilita comportamentos desonestos que inexistem no modelo ideal.

Em ambos os modelos de computação supõe-se a existência de uma parte externa, chamada de *adversário*, que controla o comportamento de um determinado subconjunto de participantes. Apenas os participantes desse subconjunto podem assumir estratégias desonestas durante a execução de um protocolo. No cenário em que o adversário não controla participante algum, a execução só pode ser atacada com uma estratégia totalmente passiva. Por isso, esse cenário é de interesse. Por outro lado, o caso em que o adversário controla todos os participantes não é de interesse. Um adversário é chamado de real ou ideal de acordo com o modelo de

[1] Chamamos de trivial porque os participantes não interagem entre si. Existe apenas interação entre os participantes e a TPC. Dependendo de como seja definido, o modelo pode admitir também a interação entre a TPC e um adversário.

computação no qual ele atua.

Seguindo o paradigma de simulação, um protocolo que computa uma funcionalidade f é considerado seguro se o comportamento de qualquer adversário real pode ser simulado por um adversário ideal. Dessa forma, a definição não leva em conta particularidades de f, mas exige que a qualquer ataque ao modelo real corresponda um ataque, com resultados similares, ao modelo ideal. Um protocolo seguro, portanto, contém todas as vulnerabilidades inerentes à funcionalidade computada (vide o exemplo de t dado anteriormente) sem, no entanto, acrescentar nenhuma.

Nos contentaremos, por enquanto, com essa definição de segurança que, embora informal, captura o conceito que formalizaremos mais adiante.

4.1.2 Restrição sobre as Funcionalidades

Tendo já uma noção intuitiva do conceito de segurança, podemos abordar um detalhe técnico sobre as funcionalidades que vamos tratar. Imagine uma funcionalidade f tal que a saída do participante P_1 dependa do valor $|x_2|$, em que x_2 é a entrada de P_2. Nesse caso, no modelo de computação ideal, P_1 só pode inferir $|x_2|$ a partir do valor de sua saída, pois a TPC não revela essa informação. Nesse caso, um protocolo seguro também não pode relevar essa informação, o que só o torna mais difícil de ser construído.

Por essa razão, limitaremos o nosso escopo a funcionalidades que recebem entradas todas do mesmo tamanho. Essa restrição, no entanto, tem apenas um papel simplificador. Com técnicas adequadas de preenchimento (*padding*), pode-se transformar uma funcionalidade arbitrária em uma que satisfaça essa restrição de igualdade de entradas.

4.1.3 Modelo de Segurança

Para que o comportamento de um adversário real possa ser simulado por um adversário ideal, ambos precisam estar, naturalmente, sob as mesmas condições. Essas condições são definidas pelo *modelo de segurança*, que é formado por uma série de suposições e restrições impostas tanto sobre o adversário quanto sobre o ambiente de execução. Os parâmetros do modelo de segurança relevantes para o restante desse capítulo são descritos a seguir.

- *Suposições de Inicialização*. São condições que devem ser satisfeitas antes do início da execução de um protocolo, como por exemplo a existência de uma infraestrutura de chave pública.

- *Canal de Comunicação*. O canal de comunicação é o meio pelo qual os participantes (honestos) trocam mensagens entre si. A especificação desse canal define o poder que o adversário possui em relação a essas mensagens. Destacamos os dois modelos de canal utilizados neste capítulo. O primeiro deles estabelece a existência de uma canal de *broadcast*, o que significa que toda mensagem enviada por uma parte honesta

pode ser coletada pelo adversário. De maneira oposta, um canal ponto-a-ponto privado retira essa habilidade do adversário. Em ambos os modelos, os canais são considerados confiáveis, de modo que o adversário não é capaz de alterar, omitir ou forjar mensagens entre participantes honestos.

- *Poder Computacional*. O poder computacional do adversário determina um limite assintótico para as estratégias que ele pode empregar. Neste trabalho, a estratégia de um adversário é sempre representada por um algoritmo probabilístico de tempo polinomial (no tamanho das entradas).

- *Formação de Conluio*. Conluio é o subconjunto dos participantes controlado pelo adversário durante um protocolo. A formação do conluio pode se dar de maneira adaptativa ou não adaptativa. Uma estratégia adaptativa permite ao usuário alterar o conjunto de participantes desonestos durante a execução do protocolo. Por outro lado, uma estratégia não adaptativa, também chamada de estática, requer que o adversário defina quais participantes fazem parte do conluio antes do início do protocolo. Neste trabalho consideraremos apenas adversários não adaptativos.

- *Estratégia de Ataque*. A estratégia de ataque define os meios pelos quais o adversário pode agir para comprometer a execução de um protocolo. Iremos considerar duas estratégias possíveis: a semi-honesta, ou passiva, e a maliciosa, ou ativa. Um adversário semi-honesto deve seguir corretamente o protocolo, mas pode manter um registro de todas as mensagens às quais teve acesso durante a execução, de modo a estudá-las após o término do protocolo. Por outro lado, um adversário ativo não segue, necessariamente, as regras do protocolo. Ele pode, por exemplo, mudar suas entradas, enviar mensagens malformadas ou fora de ordem, mudar suas saídas e suspender a execução prematuramente.

- *Quebra da Justiça*. Justiça é a garantia de que ou todo participante recebe o seu resultado da computação ou nenhum recebe. Iremos assumir, neste trabalho, que a quebra de justiça não é uma quebra de segurança. De maneira geral, a justiça pode ser garantida sob a hipótese de que a maioria dos participantes do protocolo é honesta.

4.1.4 Segurança Contra Adversários Semi-Honestos

Nesta subseção temos como objetivo definir formalmente a noção de segurança contra adversários *semi-honestos*.

Um adversário semi-honesto, como mencionado no modelo de segurança, instrui os participantes sob seu controle a agir de acordo com as regras estabelecidas pelo protocolo. Esses participantes, no entanto, mantêm o registro de suas computações intermediárias e de todas as mensagens a que eles têm acesso. Esse registro, nas mãos do adversário, pode representar um risco à segurança do protocolo ou, mais especificamente, à privacidade dos participantes.

O modelo de segurança considerado nessa subseção tem como característica, além de um adversário semi-honesto, um canal de comunicação ponto-a-ponto privado entre os participantes honestos. Isso significa que as mensagens trocadas entre eles não podem ser coletadas pelo adversário.

Para chegarmos na definição formal de computação segura para o presente modelo de segurança, precisamos antes passar pelo importante conceito de visão da execução de um protocolo.

Definição 4.31 (Visão da execução de um protocolo - adversário semi-honesto e canal privado)**.** *Seja f uma m-funcionalidade e Π um protocolo que computa f. Definimos $VIS\tilde{A}O_i^{\Pi}(\bar{x}) \stackrel{def}{=} (x_i, r_i, msg_1, \cdots, msg_z)$ como a visão do participante P_i durante a execução de Π sobre a entrada $\bar{x} \stackrel{def}{=} (x_1, \cdots, x_m)$, em que x_i é a entrada fornecida por P_i, r_i é uma cadeia de bits usada por P_i como fonte de aleatoriedade e os elementos msg_j são as mensagens destinadas a P_i durante a execução de Π. Note que a saída do participante P_i é totalmente determinada por $VIS\tilde{A}O_i^{\Pi}$.*

A visão da execução, como apresentada na Definição 4.31, é formada por todas as informações que um participante tem acesso durante a execução de um protocolo. Além da sua própria entrada, da sua saída e da sua fonte de aleatoriedade, o participante é capaz de registrar as mensagens por ele recebidas. Note que mensagens não destinadas ao participante não aparecem em sua visão por conta da hipótese da existência de um canal privado. A junção das visões dos participantes desonestos constitui o registro que um adversário semi-honesto mantém da execução do protocolo.

Com a noção de visão, já podemos definir o que significa um protocolo computar uma funcionalidade de maneira segura. Fazemos isso na Definição 4.32 a seguir.

Definição 4.32 (Computação segura de uma funcionalidade por um protocolo - adversário semi-honesto e canal privado)**.** *Seja f uma m-funcionalidade, $\bar{x} \stackrel{def}{=} (x_1, \cdots, x_m)$ uma sequência de entrada, Π um protocolo que computa f e $f_i(\bar{x})$ o i-ésimo elemento de $f(\bar{x})$. Para todo conjunto $I = \{i_1, \cdots, i_t\} \subseteq \{1, \cdots, m\}$, definimos $f_I(\bar{x}) \stackrel{def}{=} (f_{i_1}(\bar{x}), \cdots, f_{i_t}(\bar{x}))$ e $VIS\tilde{A}O_I^{\Pi}(\bar{x}) \stackrel{def}{=} (VIS\tilde{A}O_{i_1}^{\Pi}(\bar{x}), \cdots, VIS\tilde{A}O_{i_t}^{\Pi}(\bar{x}))$. Um protocolo Π computa f com segurança contra adversários semi-honestos, supondo a existência de um canal privado, se existe um algoritmo probabilístico de tempo polinomial S tal que, para todo $C \subseteq \{1, \cdots, m\}$ representando um conluio de participantes:*

- *Se f é uma funcionalidade determinística:*

$$\{S(C, (x_{i_1}, \ldots, x_{i_t}), f_C(\bar{x}))\}_{\bar{x} \in (\{0,1\}^*)^m} \stackrel{c}{\equiv} \{VIS\tilde{A}O_C^{\Pi}(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^m} \quad (4.1)$$

- *Caso geral:*

$$\{S(C, (x_{i_1}, \ldots, x_{i_t}), f_C(\bar{x})), f(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^m} \stackrel{c}{\equiv} \{VIS\tilde{A}O_C^{\Pi}(\bar{x}), SA\acute{I}DA^{\Pi}(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^m} \quad (4.2)$$

em que $SA\acute{I}DA_{\Pi}(\bar{x})$ denota a saída do protocolo Π quando executado sobre as entradas $\bar{x}$.

A Definição 4.32 estabelece que, em um protocolo seguro, toda a informação obtida pelo adversário, i.e., a junção $VIS\tilde{A}O_C^{\Pi}$ das visões dos participantes do conluio C, pode ser eficientemente computada a partir das entradas e das saídas dos participantes que ele controla. Isso é decorrência da existência do algoritmo S que, em tempo polinomial e alimentado apenas por essas entradas e saídas, consegue gerar conteúdo indistinguível de $VIS\tilde{A}O_C^{\Pi}$. Em particular, isso significa que a visão de um adversário pode ser eficientemente simulada sem acesso a mensagem alguma. Em um protocolo seguro, portanto, as mensagens a que o adversário tem acesso não lhe dão qualquer conhecimento adicional[2].

Ressaltamos que ambas as equações da Definição 4.32 são equivalentes quando tratam de funcionalidades determinísticas. Isso segue do fato de que $f(\bar{x}) = SA\acute{I}DA^{\Pi}(\bar{x})$ quando f é determinística e, portanto, a Equação (4.2) não acrescenta nenhuma informação a mais. No entanto, no caso de funcionalidades probabilísticas, a Equação (4.2) é de fato mais restritiva. Isso porque ela obriga que S não apenas simule uma visão que se encaixe na distribuição probabilística de $VIS\tilde{A}O_C^{\Pi}$, mas também que preserve a relação existente entre o que cada participante (desonesto) tem acesso e a saída do protocolo como um todo.

Para entender melhor como a Equação (4.2) é mais restritiva, considere a 2-funcionalidade (probabilística) f tal que $f(1^n,1^n) \stackrel{def}{=} (r,\lambda)$, com $r \in \{0,1\}^n$ uniformemente distribuído e λ representando a cadeia vazia. Agora considere o protocolo Π que computa f da seguinte maneira: o participante P_1 sorteia r uniformemente, envia r para P_2 e define r como sua saída. Claramente Π computa f, mas não é seguro, pois P_2 aprende a saída de P_1, o que não aconteceria numa execução ideal.

Mesmo Π não sendo seguro, é possível criar um algoritmo S que satisfaça a Equação (4.1), considerando $C = \{2\}$. Note que $VIS\tilde{A}O_C^{\Pi} = (1^n,\lambda,r)$, pois r é uma mensagem recebida por P_2 e $2 \in C$. Vamos definir um algoritmo S tal que $S(C,1^n,\lambda) \stackrel{def}{=} (1^n,\lambda,r')$, com r' gerado uniformemente e não necessariamente igual ao recebido por P_2 em Π. Perceba que, desde que r' tenha a mesma distribuição que r, temos que $\{(1^n,\lambda,r')\} \stackrel{c}{\equiv} \{(1^n,\lambda,r)\}$, satisfazendo a Equação (4.1). Mas, em relação à Equação (4.2), temos que $\{(1^n,\lambda,r'),f(1^n,1^n)\} \stackrel{c}{\not\equiv} \{(1^n,\lambda,r),SA\acute{I}DA^{\Pi}(1^n,1^n)\}$. Note que é sempre verdade que $SA\acute{I}DA^{\Pi}(1^n,1^n) = (r,\lambda)$, mas nem sempre $f(1^n,1^n) = (r',\lambda)$. Ou seja, S falha em simular a relação existente entre a saída dos participantes do conluio C e a saída completa do protocolo. Com base nisso, podemos distinguir $\{S(C,1^n,f_C(1^n,1^n)),f(1^n,1^n)\}$ de $\{VIS\tilde{A}O_C^{\Pi}(1^n,1^n),SA\acute{I}DA^{\Pi}(1^n,1^n)\}$.

Apenas para a conveniência em futuras referências, vamos resumir a Definição 4.32 a seguir.

Definição 4.33 (Computação privada de uma funcionalidade)**.** *Seja f uma m-funcionalidade. Se Π é um protocolo que computa f de forma segura, conforme as condições da Definição 4.32, definimos, então, que Π computa f privadamente (ou de forma privada).*

[2]Pela definição de indistinguibilidade (Definição 3.3), o adversário pode, na verdade, extrair informações adicionais das mensagens trocadas no protocolo, mas apenas com uma probabilidade desprezível.

Usamos a Definição 4.33 para estabelecer que, no modelo de segurança considerado, com um adversário semi-honesto e um canal privado, a segurança de um protocolo equivale à sua privacidade. Isso porque a única habilidade que um adversário semi-honesto possui é a de bisbilhotar mensagens e, ainda assim, devido à suposição de um canal privado, somente as recebidas pelos participantes desonestos. Dessa forma, o único tipo de ataque que tal adversário pode conduzir é justamente contra a privacidade dos participantes.

4.1.5 Segurança Contra Adversários Maliciosos

O nosso objetivo nessa subseção é, novamente, definir a noção de computação segura, mas dessa vez contra um tipo mais poderoso de adversário: o adversário malicioso (ou ativo).

O modelo de segurança que consideraremos agora impõe menos restrições ao adversário que o da subseção anterior. Primeiramente, admitimos a existência de um canal de *broadcast*, permitindo, portanto, que o adversário tenha acesso a todas as mensagens trocadas entre quaisquer participantes. O aspecto mais relevante desse modelo, no entanto, é a suposição de adversários maliciosos (ou ativos).

O adversário malicioso é bem mais poderoso que o semi-honesto e pode, por exemplo, enviar mensagens malformadas ou fora de ordem e abortar prematuramente a execução. Nesse tipo de situação, em que o adversário aborta o protocolo, os participantes honestos correm o risco de não receberem suas saídas, o que é conhecido como quebra de justiça.

Para incorporar na nossa definição de segurança a possibilidade do adversário abortar prematuramente, precisamos modificar ligeiramente a noção de modelo ideal de computação. Lembramos que esse modelo é caracterizado por um protocolo trivial que computa uma funcionalidade f com o auxílio de uma TPC e é nossa referência de computação segura.

Registramos essa modificação na Definição 4.34 a seguir.

Definição 4.34 (Modelo ideal de computação - adversário malicioso e canal de *broadcast*)**.** *Seja f uma m-funcionalidade, $\bar{x}$ uma sequência de entrada para f, $C \stackrel{def}{=} \{c_1, \ldots c_t\} \subseteq \{1, \ldots, m\}$ o conluio de participantes desonestos e $f_C(\bar{x}) \stackrel{def}{=} (f_{c_1}(\bar{x}), \ldots, f_{c_t}(\bar{x}))$ a sequência de saídas de f para os participantes do conluio. O protocolo (trivial) do modelo ideal de computação para f é definido como a seguir.*

E1. Cada participante entrega sua entrada diretamente à TPC. Admitimos que essa comunicação é feita de forma segura, i.e., o adversário não pode interceptar as mensagens contendo as entradas dos participantes honestos.

E2. A TPC computa o valor $f(\bar{x})$ e entrega ao adversário as saídas $f_C(\bar{x})$ dos participantes do conluio C. O adversário decide sobre a continuidade do protocolo:

- *Se o adversário decidir que o protocolo deve continuar, a TPC entrega as saídas também aos participantes honestos.*

- *Caso contrário, a TPC envia $\perp$ para os participantes honestos, sinalizando que a execução foi abortada.*

O modelo ideal da Definição 4.34 permite ao adversário quebrar a justiça da computação de f. Esse modelo, no entanto, torna-se igual ao discutido na Seção 4.1.1 quando o adversário decide não abortar.

De maneira semelhante à Seção 4.1.4, vamos estabelecer uma definição de segurança baseada em simulação. O conceito de *visão de uma execução*, no entanto, é consideravelmente diferente do definido na seção anterior. Ele precisa ser revisto de modo a explicitar a maior influência do adversário durante toda a execução (real ou ideal). Nesta subseção, definiremos dois tipos de *visões conjuntas*, um para cada tipo de execução.

Começamos definindo, a seguir, a visão conjunta da execução ideal.

Definição 4.35 (Visão conjunta da execução ideal)**.** ***Definição auxiliar:*** *Seja $I \stackrel{def}{=} \{i_1,\ldots,i_t\} \subseteq \{1,\cdots,n\}$ um conjunto qualquer e $\bar{a} \stackrel{def}{=} (a_1,\ldots,a_n)$ uma sequência. Definimos $\bar{a}_I \stackrel{def}{=} (a_{i_1},\ldots,a_{i_t})$ como a sequência formada pelos elementos de $\bar{a}$ cujos índices estão em I.*

Seja f uma m-funcionalidade, $C \stackrel{def}{=} \{c_1,\ldots,c_t\} \subseteq \{1,\cdots,m\}$ o conluio de participantes controlados pelo adversário e $H \stackrel{def}{=} \{1,\cdots,m\} \setminus C$ o conjunto de participantes honestos. Definimos o adversário ideal como o par (C,B), onde B é um algoritmo polinomial probabilístico representando a estratégia desse adversário. Denotaremos por $IDEAL_{f,C,B}(\bar{x})$ a visão conjunta da execução ideal de f sobre a sequência de entrada $\bar{x} \stackrel{def}{=} (x_1,\ldots x_m)$. Ela é definida em dois casos:

$$IDEAL_{f,C,B}(\bar{x}) \stackrel{def}{=} (\perp^{|H|}, B(\bar{x}_C, C, r, f_C(\bar{y}), \perp)) \text{ se o adversário decidir abortar} \quad (4.3)$$

$$IDEAL_{f,C,B}(\bar{x}) \stackrel{def}{=} (f_H(\bar{y}), B(\bar{x}_C, C, r, f_C(\bar{y})) \text{ caso contrário} \quad (4.4)$$

onde r é uma cadeia de bits que serve como fonte de aleatoriedade para B e $\bar{y} \stackrel{def}{=} (y_1,\ldots,y_m)$, com $y_i = x_i$ se $i \in H$ e $y_i = B(\bar{x}_C, C, r)_i$ caso $i \in C$.

A visão conjunta, conforme a Definição 4.35, é uma tupla contendo as saídas dos participantes honestos e desonestos, definida em dois casos. A Equação (4.3) cobre o caso em que o adversário decide suspender a execução. Nesse cenário, os participantes honestos recebem $\perp$ como saída. Já os desonestos recebem o valor $B(\bar{x}_C, C, r, f_C(\bar{y}), \perp)$, definido pela estratégia do adversário com base nas entradas do conluio ($\bar{x}_C$), na fonte de aleatoriedade (r), na saída recebidas da TPC ($f_C(\bar{y})$) e da escolha por suspender a execução ($\perp$).

Note que a funcionalidade f é computada pela TPC sobre a sequência de entrada $\bar{y}$, ao invés de $\bar{x}$, refletindo o poder do adversário de substituir as entradas dos participantes desonestos.

A entrada (possivelmente) modificada y_i de um participante desonesto é definida como $B(\bar{x}_C, C, r)_i$. Ou seja, ela é função apenas das entradas originais dos participantes do conluio ($\bar{x}_C$) e da fonte de aleatoriedade do adversário (r). O adversário não pode usar as entradas dos

participantes honestos, as quais ele naturalmente não tem acesso, para definir as do seu conluio. Tampouco pode aguardar uma resposta da TPC para então substituir as entradas dos participantes desonestos.

No caso em que o adversário decide não abortar, correspondente à Equação (4.4), a saída recebida pelos participantes honestos é calculada com base na entrada (possivelmente) alterada $\bar{y}$. A saída dos participantes do conluio é definida de maneira semelhante ao do caso anterior.

Vamos prosseguir agora com a definição da visão conjunta da execução real.

Definição 4.36 (Visão conjunta da execução real)**.** *Seja f uma m-funcionalidade, Π um protocolo que computa f e $\bar{x} = (x_1, \ldots, x_m)$ uma sequência de entrada para f (e Π). Definimos o adversário de Π como o par (C, A), em que $C \subseteq \{1, \cdots, m\}$ é o conluio de participantes desonestos e A é um algoritmo probabilístico que representa a estratégia do adversário. Definimos $H \stackrel{def}{=} \{1, \cdots, m\} \setminus C$ como o conjunto de participantes honestos e $SAÍDA_H^{\Pi}$ como as saídas obtidas após a execução de Π pelos participantes honestos. Definimos $SAÍDA_C^{\Pi}$ de maneira análoga. Definimos a visão conjunta da execução real do protocolo Π como a seguir:*

$$REAL_{\Pi,C,A}(\bar{x}) \stackrel{def}{=} (SAÍDA_H^{\Pi}, SAÍDA_C^{\Pi}) \tag{4.5}$$

A visão conjunta real é uma tupla com as saídas dos participantes honestos e desonestos, assim como a visão conjunta ideal. Enfatizamos que o comportamento dos participantes desonestos é completamente determinado pela estratégia A do adversário. Isso inclui as entradas que esses participantes fornecem a Π bem como as mensagens que eles enviam e se devem ou não abortar.

Com a definição dos conceitos de visão conjunta real e ideal, podemos agora estabelecer a noção de segurança contra adversários maliciosos.

Definição 4.37 (Computação segura de uma funcionalidade por um protocolo - adversário malicioso e canal de *broadcast*)**.** *Seja f uma m-funcionalidade e Π um protocolo que computa f. Definimos que Π computa f de forma segura se, para todo algoritmo probabilístico de tempo polinomial A, representando a estratégia do adversário real, existe um algoritmo probabilístico de tempo polinomial S, representando a estratégia do adversário ideal, de modo que, para todo conluio $C \subseteq \{1, \cdots, m\}$:*

$$\{IDEAL_{f,C,S}(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^m} \stackrel{c}{\equiv} \{REAL_{\Pi,C,A}(\bar{x})\}_{\bar{x} \in (\{0,1\}^*)^m} \tag{4.6}$$

É importante ressaltar uma diferença fundamental entre as definições de segurança contra adversários semi-honestos (Definição 4.33) e maliciosos (Definição 4.37). Na primeira delas, fizemos o uso de uma visão *local* da execução, particular de cada participante e que nos permitiu capturar o que é aprendido pelo adversário. Já na segunda, quando supomos um adversário malicioso, fizemos o uso de uma visão *conjunta*, a qual contém também as saídas dos participantes honestos. Portanto a visão conjunta não representa *apenas* o que um adversário

aprende. Ela também estabelece uma relação entre as saídas dos participantes honestos e desonestos, a qual deve ser reproduzida (pelos adversários) tanto na execução real quanto na ideal, onde não há interação entre os participantes.

É interessante notar que, segundo a Definição 4.37, manipular as mensagens trocadas durante a execução de um protocolo seguro não confere vantagem ao adversário real. Isso pode ser concluído a partir da garantia de que se existe uma estratégia que usa essas mensagens para atacar o protocolo, existe uma estratégia correspondente que atinge os mesmos resultados no modelo ideal, onde não há troca de mensagens entre os participantes.

No entanto, a Definição 4.37 ainda permite alguns comportamentos indesejáveis por parte do adversário. Um deles, como já mencionado, é a capacidade de abortar prematuramente, o que significa que os participantes honestos recebem $\perp$ ao invés da saída apropriada. Outro comportamento possível é o de alterar as saídas dos participantes honestos através da substituição das entradas dos desonestos. Isso, no entanto, não significa que o adversário consiga aprender (ou predizer) essas saídas. Além disso, o adversário pode instruir aos participantes do conluio a não iniciarem o protocolo[3].

4.2 Adversário Semi-Honesto

Até agora nos concentramos em discutir o que significa computar uma funcionalidade através de um protocolo de maneira segura. Para isso, consideramos adversários semi-honestos e maliciosos. Nesta seção, aprofundaremos essa discussão tratando, especificamente, da computação privada de uma funcionalidade arbitrária, i.e., da computação segura contra adversários semi-honestos.

Nosso objetivo é exibir um procedimento para computar privadamente *qualquer* funcionalidade. No entanto, vamos considerar inicialmente apenas funcionalidades *determinísticas*. Com essa restrição em mente, vamos apresentar, na Subseção 4.2.1, como as funcionalidades serão representadas. Em seguida, na Subseção 4.2.2 vamos definir importantes conceitos sobre composição e redução, os quais nos ajudarão a apresentar os demais resultados e construções de maneira modular. Definiremos, na Subseção 4.2.3, a importante funcionalidade para *transferência oblívia*, a qual servirá como base para muitos resultados ao longo de todo o capítulo. Na Subseção 4.2.4 apresentaremos o procedimento para computação privada de uma funcionalidade determinística. Em seguida, na Subseção 4.2.5, mostraremos como estender esse resultado também para funcionalidades probabilísticas. Finalmente, na Subseção 4.2.6 estabeleceremos nosso resultado principal de maneira a facilitar a sua utilização no decorrer do capítulo.

[3] A estratégia de não iniciar o protocolo pode ser encarada como um caso especial de substituição de entradas, no qual o adversário substitui as entradas dos membros de seu conluio por $\perp$.

4.2.1 Representação das Funcionalidades

Começaremos definindo a seguir como as funcionalidades serão representadas ao longo desta seção.

Definição 4.38 (Representação de funcionalidades)**.** *Seja f uma m-funcionalidade determinística e n o tamanho de cada uma das cadeias de bits de entrada de f. Ou seja, na notação que utilizamos até aqui, se $\bar{x} \stackrel{def}{=} (x_1, \cdots, x_m)$ é uma entrada para f, então $|x_i| = n$ para todo $i \in \{1, \cdots, m\}$. Definimos então que f é representada por um circuito lógico acíclico CL tal que:*

1. *É formado apenas por portas AND e XOR, cada uma possuindo exatamente duas conexões de entradas e uma saída.*
2. *Existem n conexões de entrada para cada um dos m participantes e, portanto, $n \times m$ no total.*
3. *Existem n conexões de saída para cada um dos m participantes, totalizando $n \times m$. (Assumindo, por simplicidade, que cada participante recebe n bits de saída).*
4. *Todas as conexões são numeradas de forma que:*
   - *As conexões de entrada são numeradas de 1 a $n \times m$. O i-ésimo participante fornece o j-ésimo bit de sua entrada através da conexão $(i-1) \times n + j$.*
   - *As conexões de saída são numeradas de $L - n \times m + 1$ a L, em que L é a quantidade total de conexões em CL. O i-ésimo participante recebe o j-ésimo bit de sua saída através da conexão $L - (m + 1 - i) \times n + j$.*
   - *Para toda porta, inclusive as intermediárias, as conexões de suas entradas possuem numeração inferior às das conexões de sua saída.*

Ao longo desta seção toda funcionalidade será tratada como um circuito lógico, conforme a Definição 4.38. A restrição que essa definição impõe sobre os tipos de portas que podem ser utilizadas nos circuitos é bastante conveniente, pois nos permite encarar cada porta como uma operação sobre o *corpo finito com dois elementos* (F_2). Mais especificamente, a porta *AND* corresponde à operação de multiplicação ($\cdot$) e a porta *XOR* à operação de soma ($+$). Devido a isso, ao longo dessa seção, nos referimos a *CL* tanto como um circuito lógico, quanto como um circuito de operações aritméticas sobre o F_2.

Durante a computação de um circuito, é necessário garantir que cada porta lógica só seja avaliada quando suas entradas estiverem disponíveis, isto é, quando as portas que fornecem seus valores de entrada já tiverem sido avaliadas. Para facilitar esse processo, a Definição 4.38 introduz um esquema de numeração que nos permite computar um circuito partindo de suas conexões de menor número para as de maior.

Note que a Definição 4.38 assume que, nas funcionalidade tratadas, cada participante fornece *n bits* como entrada e recebe *n bits* como saída. Já discutimos anteriormente a necessidade

de que todas as entradas tenham o mesmo tamanho. Mas não precisamos assumir o mesmo em relação às saídas. Fazemos isso apenas para simplificar a numeração das conexões e facilitar a apresentação no restante da seção.

4.2.2 Composição e Redução de Funcionalidades

Antes de apresentarmos os principais resultados desta seção, vamos definir alguns conceitos importantes relacionados à composição e redução de funcionalidades. Eles nos permitem proceder de maneira modular na apresentação das construções desta seção. Começamos com a definição de oráculo.

Definição 4.39 (Oráculo para uma funcionalidade)**.** *Seja f uma m-funcionalidade. Definimos que um oráculo para f é uma entidade que se comporta como uma TPC para computação de f. Ou seja, o oráculo para f recebe entradas e emite respostas condizentes com f. Em geral, vamos denotar o oráculo para uma funcionalidade pelo mesmo símbolo com o qual nos referimos à funcionalidade em si, com o intuito de reforçar que o oráculo se comporta como uma execução ideal para a tal funcionalidade.*

Definição 4.40 (Redução privada)**.** *Sejam f e g duas funcionalidades. Definimos que f se reduz privadamente a g se existe um protocolo $\Pi_{f\to g}$ para computar privadamente f com acesso a um oráculo para g. Também definimos que $\Pi_{f\to g}$ reduz privadamente f a g.*

Pela Definição 4.40, sempre que quisermos demonstrar que uma funcionalidade f se reduz privadamente a g, devemos exibir um protocolo que compute *privadamente* f com acesso oracular à g.

A Definição 4.40 também nos ajuda a obter uma definição natural de composição. Ou seja, se existe um protocolo $\Pi_{f\to g}$ que reduz privadamente f a g e um Π_g que computa privadamente g, então podemos construir Π_f para computar f privadamente *sem auxílio de um oráculo* para g. Registramos essa noção a seguir.

Definição 4.41 (Composição de funcionalidades)**.** *Sejam f, g duas funcionalidades, $\Pi_{f\to g}$ um protocolo que reduz privadamente f a g e Π_g um protocolo que computa g privadamente. Definimos Π_f o protocolo resultante da substituição de cada chamada oracular em $\Pi_{f\to g}$ pela execução de uma instância de Π_g.*

A operação de composição, conforme definida acima, nos permite apresentar separadamente funcionalidades e protocolos para computá-las. Sempre que precisarmos, utilizaremos um oráculo como subrotina em uma construção mais complexa, ao invés de utilizar uma implementação concreta (protocolo).

A seguir registramos a correção da operação de composição.

Teorema 4.42 (Teorema da composição)**.** *Se f e g são duas funcionalidades, f se reduz a g e Π_g é um protocolo que computa g privadamente, então o protocolo Π_f construído como na Definição 4.41 computa f privadamente.*

Esboço de Prova. Se f se reduz a g, então existe um protocolo $\Pi_{f\to g}$ que computa f privadamente com acesso oracular a g. Intuitivamente, é fácil se convencer da *correção* de Π_f porque cada chamada feita a Π_g em Π_f é conduzida de maneira indistinguível da chamada feita a g em $\Pi_{f\to g}$. A segurança de Π_f também se baseia nesse fato.

Para mostrar formalmente a segurança de Π_f devemos exibir um simulador S_f da visão de cada participante, conforme a Definição 4.33. Podemos construir S_f utilizando os simuladores $S_{f\to g}$ e S_g que garantidamente existem porque, por hipótese, $\Pi_{f\to g}$ e Π_g são seguros. A ideia geral é executar o simulador $S_{f\to g}$ para obter uma "visão geral" e substituir cada interação com g por uma visão gerada pelo simulador S_g. □

4.2.3 Transferência Oblívia

O objetivo dessa subseção é apresentar a importante funcionalidade f^k_{TO} para transferência oblívia.

A funcionalidade f^k_{TO}, para algum k predeterminado, é computada por dois participantes. O participante P_1, que chamamos de *servidor*, possui uma sequência $(\sigma_1,\cdots,\sigma_k)$ de k *bits* e o P_2, que chamamos de *cliente*, quer obter o elemento σ_i, para algum $1\leq i\leq k$. O servidor então envia σ_i *sem conhecer* o valor i. O cliente, por sua vez, não aprende nada sobre qualquer outro elemento σ_j, com $i\neq j$.

Essa funcionalidade terá um papel muito importante na viabilidade e na segurança das construções que serão apresentadas nessa seção. De fato, a segurança do nosso resultado final, isto é, do protocolo para computação privada de uma funcionalidade qualquer, se baseia na segurança da computação de f^4_{TO}. A funcionalidade de f^k_{TO} é formalmente definida a seguir.

Definição 4.43 (Funcionalidade para transferência oblívia)**.** *Seja k um natural e $(\sigma_1,\cdots,\sigma_k)\in\{0,1\}^k$ uma sequência de k bits. Definimos f^k_{TO} como a 2-funcionalidade a seguir:*

$$f^k_{TO}((\sigma_1,\cdots,\sigma_k),i)\stackrel{def}{=}(\lambda,\sigma_i),\ \text{com}\ 1\leq i\leq k$$

em que λ representa a cadeia vazia. Além disso, chamaremos P_1 de servidor e P_2 de cliente.

Existem outras possíveis definições para transferência oblívia, como expõe (GOLDREICH, 2004, p.643). Seguimos, no entanto, a apresentação feita em (GOLDREICH, 2004, p.640), de onde derivamos a Definição 4.43 que é simples e suficiente para os nossos propósitos.

O protocolo para computar privadamente f^k_{TO} assume a existência de uma família de permutações arapuca aprimoradas e utiliza suas garantias de intratabilidade para alcançar segurança. Esse protocolo é descrito a seguir.

Definição 4.44 (Protocolo Π^k_{TO} para computar f^k_{TO} - adversário semi-honesto)**.** *Seja $\{f_\alpha : D_\alpha\to D_\alpha\}$ uma coleção de permutações arapuca aprimoradas, com seus algoritmos (I,S,F,F^{-1}), b um predicado núcleo-duro para essa coleção e k um natural predeterminado. Seja ainda 1^n um parâmetro de segurança em posse dos participantes P_1 e P_2. Definimos o protocolo Π^k_{TO} para computar a funcionalidade f^k_{TO} como a seguir:*

E1. (Entradas:) O servidor (P_1) fornece $(\sigma_1, \cdots, \sigma_k) \in \{0,1\}^k$. O cliente ($P_2$) fornece i, com $1 \leq i \leq k$.

E2. (Servidor inicializa permutação:) O servidor executa o algoritmo I de geração de arapuca sobre o parâmetro de segurança 1^n e um valor uniformemente escolhido r, de modo a obter $(\alpha, t) = I(1^n, r)$, em que α é o índice da permutação a ser utilizada e t é uma arapuca. O servidor envia α ao cliente.

E3. (Cliente cria sequência:) O cliente utiliza o algoritmo S de amostragem de domínio para escolher uniformemente uma sequência $(x_1, \cdots, x_k)$, com $x_j \in D_\alpha$ para todo $j \in \{1, \cdots, k\}$. Isto é, o cliente sorteia r_j de maneira uniforme e calcula $x_j = S(\alpha, r_j)$, para todo $j \in \{1, \cdots, k\}$.

O cliente então monta a sequência $\bar{y} \stackrel{def}{=} (y_1, \cdots, y_k)$ de modo que $y_i = F(\alpha, x_i)$ e $y_j = x_j$ para $j \neq i$. O cliente envia $\bar{y}$ para o servidor.

E4. (Servidor 'esconde' σ_i:) Ao receber $\bar{y}$ o servidor calcula $z_j = F^{-1}(\alpha, y_j)$ e $c_j = \sigma_j \oplus b(z_j)$, para todo $j \in \{1, \cdots, k\}$. O servidor então envia $(c_1, \cdots, c_k)$ ao cliente.

E5. (Saídas:) O cliente define $c_i \oplus b(x_i)$ como sua saída.

Com '$\oplus$' representando a operação de ou-exclusivo (XOR).

Teorema 4.45 (Segurança do protocolo Π^k_{TO})**.** *Sob a hipótese de existência de uma família de permutações arapuca aprimoradas, o protocolo Π^k_{TO} da Definição 4.44 computa privadamente a funcionalidade f^k_{TO} para todo k fixado a priori.*

Esboço de Prova. O protocolo Π^k_{TO} de fato computa a funcionalidade f^k_{TO}, pois $c_i \oplus b(x_i)$, a saída recebida pelo cliente, é igual a σ_i.

Em relação à privacidade, note que o servidor não aprende nada sobre o valor i do cliente. Tudo o que ele vê é a sequência $\bar{y}$ de k valores distribuídos uniformemente. Da mesma forma, o cliente não aprende nada sobre os demais σ_j, com $j \neq i$, sob a hipótese de que calcular $b(x_j)$ tendo acesso apenas à $F(\alpha, x_j)$ é intratável.

Note que, como nas demais provas de segurança dessa seção, o adversário (semi-honesto) não pode se desviar da especificação do protocolo. Especificamente, se o cliente fosse desonesto e montasse a sequência $\bar{y}$ como $(F(\alpha, x_1), \cdots, F(\alpha, x_k))$, ele teria acesso a *todos* os valores σ_j. □

Com o Teorema 4.45 nós finalizamos esta subseção, estabelecendo um importante resultado para o decorrer da nossa discussão, como já mencionado. Mais especificamente, utilizaremos um oráculo para a funcionalidade f^4_{TO} como base das construções do restante desta seção.

4.2.4 Computação Privada de um Circuito

O objetivo desta subseção é o de apresentar um protocolo que permita a computação privada de uma funcionalidade determinística f qualquer. Isso equivale, de acordo com a Definição 4.38, a conduzir uma computação privada de um circuito aritmético sobre o F_2 representando f.

Lembramos que uma computação é privada quando ela é segura na presença de um adversário semi-honesto. Esse adversário, por sua vez, é sempre obrigado a seguir à risca o protocolo de que está participando. Sua única habilidade é a de bisbilhotar as interações que lhe forem acessíveis. Dessa forma, assegurar que um protocolo é privado equivale, de maneira geral, a garantir que as informações obtidas licitamente por um participante qualquer durante a execução desse protocolo não revelem nada *que não deveriam*[4] sobre as entradas ou saídas dos demais.

Durante a computação de um circuito lógico, no entanto, as entradas de todos os participantes precisam ser propagadas através de portas lógicas para permitir a computação do resultado final. Para que esse processo não quebre a privacidade da computação, cada *bit* propagado pelo circuito é *dividido* entre os participantes. Esse divisão se dá de modo que nenhum participante possuindo apenas um *pedaço* do *bit* consiga reconstruir o valor original[5]. A divisão de um *bit* é definida a seguir.

Definição 4.46 (Divisão e pedaço de um *bit*)**.** *Seja $b \in \{0,1\}$ um bit. A divisão de b em m partes é uma sequência $\hat{B}$ uniformemente escolhida do conjunto $\{(b_1,\cdots,b_m) \in \{0,1\}^m \mid b = \sum_{i=1}^{m} b_i\}$, considerando-se a soma no F_2. Cada $b_i \in \hat{B}$ é chamado de um pedaço de b. No contexto de uma m-funcionalidade, dividir b entre m participantes significa gerar a divisão $\hat{B}$ e enviar $b_i \in \hat{B}$ para o participante P_i, para todo $i \in \{1,\cdots,m\}$.*

Para computar um circuito de forma privada, vamos apresentar um protocolo que avalia uma porta lógica por vez. Esse processo é conduzido de modo a manter a invariante de que todo *bit* propagado através de uma conexão esteja dividido entre os m participantes da computação. Lembramos que uma conexão pode ser uma das entradas do circuito, uma de suas saídas, ou ainda ligar a saída de uma porta lógica a uma entrada de outra.

Essa divisão dos *bits* de cada conexão significa, em particular, que nenhum participante, em qualquer momento da computação, conhece completamente os dois *bits* de entrada de qualquer porta lógica do circuito. Apenas um pedaço de cada um deles[6].

O principal desafio dessa estratégia de computação privada é, portanto, avaliar privadamente cada porta lógica. Ou seja, proceder de modo que cada participante calcule seu pedaço do

[4] Já discutimos a privacidade de uma computação (veja a funcionalidade t da Subseção 4.1.1, por exemplo). Um protocolo privado não deve acrescentar vulnerabilidades à funcionalidade que ele computa mas, de maneira geral, ele "preserva" as que já existem.

[5] De fato, se um *bit* b é dividido em m partes, nenhum *subconjunto* de tamanho inferior a m desses pedaços deve ser capaz de revelar qualquer informação sobre b.

[6] Cada participante conhece, naturalmente, os seus próprios *bits* de entrada, mesmo quando eles são divididos.

bit de saída de uma porta lógica sem vazar qualquer informação sobre os seus próprios pedaços (de entrada). Sem tampouco aprender algo sobre os pedaços dos *bits* de entrada ou saída dos demais participantes. A avaliação privada de uma porta lógica é definida a seguir.

Definição 4.47 (Avaliação privada de uma porta lógica)**.** *Sejam $a,b \in \{0,1\}$ dois bits e $\hat{A},\hat{B}$ suas respectivas divisões em m partes. Seja $\{P_1,\cdots,P_m\}$ o conjunto de participantes, de modo que P_i possui os elementos $a_i \in \hat{A}$ e $b_i \in \hat{B}$. Definimos a avaliação privada de uma porta lógica $PL \in \{XOR, AND\}$ como a m-funcionalidade $Aval_{PL}$ tal que:*

$$\begin{gathered} Aval_{PL}((a_1,b_1),\cdots,(a_m,b_m)) \stackrel{def}{=} (c_1,\cdots,c_m), \\ \text{com a restrição de que } \left(\sum_{i=1}^{m} c_i\right) = \left(\sum_{i=1}^{m} a_i\right) \ OP \ \left(\sum_{i=1}^{m} b_i\right) \end{gathered} \tag{4.7}$$

em que OP é a operação definida pela porta PL. Ou seja, OP é soma $(+)$ caso $PL = XOR$, ou OP é multiplicação $(\cdot)$ caso $PL = AND$. Em outras palavras, $Aval_{PL}$ divide o bit de saída c da porta lógica PL entre os m participantes.

A avaliação privada de uma porta lógica *XOR* é a mais simples de ser conduzida. A funcionalidade $Eval_{xor}$ pode ser computada por um protocolo que não exige interação entre os participantes. Apresentamos esse protocolo na Definição 4.48 a seguir.

Definição 4.48 (Protocolo Π_{xor} para computação privada de $Eval_{xor}$)**.** *Sejam $a,b \in \{0,1\}$ dois bits e $\hat{A},\hat{B}$ suas respectivas divisões em m partes. Seja ainda $\{P_1,\cdots,P_m\}$ o conjunto de participantes, de modo que cada P_i possui os elementos $a_i \in \hat{A}$ e $b_i \in \hat{B}$. Definimos o protocolo Π_{xor} para computar a funcionalidade $Eval_{xor}$ como segue:*

E1. (Entradas:) Cada participante P_i fornece $a_i \in \hat{A}$ e $b_i \in \hat{B}$.

E2. (Cálculo do pedaço do bit $(a+b)$:) Cada participante P_i calcula $c_i \stackrel{def}{=} a_i + b_i$.

E3. (Saídas:) Cada participante P_i define c_i como saída.

Lema 4.49 (Segurança e correção de Π_{xor})**.** *O protocolo Π_{xor} da Definição 4.48 computa privadamente a funcionalidade $Eval_{xor}$.*

Esboço de Prova. A privacidade de Π_{xor} segue diretamente do fato de que ele é calculado de forma local, sem qualquer interação externa[7]. Falta apenas verificar que seu resultado é correto, isto é, que $(\sum_{i=1}^{m} c_i) = (a+b)$. Isso se verifica porque:

[7]Note que é trivial demonstrar a segurança de Π_{xor} através da definição Definição 4.33. O algoritmo S a ser definido precisa simular, para todo $i \in C$, a visão $VISÃO_i^{\Pi} = ((a_i,b_i),\lambda,\lambda)$, uma vez que o protocolo é determinístico e nenhum participante recebe mensagens.

$$\sum_{i=1}^{m} c_i = \sum_{i=1}^{m} (a_i + b_i) = (\sum_{i=1}^{m} a_i) + (\sum_{i=1}^{m} b_i) = (a+b).$$

sendo a última igualdade válida porque a_i e b_i são, por hipótese, pedaços de a e b, respectivamente. □

No caso da avaliação privada de uma porta *AND*, a interação entre os participantes é necessária. Para apresentar um protocolo que compute privadamente a funcionalidade $Eval_{and}$, vamos proceder em duas etapas. Inicialmente apresentamos um protocolo para computar privadamente $Eval_{and}$ no caso particular de dois participantes, isto é, $m = 2$. A funcionalidade f_{TO}^4 é utilizada nessa computação. Em seguida, estenderemos essa construção para o caso geral. Iniciamos essa estratégia com a Definição 4.50 e o Lema 4.51 a seguir.

Definição 4.50 (Protocolo Π'_{and} para computação privada de $Eval_{and}$ no caso particular de dois participantes)**.** *Seja $Eval'_{and}$ a funcionalidade $Eval_{and}$, de acordo com a Definição 4.47, para o caso particular em que $m = 2$. Isto é, $Eval'_{and}((a_1,b_1),(a_2,b_2)) \stackrel{def}{=} (c_1,c_2)$, de modo que $(a_1+a_2)\cdot(b_1+b_2) = (c_1+c_2)$. Seja ainda f_{TO}^4 um oráculo para a funcionalidade de transferência oblívia (Definição 4.43, com k fixado em 4). Definimos a seguir o protocolo Π'_{and} para computar a funcionalidade $Eval'_{and}$:*

E1. (Entradas:) P_1 fornece (a_1,b_1) e P_2 fornece (a_2,b_2), em que a_1,b_1,a_2,b_2 são bits.

E2. (Etapa 1:) P_1 escolhe $c_1 \in \{0,1\}$ uniformemente.

E3. (Redução a f_{TO}^4:) Seja a funcionalidade $g_{a,b}$ tal que $g_{a,b}(x,y,z) \stackrel{def}{=} z + (x+a)\cdot(y+b)$. O participante P_1 invoca o oráculo f_{TO}^4 fazendo o papel de servidor e fornecendo a seguinte sequência como entrada:

$$\bar{g} \stackrel{def}{=} (g_{0,0}(a_1,b_1,c_1), g_{0,1}(a_1,b_1,c_1), g_{1,0}(a_1,b_1,c_1), g_{1,1}(a_1,b_1,c_1)).$$

O participante P_2 precisa do valor $g_{a_2,b_2}(a_1,b_1,c_1)$, que corresponde ao elemento de $\bar{g}$ na posição $(1+2a_2+b_2) \in \{1,2,3,4\}$. Ele faz o papel de cliente e fornece $(1+2a_2+b_2)$ como sua entrada para o oráculo f_{TO}^4. O oráculo f_{TO}^4 então faz a transferência de $g_{a_2,b_2}(a_1,b_1,c_1)$ para P_2.

E4. (Saídas:) P_1 define c_1 como saída. P_2 define $c_2 \stackrel{def}{=} g_{a_2,b_2}(a_1,b_1,c_1)$ como saída.

Lema 4.51 (Redução privada de $Eval'_{and}$ a f_{TO}^4)**.** *O protocolo da Definição 4.50 reduz privadamente a funcionalidade $Eval'_{and}$ a f_{TO}^4. Em outras palavras, Π'_{and} computa privadamente a funcionalidade $Eval'_{and}$ com auxílio de um oráculo para f_{TO}^4.*

Esboço de Prova. A Tabela 4.1 nos ajuda a conferir a correção do protocolo Π'_{and}. Ela possui uma linha para cada possível entrada do participante P_2, mostrada na primeira coluna. A segunda coluna contém a saída de Π'_{and} recebida por P_2 (a saída de P_1 é sempre c_1). A terceira coluna contém a soma das saídas do protocolo, por isso chamadas de *reais*. Por fim, a última coluna

Tabela 4.1: Valores relacionados à computação de Π'_{and}

(a_2, b_2) *(entrada de P_2)*	c_2 *(saída de P_2)*	$c_1 + c_2$ *(soma das saídas **reais**)*	$(a_1+a_2)\cdot(b_1+b_2)$ *(valor **ideal** da soma das saídas)*
(0,0)	$c_1 + a_1 \cdot b_1$	$a_1 \cdot b_1$	$(a_1+0)\cdot(b_1+0)$
(0,1)	$c_1 + a_1 \cdot (b_1+1)$	$a_1 \cdot (b_1+1)$	$(a_1+0)\cdot(b_1+1)$
(1,0)	$c_1 + (a_1+1) \cdot b_1$	$(a_1+1) \cdot b_1$	$(a_1+1)\cdot(b_1+0)$
(1,1)	$c_1 + (a_1+1) \cdot (b_1+1)$	$(a_1+1) \cdot (b_1+1)$	$(a_1+1)\cdot(b_1+1)$

mostra qual é o valor correto, ou *ideal*, da soma das saídas: o produto dos *bits* $a \stackrel{def}{=} (a_1 + a_2)$ e $b \stackrel{def}{=} (b_1 + b_2)$. A correção do protocolo vem diretamente do fato que a última e penúltima colunas possuem valores iguais.

Também exige-se que as saídas c_1 e c_2 tenham distribuição uniforme para que sejam considerados pedaços de um *bit*. Isso vale para c_1 por construção. Também vale para c_2 porque ele é da forma $p+q$, com $p \stackrel{def}{=} c_1$ uniformemente distribuído e p e q independentes. Se p tem distribuição uniforme, então $Pr[p=1] = Pr[p=0] = \frac{1}{2}$. Daí temos que $Pr[(p+q)=0] = (Pr[p=0]\cdot Pr[q=0]) + (Pr[p=1]\cdot Pr[q=1]) = \frac{1}{2}\cdot(Pr[q=0] + Pr[q=1]) = \frac{1}{2}$. Consequentemente, $Pr[(p+q)=1] = 1 - Pr[(p+q)=0] = \frac{1}{2}$. Portanto concluímos que o valor $(p+q)$ é também uniformemente distribuído, sem fazer qualquer suposição sobre a distribuição de q. Lembramos que a soma $(+)$ é feita no F_2 e , portanto equivale ao *XOR* $(\oplus)$.

□

Continuando nossa estratégia para a avaliação privada de uma porta *AND*, vamos reduzir o caso geral, com m participantes, para o caso específico de dois participantes que foi tratado na Definição 4.50. Para isso, vamos manipular algebricamente a expressão que define o *AND* (multiplicação no F_2) de dois *bits* a e b com base em seus pedaços a_i e b_i, para $1 \leq i \leq m$:

$$\left(\sum_{i=1}^{m} a_i\right) \cdot \left(\sum_{i=1}^{m} b_i\right) \tag{4.8}$$

$$= \left(\sum_{i=1}^{m} a_i \cdot b_i\right) + \left(\sum_{i=1}^{m} (b_i \cdot \sum_{j=1}^{i-1} a_j)\right) + \left(\sum_{i=1}^{m} (b_i \cdot \sum_{j=i+1}^{m} a_j)\right) \tag{4.9}$$

$$= \left(\sum_{i=1}^{m} a_i \cdot b_i\right) + \left(\sum_{1 \leq i < j \leq m} (a_i \cdot b_j + a_j \cdot b_i)\right) \tag{4.10}$$

$$= (1-(m-1)) \cdot \left(\sum_{i=1}^{m} a_i \cdot b_i\right) + \left(\sum_{1 \leq i < j \leq m} (a_i + a_j) \cdot (b_i + b_j)\right) \tag{4.11}$$

$$= m \cdot \left(\sum_{i=1}^{m} a_i \cdot b_i\right) + \left(\sum_{1 \leq i < j \leq m} (a_i + a_j) \cdot (b_i + b_j)\right) \tag{4.12}$$

Ressaltamos que as equações acima usam a aritmética *modulo 2*, pois seus operadores

são definidos no F_2. Em particular, isso significa que $-1 = +1$ e $2 \cdot x = 0$, por exemplo.

A Equação (4.8) define a operação de *AND* de forma convencional, a partir dos *bits* a e b que estão escritos como o somatório de seus pedaços. A Equação (4.9) é obtida através do uso da distribuição da multiplicação sobre a soma e da organização do resultado em três parcelas principais. Já a Equação (4.10) é fruto de uma reorganização dos índices dos somatórios. Na Equação (4.11) subtraímos o valor $(m-1) \cdot (\sum_{i=1}^{m} a_i \cdot b_i)$ da parcela mais à esquerda e o adicionamos na parcela mais à direita, não alterando, portanto, o valor final da expressão. Na Equação (4.12) usamos o fato que $(2 = 0 \; mod \; 2)$ e $(m = -m \; mod \; 2)$, para concluir que $(1-(m-1)) = 2-m = m$. Daí concluímos que a Equação (4.12) é equivalente às demais e, em particular, à Equação (4.8) servindo, portanto, para definir a operação de *AND* entre os *bits* a e b.

Do ponto de vista de um participante P_i, segundo a Equação (4.12), a divisão do *bit* da saída de uma porta *AND* pode ser calculado em duas parcelas. A primeira delas, $(m \cdot a_i \cdot b_i)$, pode ser calculada localmente porque envolve apenas valores conhecidos por P_i. Já a segunda, $\big((a_i + a_j) \cdot (b_i + b_j)\big)$, envolve valores conhecidos por P_i e P_j e pode ser calculada com o auxílio de um oráculo para $Eval'_{and}$ (Definição 4.50). Essas observações são centrais para o protocolo apresentado na Definição 4.52 a seguir.

Definição 4.52 (Protocolo Π_{and} para computação privada de $Eval_{and}$ no caso geral de m participantes)**.** *Sejam $a, b \in \{0,1\}$ dois bits e $\hat{A}, \hat{B}$ suas respectivas divisões em m partes. Seja $\{P_1, \cdots, P_m\}$ o conjunto de participantes, de modo que P_i possui $a_i \in \hat{A}$ e $b_i \in \hat{B}$. Por fim, seja $Eval'_{and}$ um oráculo para a funcionalidade da Definição 4.50. Definimos o protocolo Π_{and} para computar a funcionalidade $Eval_{and}$ como a seguir:*

E1. (Entradas:) Cada participante P_i fornece $(a_i, b_i) \in \{0,1\}^2$.

E2. (Redução ao caso de dois participantes:) Para cada par de participantes (P_i, P_j), com $1 \leq i < j \leq m$, P_i invoca o oráculo $Eval'_{and}$ fornecendo (a_i, b_i) como entrada. P_j fornece (a_j, b_j). Seja $c_i^{\{i,j\}}$ a saída do oráculo recebida por P_i e $c_j^{\{i,j\}}$ a saída recebida por P_j. Note que, pela definição de $Eval'_{and}$, $c_i^{\{i,j\}} + c_j^{\{i,j\}} = (a_i + a_j) \cdot (b_i + b_j)$.

E3. (Saídas:) Cada participante P_i define $c_i \stackrel{def}{=} m \cdot a_i \cdot b_i + \sum_{j \neq i} c_i^{\{i,j\}}$ como saída.

Lema 4.53 (Redução de $Eval_{and}$)**.** *O protocolo Π_{and} reduz privadamente $Eval_{and}$ (caso geral) a $Eval'_{and}$ (caso de dois participantes)*

Esboço de Prova. Primeiramente, ressaltamos que, de fato, Π_{and} computa corretamente $Eval_{and}$. A soma de todas as saídas de Π_{and} ($\sum_{i=1}^{m} c_i$) corresponde exatamente à Equação (4.12) que, por sua vez equivale à formulação tradicional da operação *AND* usada na Definição 4.47. Além disso, utilizamos o mesmo argumento da prova do Lema 4.51 para concluir que cada c_i da saída de Π_{and} tem distribuição uniforme.

A privacidade de Π_{and}, por sua vez, se sustenta na segurança do oráculo $Eval_{and}$. □

Até agora temos discutido como uma porta lógica de um circuito representando uma funcionalidade (determinística) pode ser avaliada de maneira privada. Na sequência, aplicaremos os resultados conseguidos até aqui, em especial a redução da Definição 4.52, na construção de um protocolo que nos permita avaliar privadamente o *circuito* inteiro. A ideia geral é, inicialmente, fazer a divisão dos *bits* de entrada, de modo que cada participante tenha um pedaço de cada um deles. Em seguida, as conexões são percorridas na ordem de sua numeração, como estabelecido em Definição 4.38, e a porta lógica de 'origem' de cada conexão é avaliada privadamente. Dessa forma, queremos que, a cada passo desse protocolo, um novo *bit* seja calculado e seu valor dividido entre os participantes. Esse processo se repete até que todas as conexões tenham sido consideradas. No final, cada participante detém exatamente um pedaço de cada *bit* de saída. Ele então envia os pedaços dos *bits* que ele detém para os participantes adequados. Desse modo cada participante terá acesso apenas aos *bits* de sua saída.

Essa ideia geral é registrada na Definição 4.54 a seguir.

Definição 4.54 (Protocolo Π_{CL} para computação de um circuito)**.** *Seja CL um circuito representando uma m-funcionalidade f determinística e $\{P_1,\cdots,P_m\}$ um conjunto de participantes. Seja $\hat{x} \stackrel{def}{=} (x_1,\cdots,x_m) \in (\{0,1\}^n)^m$ uma sequência de entrada para f. Definimos que n também é o tamanho, em bits, da saída de cada participante. Seja L a quantidade total de conexões em CL, numeradas de acordo com a Definição 4.38, e $Eval_{and}$ um oráculo para a funcionalidade de avaliação privada de uma porta AND. Definimos o protocolo Π_{CL} para conduzir a computação de CL como a seguir:*

E1. (Entradas:) Cada participante P_i fornece a cadeia $x_i \in \{0,1\}^n$.

E2. (Divisão dos bits de entrada:) Para todo $i \in \{1,\cdots,m\}$ e todo $j \in \{1,\cdots,n\}$, o participante P_i divide o bit b_j de sua entrada (na conexão de número $(i-1)\cdot n + j$), em m partes e distribui os pedaços da seguinte maneira:

1. *Para todo $k \neq i$, P_i escolhe c_k uniformemente e envia esse valor para P_k. Esse é o pedaço do bit b_j correspondente a P_k.*
2. *P_i define o seu pedaço do bit b_j como $(b_j + \sum_{k\neq i} c_k)$.*

E3. (Computação do circuito:) Consideramos as conexões em ordem crescente de numeração, a partir da de número $n\cdot m+1$, isto é, pulando as conexões de entrada. Suponha que estejamos considerando uma determinada conexão α originada na saída de uma porta lógica PL. Sejam $a,b \in \{0,1\}$ os bits de entrada de PL, e $\hat{A},\hat{B}$ suas respectivas divisões em m partes. Naturalmente, cada participante P_i possui $a_i \in \hat{A}$ e $b_i \in \hat{B}$. Queremos calcular o pedaço do bit de saída de PL para cada participante P_i. Existem dois casos a serem considerados:

(i) PL = XOR e, portanto, o valor do bit propagado pela conexão α é $(a+b)$. Nesse caso, P_i define (a_i+b_i) como seu pedaço do valor $(a+b)$.

(ii) PL = AND e, portanto, o valor do bit propagado pela conexão α é $(a \cdot b)$. Nesse caso, os participantes invocam o oráculo $Eval_{and}$. Cada participante P_i, para todo $i \in \{1, \cdots, m\}$, fornece (a_i, b_i) como entrada e recebe c_i como saída. Então P_i define c_i como seu pedaço do valor $(a \cdot b)$.

E4. (Reconstrução dos bits de saída:) Cada participante envia os pedaços dos bits de saída que não lhe pertencem para os participantes correspondentes.

Isto é, para todo $i, j \in \{1, \cdots, m\}$, com $i \neq j$, P_i envia seu pedaço do bit da conexão $L - (m + 1 - i) \cdot n + j$ para o participante j.

E5. (Saídas:) Cada participante P_i faz a soma dos pedaços dos seus bits de saída, obtidos nas duas etapas anteriores.

Seja s_j a soma dos pedaços do j-ésimo bit de P_i. Então P_i define a sequência $(s_1, \cdots, s_n)$ como saída.

Lema 4.55 (Segurança do protocolo Π_{CL})**.** *O protocolo Π_{CL} reduz privadamente a computação de um circuito lógico à funcionalidade $Eval_{and}$, conforme apresentada na Definição 4.52.*

Esboço de Prova. O protocolo Π_{CL} trabalha com o conjunto C das conexões que propagam um *bit* cujo valor já é conhecido e está dividido entre os m participantes. O objetivo do procedimento de avaliação definido pelo protocolo é, portanto, aumentar C até que ele abranja todas as conexões do circuito. Como vimos, esse conjunto é aumentado uma conexão por vez. A correção de Π_{CL} se baseia na manutenção da invariante do conjunto toda vez que ele for aumentado (i.e., as conexões de C são tais que o *bit* propagado por elas é conhecido e está dividido entre os m participantes).

Para verificar que essa invariante é mantida por Π_{CL}, podemos aplicar um raciocínio indutivo. Como caso base, temos o conjunto inicial C_0, formado pelas conexões de entrada do circuito. Sua construção, na etapa E2, garante que ele é válido.

A etapa E3 trata do caso indutivo. Nela, uma conexão α que se origina na saída de alguma porta lógica CL é adicionada a C_i, originando C_{i+1}. A hipótese indutiva utilizada é a de que as conexões de entrada de CL estão em C_i. Isso é verdade porque consideramos as conexões por ordem crescente de suas numerações. O *bit* propagado pela conexão α é calculado e dividido ou pelo oráculo $Eval_{and}$ ou pelo procedimento equivalente a $Eval_{xor}$[8]. Por isso, a validade de $C_{i+1} \stackrel{def}{=} C_i \cup \{\alpha\}$, resultante da adição da conexão α ao conjunto C_i, se baseia na correção dessas duas funcionalidades.

A privacidade de Π_{CL} se baseia no fato de que os participantes conhecem apenas pedaços dos *bits* que se propagam pelo circuito. Isso é garantido pela segurança das funcionalidade $Eval_{and}$ e $Eval_{xor}$.

[8]Nós propositalmente não assumimos a existência de um oráculo para $Eval_{xor}$, seguindo a apresentação em (GOLDREICH, 2004, p.705). De fato, $Eval_{xor}$ nem sequer é tratada como uma funcionalidade em GOLDREICH (2004), devido à sua natureza trivial.

□

O Lema 4.55 estabelece um importante resultado *parcial*. Ele nos garante a existência de um procedimento para computar privadamente qualquer funcionalidade *determinística*. Na subseção a seguir vamos remover essa restrição.

4.2.5 Funcionalidades Probabilísticas

O protocolo Π_{CL} da Definição 4.54 e o resultado do Lema 4.55 nos garantem que qualquer funcionalidade *determinística* pode ser privadamente computada. Para estender esse resultado a qualquer funcionalidade nós vamos mostrar que toda funcionalidade probabilística pode ser (privadamente) reduzida a uma determinística e, portanto, ser computada privadamente utilizando Π_{CL} da Definição 4.54.

Essa redução é feita de uma maneira simples. Inicialmente reestruturamos uma funcionalidade probabilística de modo a tornar sua fonte de aleatoriedade explícita como entrada. Dessa forma, a funcionalidade passa a ser determinística, uma vez que sua saída depende exclusivamente de suas entradas. Para finalizar, exibimos um protocolo para formalizar a redução, de modo a demonstrar sua correção nos moldes do Teorema 4.42.

Definição 4.56 (Reescrita de uma funcionalidade probabilística)**.** *Seja f uma m-funcionalidade probabilística e $\bar{x} = (x_1, \cdots, x_m) \in (\{0,1\}^n)^m$ uma sequência de entrada. Tornamos explícita a fonte de aleatoriedade de f como uma cadeia de bits $r \in \{0,1\}^{poli(n)}$ e reescrevemos f como a seguir:*

$$f(x_1, \cdots, x_m) \stackrel{def}{=} f'(r, x_1, \cdots, x_m)$$

De acordo com Definição 4.56, f pode ser encarada como a funcionalidade probabilística que sorteia r aleatoriamente e computa f' deterministicamente.

Definição 4.57 (Versão determinística de uma funcionalidade probabilística)**.** *Seja f uma m-funcionalidade probabilística e $\bar{x} = (x_1, \cdots, x_m) \in (\{0,1\}^n)^m$ uma sequência de entrada. Definimos a versão determinística de f como a m-funcionalidade (determinística) f_{det} a seguir:*

$$f_{det}((x_1, r_1), \cdots, (x_m, r_m)) \stackrel{def}{=} f'(\bigoplus_{i=1}^{m} r_i, x_1, \cdots, x_m)$$

em que f' é como na Definição 4.56.

Cada elemento da sequência de entrada da versão determinística de uma m-funcionalidade f é estendido de modo a receber uma componente da aleatoriedade de f (ou seja, x_i é estendido para (x_i, r_i)). Essa maneira de definir f_{det} é conveniente porque permite calcular f' distribuidamente de maneira mais natural, com cada participante contribuindo para a cadeia r entregue a f'.

A Definição 4.57 nos indica como alcançar nosso objetivo final de reduzir uma funcionalidade probabilística a uma determinística. Vamos definir um protocolo para computar f

(probabilística) na presença de um oráculo para sua versão determinística f_{det}. Definimos esse protocolo a seguir.

Definição 4.58 (Redução de funcionalidade probabilística para determinística)**.** *Seja f uma m-funcionalidade probabilística e f_{det} um oráculo para sua versão determinística, de acordo com a Definição 4.57. Seja $\bar{x} \stackrel{def}{=} (x_1, \cdots, x_m) \in (\{0,1\}^n)^m$ uma sequência de entrada para f e $\{P_1, \cdots, P_m\}$ o conjunto de participantes. Definimos o protocolo $\Pi_{prob \to det}$ para reduzir f a f_{det} como a seguir:*

E1. (Entradas:) Cada participante P_i fornece a cadeia x_i.

E2. (Sorteio das componentes da aleatoriedade:) Cada participante P_i seleciona uniformemente $r_i \in \{0,1\}^{poli(n)}$.

E3. (Chamada oracular:) Cada participante P_i fornece a entrada (x_i, r_i) ao oráculo f_{det}. Seja s_i a saída do oráculo para P_i.

E4. (Saída:) Cada participante P_i define s_i como saída.

Lema 4.59 (Correção e segurança de $\Pi_{prob \to det}$)**.** *O protocolo $\Pi_{prob \to det}$ da Definição 4.58 reduz privadamente uma funcionalidade probabilística f à sua versão determinística f_{det}.*

Esboço de Prova. É simples verificar que $\Pi_{prob \to det}$ computa f corretamente. Ele consiste basicamente de uma etapa para escolha dos valores r_i e de outra para a realização da chamada oracular.

A privacidade dele se baseia na segurança de f_{det}. Ou seja, um simulador para f pode ser criado a partir do simulador para f_{det}. □

Para concluir esta seção, vamos fazer um breve resumo dos resultados que alcançamos. O Lema 4.51 nos permite avaliar privadamente uma porta *AND* no caso especial de um protocolo com 2 participantes. Ele funciona sob a hipótese de que f_{TO}^4 pode ser computada privadamente. Por sua vez, o Lema 4.53 reduz o caso geral da computação de uma porta *AND* ao caso especial de dois participantes. Com isso concluímos que uma porta *AND* pode ser avaliada privadamente por m participantes sob a hipótese de que f_{TO}^4 pode ser computada privadamente.

Na Definição 4.44, mostramos que a funcionalidade f_{TO}^k, em particular f_{TO}^4, pode ser computada privadamente sob a hipótese da existência de uma coleção de permutações arapuca aprimoradas.

Utilizando os resultados sobre a avaliação de portas lógicas, o Lema 4.55 estabelece que qualquer circuito lógico, representando uma funcionalidade determinística, pode ser privadamente avaliado. Por fim, com o Lema 4.59 podemos concluir que qualquer funcionalidade probabilística pode ser reduzida privadamente a uma determinística.

Desse modo, combinando todos esses resultados preliminares, formamos o resultado final e mais importante desta seção, registrado no Teorema 4.60.

Teorema 4.60 (Computação privada de uma funcionalidade qualquer)**.** *Sob a hipótese da existência de permutações arapuca aprimoradas, qualquer funcionalidade (probabilística ou determinística) pode ser privadamente computada.*

Esboço de Prova. Como mencionado, esse teorema é fruto dos resultados estabelecidos anteriormente, principalmente pelos Lemas 4.55 (computação privada de um circuito) e 4.59 (redução de uma funcionalidade probabilística para uma determinística).

Para "retirar" a dependência de oráculos desse resultado final devemos aplicar sucessivamente a operação de composição, permitida pelo Teorema 4.42. □

4.2.6 Protocolos Canônicos

O Teorema 4.60 já estabelece o resultado que perseguimos nesta subseção. No entanto, seguindo a apresentação em (GOLDREICH, 2004, p.707), e procurando simplificar futuros resultados, vamos definir, a seguir, um tipo especial de protocolo.

Definição 4.61 (Protocolos canônicos)**.** *Um protocolo Π que computa privadamente uma m-funcionalidade f é chamado canônico se sua execução é conduzida em duas etapas como a seguir:*

E1. (Divisão da saída:) Os participantes computam a m-funcionalidade g tal que $g(x_1,\cdots,x_m) \stackrel{def}{=} ((r_1^1,\cdots,r_m^1),\cdots,(r_1^m,\cdots,r_m^m))$ de modo que os r_j^is são escolhidos uniformemente dentre os valores que satisfazem $(\bigoplus_{i=1}^m r_1^i,\cdots,\bigoplus_{i=1}^m r_m^i) = f(x_1,\cdots,x_m)$.

E2. (União da saída:) Para $i = 2,\cdots,m$ e todo $j \in \{1,\cdots,m\} \setminus \{i\}$, o participante P_i, envia r_j^i a P_j. Em seguida, P_1 envia r_1^i a P_j, para $i = 2,\cdots,m$. Por fim, cada participante P_j calcula sua saída como $\bigoplus_{i=1}^m r_j^i$.

Pela Definição 4.61, em um protocolo canônico para computar f, os participantes inicialmente computam uma funcionalidade g que divide a saída de f entre todos. Desse modo, ao final da etapa E1, nenhum participante conhece sua saída, a qual só pode ser obtida no fim da etapa E2. Ou seja, um protocolo canônico é dividido em uma fase onde a computação é de fato conduzida e as saídas, divididas, e outra onde as saídas são reconstruídas. Note que o protocolo Π_{CL} é canônico. Isso nos permite estabelecer que:

Teorema 4.62. *Sob a hipótese da existência de permutações arapuca aprimoradas, qualquer funcionalidade pode ser privadamente computada por um protocolo canônico.*

Esboço de Prova. O teorema segue do Teorema 4.60 e do fato de que Π_{CL} é canônico. □

4.3 Adversário Malicioso

O resultado final da seção anterior estabelece que qualquer funcionalidade pode ser computada seguramente por um protocolo na presença de um adversário semi-honesto. Esse tipo de adversário é, por si só, de interesse porque pode modelar alguns comportamentos na prática. Especialmente quando o atacante de um protocolo não tem conhecimento, ou incentivo, para modificar o código que define a sua estratégia de execução. Nesses casos, ele pode apenas procurar por brechas na privacidade do protocolo. Esse comportamento também é conhecido como "honesto, mas curioso".

No entanto, como já discutido, o adversário malicioso é capaz de modelar comportamentos mais gerais. Ele é mais poderoso e tem a capacidade de se desviar arbitrariamente do comportamento estabelecido por um protocolo. Entre suas habilidades estão a substituição de entradas (dos participantes de seu conluio), o envio de mensagens não previstas, malformadas, fora de ordem ou duplicadas e a suspensão prematura da execução do protocolo. O poder de abortar o protocolo é de particular interesse porque pode levar à quebra de justiça.

É claro que um protocolo seguro contra um adversário semi-honesto não necessariamente o é contra um adversário malicioso. De fato, boa parte dos protocolos apresentados na seção anterior tem sua segurança trivialmente quebrada por esses adversários mais poderosos. Por isso, o objetivo desta seção é estender os resultados obtidos anteriormente de modo a definir protocolos resistentes a adversários maliciosos.

Para atingir esse objetivo, apresentaremos um *compilador* para transformar um protocolo seguro contra adversários semi-honestos em um outro resistente a adversários maliciosos. Mais especificamente, estaremos interessados em forçar o comportamento semi-honesto *fortalecido* a um adversário malicioso. Chamamos de *fortalecido* porque, embora seja muito parecido com o comportamento semi-honesto, algumas ações maliciosas ainda lhe são permitidas. Nós o definimos a seguir.

Definição 4.63 (Comportamento semi-honesto fortalecido)**.** *O comportamento semi-honesto fortalecido, com respeito a um protocolo* Π*, é uma estratégia que obedece as seguintes condições:*

(i) (Escolha de entradas:) Antes de realizar qualquer ação em Π*, um participante semi-honesto fortalecido pode escolher uma entrada* $x' \in \{0,1\}^{|x|}$ *baseando-se em sua entrada original* u*. A partir dessa escolha, o participante deve agir sempre de forma coerente com* x'*.*

(ii) (Escolha de fonte de aleatoriedade:) Um participante semi-honesto fortalecido escolhe sua fonte de aleatoriedade uniformemente entre as cadeias binárias de tamanho especificado por Π*.*

(iii) (Envio de mensagens:) A cada etapa de Π*, o participante semi-honesto fortalecido pode mandar uma mensagem em total conformidade com* Π *ou decidir abortar.*

Um adversário semi-honesto fortalecido, conforme definido acima, pode mudar as entradas de seu conluio, embora não o possa fazer durante a execução do protocolo. Além disso, tal adversário pode decidir abortar o protocolo a qualquer momento. Essas duas ações maliciosas, infelizmente, não podem ser impedidas, ou pelo menos não sem acrescentar mais suposições ao modelo de segurança. Apesar disso, um adversário semi-honesto fortalecido também pode ser encarado como um adversário *malicioso enfraquecido*, pois suas possibilidades de trapaça são severamente limitadas.

Dedicaremos o restante da seção à construção do compilador mencionado. Para isso, precisaremos também definir algumas construções auxiliares. A seção é dividida como segue. Inicialmente discutiremos, na Subseção 4.3.1, um detalhe técnico sobre canais de comunicação. Para lidar com esse detalhe, definiremos compiladores *auxiliares* nas Subseções 4.3.2 e 4.3.3. Em seguida definiremos a composição e redução de funcionalidades na Subseção 4.3.4, de maneira similar ao caso de adversários semi-honestos. Na Subseção 4.3.5 definiremos funcionalidades importantes e suas implementações. Algumas delas serão utilizadas diretamente pelo nosso compilador. Finalmente, na Subseção 4.3.6, apresentaremos a construção mais importante desta seção: o compilador propriamente dito. A partir dele seremos capazes de estabelecer o resultado final e mais importante do capítulo, sobre a possibilidade da computação segura de uma funcionalidade qualquer na presença de um adversário malicioso.

4.3.1 Canal de Comunicação

A noção de segurança contra adversários semi-honestos, conforme Definição 4.33, se baseia na indistinguibilidade das visões dos adversários real e ideal. A visão de um protocolo, de acordo com a Definição 4.31, considera a existência de canais ponto-a-ponto privados entre cada par de participantes. Dessa forma, na visão de um participante estavam apenas as mensagens recebidas por ele, uma vez que as trocadas entre os demais lhe eram inacessíveis.

Nesta seção, seguindo a apresentação em GOLDREICH (2004), consideramos que os **participantes se comunicam através de um canal de *broadcast***. Isso significa que toda mensagem enviada por um participante é recebida por todos os demais, mesmo quando endereçada a apenas um destinatário específico.

Como ainda trataremos de algumas provas de segurança considerando adversários semi-honestos, precisamos nos acomodar a essa mudança de canal de comunicação. Para isso, no entanto, pouca coisa precisa ser alterada. É necessário apenas modificar a Definição 4.31 de modo que a *visão da execução de um protocolo* contenha *todas* as mensagens trocadas durante o protocolo. Dessa forma, a definição de segurança contra adversário semi-honestos (Definição 4.33) continua a mesma.

4.3.2 Pré-Compilador

Com a adoção de um canal de *broadcast*, precisamos adaptar protocolos que executam sobre canais ponto-a-ponto. Mais especificamente os gerados de acordo com a Seção 4.2. Para isso utilizamos um procedimento para modificar esses protocolos de modo a provê-los com um canal de comunicação ponto-a-ponto, emulado a partir do canal de *broadcast*. Naturalmente que essa modificação deve preservar a segurança (privacidade) dos protocolos. Chamamos esse procedimento de *pré-compilador*.

A construção do pré-compilador é bastante intuitiva e tem como única novidade até aqui a utilização de um esquema criptográfico assimétrico. Registramos seu funcionamento a seguir.

Definição 4.64 (Pré-compilador)**.** *Seja $\Gamma = (Gen, Enc, Dec)$ um esquema criptográfico assimétrico, em que Gen é o algoritmo de geração das chaves pública e secreta, Enc é o algoritmo de encriptação e Dec o de decriptação. Seja Π um protocolo de m participantes que executa sobre um canal ponto-a-ponto privado. Definimos o protocolo Π' resultante da transformação de Π operada pelo pré-compilador como a seguir:*

E1. (Inicialização:) Cada participante P_i invoca Gen para obter o par (pk_i, sk_i) de chaves pública e secreta, para todo $i \in \{1, \cdots, m\}$.

E2. (Publicação das chaves:) Cada participante P_i publica sua chave pk_i através do canal de broadcast e mantém sua chave sk_i em segredo, para todo $i \in \{1, \cdots, m\}$.

E3. (Emulação de Π:) Toda mensagem $msg_{i\to j}$ especificada por Π a ser enviada por P_i e recebida por P_j é tratada como a seguir. O participante P_i usa a chave pk_j para gerar $c_{i\to j} \stackrel{def}{=} Enc(pk_j, msg_{i\to j})$ e o canal de broadcast para publicar $c_{i\to j}$. Todos os participantes recebem $c_{i\to j}$, mas apenas P_j é capaz de recuperar $msg_{i\to j}$ invocando $Dec(sk_j, c_{i\to j})$.

Como mostra a Definição 4.64, a ideia geral do pré-compilador é emular um canal ponto-a-ponto privado a partir de um esquema criptográfico assimétrico. Desse modo, toda mensagem enviada no protocolo original é encriptada pelo emissor com a chave pública do receptor. Dessa maneira, a mensagem pode ser publicada no canal de *broadcast* sem que os demais participantes consigam decifrá-la.

A seguir estabelecemos um resultado alternativo ao do Teorema 4.62, possibilitado pelo pré-compilador.

Lema 4.65 (Efeito do pré-compilador)**.** *Sob a hipótese da existência de coleções de permutações arapuca aprimoradas, toda m-funcionalidade pode ser privadamente computada, por um protocolo canônico, sobre um canal de broadcast.*

Esboço de Prova. Seja f uma m-funcionalidade e Π um protocolo canônico para computá-la privadamente, conforme o Teorema 4.62. Usamos uma coleção de permutações arapuca

aprimoradas para construir um esquema criptográfico assimétrico[9] e, através da Definição 4.64, obtemos Π' para computar f sobre um canal de *broadcast*.

Para provar que Π' computa privadamente f, precisamos simular a visão da execução que o adversário (semi-honesto) de Π' tem. Para isso, primeiro simulamos a visão do participante para o protocolo Π[10]. Depois encriptamos todas as mensagens enviadas ou recebidas por participantes do conluio do adversário. Note que isso é possível porque conhecemos tanto as mensagens originais quanto as chaves públicas adequadas. Fica faltando apenas as mensagens trocadas entre os participantes honestos, as quais não estão presentes na visão de Π (porque Π executa sobre um canal ponto-a-ponto), mas precisam estar na visão de Π' (porque Π' executa sobre um canal de *broadcast*). Para tratar isso, usamos as chaves públicas das partes honestas para encriptar cadeias arbitrárias.

A análise da simulação da visão do adversário de Π' é suportada por duas garantias. A primeira é a de que a visão simulada de Π é indistinguível da visão real de Π. A segunda é a da intratabilidade de se distinguir cifrotextos de Γ. Com isso, se algum algoritmo é capaz de distinguir a visão simulada, como descrita acima, da visão real de Π' então ou ele é capaz de distinguir a visão simulada de Π ou é capaz de distinguir cifrotextos do esquema utilizado. Qualquer uma das alternativas representa uma contradição às hipóteses assumidas. □

4.3.3 Pós-Compilador

O compilador principal é um procedimento que transforma um protocolo seguro contra adversários semi-honestos em um seguro contra adversário maliciosos e que executa sobre um canal de *broadcast*. Nosso objetivo, no entanto, é obter um protocolo em que os participantes se comuniquem por conexões ponto-a-ponto, que é o tipo padrão de canal de comunicação. Por isso, apresentamos outro procedimento, chamado de *pós-compilador*. Ele recebe o protocolo resultante do compilador e o transforma em outro protocolo que executa sobre canais ponto-a-ponto preservando, naturalmente, a sua segurança.

O que esse pós-compilador precisa fazer é emular um canal de *broadcast* a partir de uma rede de conexões ponto-a-ponto. Supondo a existência de um esquema de assinaturas, essa emulação equivale ao problema conhecido como *Authenticated Byzantine Agreement* (AuthBA).

Nesse problema, cada participante conhece a chave de verificação (da assinatura) dos demais. O participante P_1 possui um *bit* b e deseja que todos os demais recebam esse valor. Caso P_1 seja honesto, todos os participantes devem ser capazes de receber o valor b. Caso contrário, eles podem receber qualquer outro valor, desde que exista um consenso entre eles.

Apresentamos um protocolo para resolver esse problema a seguir.

Definição 4.66 (Construção 7.5.17 em (GOLDREICH, 2004, p.711) - Protocolo Π_{AuthBA} para o AuthBA)**.** *Seja $\{P_1, \cdots, P_m\}$ o conjunto de m participantes que desejam computar o protocolo*

[9] Ver Definição 3.26 sobre como construir tal esquema.

[10] Essa visão pode ser simulada porque, segundo o Teorema 4.60, Π é seguro e, portanto, de acordo com a Definição 4.33, existe um algoritmo probabilístico para simulá-la.

para o AuthBA. Assumimos um esquema de assinatura tal que o tamanho das assinaturas geradas dependa apenas de um parâmetro de segurança. Definimos o protocolo Π_{AuthBA} *como a seguir:*

E1. (Entradas:) O participante P_1 *fornece* $b \in \{0,1\}$.

E2. (Fase 1:) P_1 *gera uma assinatura* s_1 *de* b *e envia* (b, s_1) *para os demais participantes.* P_1 *pode suspender sua execução ao término desta fase.*

Definição Auxiliar: Uma mensagem é dita ser (v,i)*-autêntica se tem a forma* $(v, s_{p_1}, \cdots, s_{p_i})$*, em que* $p_1 = 1$*, todos os* p_j*'s são distintos e, para todo* $j \in \{1, \cdots, i\}$ *o valor* s_{p_j} *é aceito como uma assinatura para* $(v, s_{p_1}, \cdots, s_{p_{j-1}})$ *validada com a chave de verificação do participante* P_{p_j}*. Ou seja,* $s_{p_j} = Assina(Sk_{p_j}, (v, s_{p_1}, \cdots, s_{p_{j-1}}))$*, em que* Sk_{p_j} *é a chave secreta de* P_{p_j}.

E3. (Fase $i = 2, \cdots, m$*:) Cada participante, além de* P_1*, inspeciona as mensagens recebidas na Fase* $i-1$ *e envia versões assinadas das mensagens* $(\cdot, i-1)$*-autênticas que ele recebeu. Especificamente, para cada* $v \in \{0,1\}$*, se* P_j *recebeu uma mensagem* $(v, i-1)$*-autêntica* $(v, s_{p_1}, \cdots, s_{p_{i-1}})$ *tal que todos os* p_k*'s são diferentes de* j*, então* P_j *adiciona sua assinatura e envia a mensagem resultante, que é* (v,i)*-autêntica, para todos os demais.*

E4. (Saídas:) Cada participante, além de P_1*, avalia a execução do protocolo:*

(a) Se, para valores $i_0, i_1 \in \{1, \cdots, m\}$*, não necessariamente distintos, o participante recebeu uma mensagem* $(0, i_0)$*-autêntica e outra* $(1, i_1)$*-autêntica, então ele assume que* P_1 *é desonesto e define sua saída como* $\perp$ *(erro).*

(b) Se, para um (único) valor $v \in \{0,1\}$ *e algum* $i \in \{1, \cdots, m\}$*, o participante recebeu uma mensagem* (v,i)*-autêntica, ele define sua saída como* v.

(c) Se o participante não recebeu qualquer mensagem (v,i)*-autêntica, para qualquer* $v \in \{0,1\}$ *e* $i \in \{1, \cdots, m\}$*, então ele assume que* P_1 *é desonesto e define sua saída como* $\perp$ *(erro).*

Lema 4.67 (Proposição 7.5.18 em (GOLDREICH, 2004, p. 712) - Propriedades do protocolo Π_{AuthBA})**.** *Assumindo que é impraticável forjar uma assinatura válida no esquema de assinaturas utilizado, o protocolo da Definição 4.66 possui as seguintes propriedades:*

1. *Não é factível conduzí-lo de modo que as saídas de dois participantes honestos sejam diferentes.*
2. *Se* P_1 *é honesto, então todos os demais participantes honestos irão definir a entrada de* P_1 *como saída.*

Esboço de Prova. Com j e v fixos, suponha que na Fase $(i-1)$ o participante P_j recebe uma mensagem $(v, i-1)$-autêntica e suponha ainda que i é o menor inteiro para o qual isso acontece.

Nesse caso, se P_j é honesto, então ele envia uma mensagem (v,i)-autêntica na Fase i, de modo que todos os participantes recebem uma mensagem (v,i)-autêntica na Fase i. Por isso, para todo v, se um participante honesto recebe uma mensagem $(v,\cdot)$-autêntica, então todos os outros participantes honestos também recebem. Portanto concluímos que o item 1 é verdadeiro. O item 2, por sua vez, segue do item 1 e do fato de que se P_1 é honesto e tem entrada v, então todos os participantes honestos recebem uma mensagem $(v,1)$-autêntica. Além disso, nenhum deles poderá receber uma mensagem (v',i)-autêntica com $v' \neq v$ e algum $i \in \{1,\cdots,m\}$. □

Com a definição do Π_{AuthBA}, a construção do pós-compilador torna-se intuitiva e é registrada na definição a seguir.

Definição 4.68 (O pós-compilador)**.** *Seja Π um protocolo seguro contra adversários maliciosos e que executa sobre um canal de broadcast. O pós-compilador transforma Π em Π' substituindo toda utilização do canal de broadcast por uma execução de Π_{AuthBA}. Isto é, toda mensagem publicada no canal de broadcast em Π é publicada através de Π_{AuthBA} em Π'. O protocolo Π' executa sobre um canal ponto-a-ponto. Segundo GOLDREICH (2004), a seguinte ressalva deve ser feita em relação à composição das execuções de Π_{AuthBA}: um atacante pode replicar uma mensagem válida de uma execução passada de Π_{AuthBA}. Uma maneira de lidar com esse problema é anexar identificadores de execução nas mensagens enviadas em Π_{AuthBA}. Esses identificadas devem ser gerados pelo protocolo de mais alto nível, no caso o Π'.*

Usamos o lema a seguir para estabelecer que o efeito do nosso pós-compilador é, de fato, o esperado.

Lema 4.69 (Proposição 7.5.19 em GOLDREICH (2004) - Efeito do pós-compilador)**.** *Sob a hipótese da existência de funções unidirecionais, qualquer m-funcionalidade que pode ser computada seguramente na presença de um adversário malicioso usando um canal de broadcast pode também ser computada seguramente usando-se um canal ponto-a-ponto.*

Esboço de Prova. Seja Π um protocolo que computa seguramente uma m-funcionalidade f na presença de um adversário malicioso usando um canal de *broadcast*. Seja Π' o resultado da transformação de Π operada pelo pós-compilador. O lema segue do fato de que Π' é seguro e usa um canal ponto-a-ponto.

A prova de que Π' é seguro envolve criar um adversário para a execução real de Π (o qual é seguro) sob a hipótese de que existe um adversário S para Π'. A construção de tal adversário é simples. Basta copiar todo o comportamento de S, publicando no canal de *broadcast* todas as mensagens enviadas através do protocolo Π_{AuthBA}. □

4.3.4 Composição e Redução de Funcionalidades

Analogamente ao que foi feito para o caso de adversários semi-honestos, vamos apresentar algumas definições relativas à composição e redução de funcionalidades. Esses conceitos nos permitem modularizar a construção de protocolos mais complexos.

Podemos reaproveitar quase tudo o que foi definido na Subseção 4.2.2, mas é importante ressaltar uma diferença na definição de oráculo. A ideia geral continua a mesma, ou seja, um oráculo representa uma TPC para computar alguma funcionalidade. No entanto, de acordo com a Definição 4.34, essa execução ideal pode ser abortada pelo adversário (o que não acontece no caso semi-honesto). Para capturar essa diferença, assumimos que um oráculo sempre responde primeiramente ao participante que o invocou. Esse participante então decide se a execução será suspensa ou não, isto é, se os demais participantes devem também receber suas respectivas saídas. Como consequência disso, *apenas* o participante que *invocou* o oráculo pode abortá-lo[11].

Com essa diferença em mente, notamos que as demais definições são parecidas com as do cenário de um adversário semi-honesto. Reescrevemos essas definições abaixo.

Definição 4.70 (Redução segura)**.** *Sejam f e g duas funcionalidades. Definimos que f se reduz seguramente a g se existe um protocolo $\Pi_{f\to g}$ para computar f de forma segura com acesso a um oráculo para g. Também definimos que $\Pi_{f\to g}$ reduz seguramente f a g.*

Definição 4.71 (Protocolo de composição)**.** *Sejam f e g duas funcionalidades, $\Pi_{f\to g}$ um protocolo que reduz seguramente f a g e Π_g um protocolo que computa g de maneira segura. Definimos Π_f o protocolo resultante da substituição de cada chamada oracular em $\Pi_{f\to g}$ pela execução de uma instância de Π_g.*

Teorema 4.72 (Teorema da composição)**.** *Sejam f e g duas funcionalidade. Se f se reduz seguramente a g e Π_g é um protocolo que computa g seguramente, então o protocolo Π_f da Definição 4.71 computa f seguramente.*

4.3.5 Funcionalidades Auxiliares

Seguindo a proposta de modularizar nossa apresentação, vamos definir algumas funcionalidades que servirão como subrotinas para os protocolos do restante desta seção. Algumas delas terão um papel central, sendo, inclusive, invocadas diretamente pelo compilador construído mais adiante.

Ressaltamos que em todos os protocolos definidos nesta subseção os participantes se comunicam através de um canal de *broadcast*.

4.3.5.1 *Broadcast* privado

A ideia geral da funcionalidade f_{broad} para condução de *broadcasts* privados é prover um canal de *broadcast* isolado do ambiente externo. Para entender melhor sua utilidade, considere um adversário (malicioso) que não controla nenhum participante, mas ainda é capaz de interceptar mensagens no canal de comunicação. Nesse cenário, qualquer mensagem publicada no canal de *broadcast* pode ser coletada por ele. No entanto, as publicadas através da funcionalidade f_{broad}

[11] Note que definimos a execução ideal na presença de adversários maliciosos de maneira muito similar (Definição 4.34). A diferença é que na execução ideal o *adversário* é quem decide abortar e não um dos participantes, como na chamada oracular.

são conhecidas apenas por seus participantes, ficando fora do alcance de qualquer adversário externo. Naturalmente, um adversário que controla um conluio *não vazio* de participantes tem tando poder sobre as mensagens enviadas por f_{broad} quanto sobre as enviadas pelo canal de *broadcast* usual.

Definimos f_{broad} a seguir.

Definição 4.73 (Funcionalidade f_{broad} para condução de *broadcasts* privados)**.** *Seja $\alpha \in \{0,1\}^*$ uma cadeia binária. Definimos a funcionalidade f_{broad} para condução de broadcasts privados como a seguir:*

$$f_{broad}(\alpha, 1^{|\alpha|}, \cdots, 1^{|\alpha|}) \stackrel{def}{=} (\alpha, \cdots, \alpha)$$

Prosseguimos com a definição de um protocolo para computar f_{broad} e, em seguida, estabelecemos que ele o faz de maneira segura.

Definição 4.74 (Protocolo Π_{broad} para condução de *broadcasts* privados)**.** *Supondo a existência de um esquema de assinaturas e de um esquema criptográfico assimétrico, definimos o protocolo Π_{broad} para computar a funcionalidade da Definição 4.73 como a seguir:*

E1. (Entradas:) O participante P_1 fornece a cadeia $\alpha \in \{0,1\}^*$, os demais fornecem a cadeia $1^{|\alpha|}$.

E2. (Esquema de assinaturas:) Cada participante gera um par de chaves para o esquema de assinaturas e publica sua chave pública (de verificação) pelo canal usual de broadcast.

E3. (Esquema de encriptação:) Cada participante gera um par de chaves, uma pública e outra secreta, para um esquema criptográfico assimétrico. Esse passo serve para emular um canal de comunicação ponto-a-ponto, como feito na Definição 4.64 (pré-compilador).

E4. (Protocolo Π_{AuthBA} como subrotina:) Os participantes utilizam o protocolo Π_{AuthBA} da Definição 4.66 para que P_1 envie seguramente o valor α para os demais.

E5. (Saídas:) P_1 define α como sua saída e os demais definem o valor obtido em Π_{AuthBA} como sua saída.

Lema 4.75 (Segurança do protocolo Π_{broad})**.** *Sob a hipótese da existência de permutações arapuca, o protocolo Π_{broad} computa seguramente a funcionalidade da Definição 4.73.*

Esboço de Prova. Usamos a hipótese da existência de permutações arapuca para construir os esquemas de assinaturas e de criptografia assimétrica usados no protocolo.

A ideia geral de Π_{broad} é, primeiramente, emular um canal de comunicação ponto-a-ponto, utilizando um esquema criptográfico assimétrico, para então executar uma instância do protocolo Π_{AuthBA}. Por isso, seguindo a apresentação em (GOLDREICH, 2004, p.717), concluímos que a segurança de Π_{broad} se reduz naturalmente à segurança das construções Π_{AuthBA} e da Definição 4.64. □

4.3.5.2 Transmissão de Imagem

Na funcionalidade para transmissão de imagem, o participante P_1 possui um valor secreto α e quer enviar aos demais o valor $j(\alpha)$, para alguma função j predefinida. Naturalmente, é exigido que, ao final da execução, todo o conhecimento obtido por um participante sobre α seja o extraído de $j(\alpha)$. É também exigido que P_1 de fato conheça o valor α correspondente ao $j(\alpha)$ enviado. O participante P_1 terá trapaceado se não cumprir essa exigência e isso deve ser detectável pelos demais. Esse último requisito é, essencialmente, o que nos impede de propor um protocolo em que P_1 simplesmente publica $j(\alpha)$ através do canal de *broadcast* diretamente (ou da funcionalidade f_{broad}).

Registramos a definição da funcionalidade f_{img} a seguir.

Definição 4.76 (Funcionalidade f_{img} para transmissão de imagem)**.** *Seja $\alpha \in \{0,1\}^*$ uma cadeia binária e j uma função predefinida e computável em tempo polinomial. Definimos a funcionalidade de transmissão de imagem f_{img} como a seguir:*

$$f_{img}(\alpha, 1^{|\alpha|}, \cdots, 1^{|\alpha|}) \stackrel{def}{=} (\lambda, j(\alpha), \cdots, j(\alpha))$$

Garantir que o valor recebido pelos participantes é de fato $j(\alpha)$ sem que, para isso, P_1 revele α é a principal dificuldade em se implementar f_{img}. Para contorná-la, utilizamos um sistema de provas fortes de conhecimento de conhecimento zero. O protocolo para computar f_{img} é definido a seguir.

Definição 4.77 (Protocolo Π_{img} para transmissão de imagem)**.** *Seja j uma função computável em tempo polinomial. Por simplicidade, assumimos que $|j(x)| = |j(y)|$ sempre que $|x| = |y|$. Seja $R \stackrel{def}{=} \{(v,w)|v = j(w)\}$ a relação definida pela função j e f_{broad} um oráculo para a funcionalidade de condução de broadcasts privados. Definimos o protocolo Π_{img} para transmissão de imagem como a seguir:*

E1. (Entradas:) P_1 fornece $\alpha \in \{0,1\}^*$ e os demais fornecem $1^{|\alpha|}$.

E2. (Publicação de $j(\alpha)$:) O objetivo dessa etapa é fazer P_1 enviar $j(\alpha)$ a todos os demais participantes.

Para isso, P_1 invoca o oráculo f_{broad} fornecendo $v \stackrel{def}{=} j(\alpha)$ como entrada. Os demais fornecem $1^{|\alpha|}$.

E3. (Prova de conhecimento de conhecimento zero:) O objetivo dessa etapa é fazer com que P_1 prove a todos os demais que possui o valor α tal que $j(\alpha) = v$.

Para isso, cada participante P_i, para $i \in \{2, \cdots, m\}$, invoca um sistema de provas fortes de conhecimento de conhecimento zero para a relação R, de modo que P_1 faça o papel de provador e P_i o de verificador. Ambos P_1 e P_i tem acesso a v e, além disso, P_1 tem entrada auxiliar α. O objetivo de P_1 é provar a P_i que conhece α tal que $(v, \alpha) \in R$.

Se P_i rejeitar essa prova, ele deve publicar a fonte de aleatoriedade usada para que os demais participantes possam comprovar que sua rejeição foi justificada. Todas as trocas de mensagens devem ocorrer através do oráculo f_{broad}.

E4. (Saídas:) Para $i \in \{2, \cdots, m\}$, se P_i receber uma rejeição justificada para alguma prova de P_1, então ele admite que P_1 é desonesto e define sua saída como $\perp$ (erro). Senão P_i define sua saída como v.

Finalmente, estabelecemos que, de fato, o protocolo Π_{img} da Definição 4.77 computa f_{img} de maneira segura.

Lema 4.78 (Segurança do protocolo Π_{img})**.** *Se a prova de conhecimento em Π_{img} for conduzida por um sistema de provas fortes de conhecimento de conhecimento zero, então Π_{img} computa f_{img} de forma segura com o auxílio do oráculo f_{broad}.*

Esboço de Prova. Note que P_1 será considerado desonesto apenas se algum participante rejeitar justificadamente uma de suas provas de conhecimento. Essa observação reduz, informalmente, a correção do protocolo à segurança do sistema de provas fortes de conhecimento de conhecimento zero. Em particular, se P_1 agir honestamente, isto é, publicar o valor $j(\alpha)$ conhecendo α, então será infactível para qualquer participante justificadamente rejeitar uma prova de conhecimento exibida por P_1. Isso decorre da propriedade de corretude do sistema de prova de conhecimento. □

4.3.5.3 Computação Multilateral Autenticada

Na funcionalidade para computação multilateral autenticada, o participante P_1 possui um certo α e quer enviar aos demais o valor $g(\alpha)$, para alguma função g predefinida. Os demais participantes, por sua vez, possuem o valor $h(\alpha)$ que serve para *autenticar* o valor $g(\alpha)$ recebido. Naturalmente exigimos que o valor $g(\alpha)$ seja autenticado pelo valor $h(\alpha)$ sem que para isso α seja revelado.

Definimos, a seguir, a funcionalidade f_{cma} para computação multilateral autenticada.

Definição 4.79 (Funcionalidade f_{cma} para computação multilateral autenticada)**.** *Sejam g e h duas funções calculáveis em tempo polinomial. Assumimos, por simplicidade, que h é tal que $|h(\alpha)| = |\alpha|$. Se h não tiver essa propriedade, podemos construir $\alpha' = (\alpha, 1^{|h(\alpha)|})$ e definir h' tal que $h'(\alpha') = (h(\alpha), 1^{|\alpha|})$. Definimos f_{cma} como a m-funcionalidade para a computação de g autenticada por h a seguir:*

$$f_{cma}(\alpha, \beta_2, \cdots, \beta_m) \stackrel{def}{=} (\lambda, v_2, \cdots, v_m),$$

em que $v_i \stackrel{def}{=} \begin{cases} g(\alpha) \text{ se } \beta_i = h(\alpha), \\ (h(\alpha), g(\alpha)) \text{ caso contrário.} \end{cases}$

O caso em que $v_i = (h(\alpha), g(\alpha))$ serve para sinalizar um comportamento malicioso.

Como ressalta (GOLDREICH, 2004, p.672 e p.717), algumas mudanças poderiam deixar a Definição 4.79 mais intuitiva. Poder-se-ia, por exemplo, exigir que $\beta_2 = \cdots = \beta_m$ e que $v_i \stackrel{def}{=} \perp$ se $\beta_i \neq h(\alpha)$. Essas alterações, no entanto, contribuiriam para tornar a implementação dessa funcionalidade mais complexa, enquanto que a formulação da Definição 4.79 já é suficiente para o restante da apresentação.

Na sequência definimos um protocolo para computar f_{cma}.

Definição 4.80 (Protocolo Π_{cma} para computação multilateral autenticada)**.** *Seja g e h funções como na Definição 4.79 e f_{img} um oráculo para a transmissão de imagem. Definimos Π_{cma} como a seguir:*

E1. (Entradas:) O participante P_1 fornece $\alpha \in \{0,1\}^*$ e os participantes P_i fornecem $\beta_i \in \{0,1\}^{|\alpha|}$, para todo $i \in \{2,\cdots,m\}$.

E2. (Transmissão de imagem:) O objetivo desta etapa é fazer P_1 transmitir aos demais participantes o par $(h(\alpha), g(\alpha))$.

Para isso, P_1 inicia uma chamada oracular a f_{img}, fornecendo α como entrada. Os demais participantes fornecem $1^{|\alpha|}$ e esperam receber $j(\alpha) \stackrel{def}{=} (h(\alpha), g(\alpha))$ como saída do oráculo.

E3. (Saídas:) Para todo $i \in \{2,\cdots,m\}$, seja (u,v) o valor recebido por P_i da chamada oracular da etapa anterior. P_i define v como sua saída se $u = \beta_i$ ou (u,v) caso contrário. Note que se P_1 abortar a etapa anterior, P_i recebe $\perp$ como saída e, nesse caso, também suspende a execução, definindo sua saída como $\perp$ (erro).

Finalmente, estabelecemos que o protocolo da Definição 4.80 é seguro.

Lema 4.81 (Segurança do protocolo Π_{cma})**.** *Sob a hipótese da existência de permutações arapuca, o protocolo Π_{cma} computa seguramente a funcionalidade f_{cma} com auxílio de um oráculo para f_{img}.*

Esboço de Prova. De maneira informal, toda a segurança do protocolo se reduz à segurança de f_{img}, utilizada por P_1 para transmitir $(h(\alpha), g(\alpha))$ a todos os demais.

Note que todo participante obtém o par $(h(\alpha), g(\alpha))$, mesmo ao fornecer uma entrada arbitrária $h(\alpha')$ na chamada a f_{img}. Dessa forma, pode ser que um participante aprenda algo sobre α que ele ainda não sabia (nem que seja somente os próprios valores $g(\alpha)$ e $h(\alpha)$). No entanto, esse cenário também é possível na computação ideal da funcionalidade f_{cma}, de modo que o protocolo Π_{cma} não acrescenta nenhuma vulnerabilidade que já não exista.

Ressaltamos, no entanto, que se h e g forem unidirecionais, podemos esperar que o par $(h(\alpha), g(\alpha))$, naturalmente, não permita a reconstrução de α.

□

4.3.5.4 Geração de Fonte de Aleatoriedade

Na funcionalidade para geração de fonte de aleatoriedade, o participante P_1 quer obter uma cadeia binária r enquanto os demais participantes querem receber o valor $k(r)$, para alguma função k predefinida. Exige-se que r seja escolhida uniformemente do conjunto $\{0,1\}^{poly(n)}$, em que n é o parâmetro de segurança.

Vamos encarar essa funcionalidade como uma maneira de prover P_1 com uma fonte de aleatoriedade para ser utilizada durante a execução de algum protocolo. A função k, por sua vez, será utilizada como uma forma de P_1 se comprometer com esse valor.

Definimos formalmente a funcionalidade f_{alea} para geração de fonte de aleatoriedade a seguir.

Definição 4.82 (Funcionalidade f_{alea} para geração de fonte de aleatoriedade)**.** *Seja $t : \mathbb{N} \to \mathbb{N}$ um polinômio e k uma função computável em tempo polinomial. Definimos a m-funcionalidade f_{alea} para geração de fonte de aleatoriedade como a seguir:*

$$f_{alea}(1^n, \cdots, 1^n) \stackrel{def}{=} (r, k(r), \cdots, k(r))$$

em que r é escolhido uniformemente do conjunto $\{0,1\}^{t(n)}$.

A seguir definimos um protocolo para computar f_{alea}.

Definição 4.83 (Protocolo Π_{alea} para geração de fonte de aleatoriedade)**.** *Seja C um esquema de comprometimento, t e k funções como na Definição 4.82 e f_{cma} um oráculo para computação multilateral autenticada. Definimos Π_{alea} para computar f_{alea} como a seguir:*

E1. (Entradas:) Todos os participantes fornecem 1^n como entrada. Seja $t \stackrel{def}{=} t(n)$.

E2. (Sorteios iniciais:) Cada participante P_i escolhe uniformemente $r_i \in \{0,1\}^t$ e $s_i \in \{0,1\}^{t \cdot n}$.

E3. (Comprometimento com os valores sorteados:) O objetivo dessa etapa é fazer com que cada participante P_i envie o valor $c_i \stackrel{def}{=} \overline{C}_{s_i}(r_i)$ para todos os demais, se comprometendo, portanto, com a cadeia r_i.

Isso é feito da seguinte maneira. Para todo $i \in \{1, \cdots, m\}$, P_i invoca o oráculo f_{cma}, fornecendo como entrada o valor (r_i, s_i) e fazendo o papel de P_1 nessa chamada. Cada um dos demais participantes fornece $1^{t+t \cdot n}$. Se esse valor for igual a $h^{(E3)}(r_i, s_i) \stackrel{def}{=} 1^{|r_i|+|s_i|}$, o participante recebe $g^{(E3)}(r_i, s_i) \stackrel{def}{=} \overline{C}_{s_i}(r_i)$.

E4. (Revelação dos valores sorteados:) O objetivo dessa etapa é fazer com que cada participante P_i, com $i \neq 1$, revele a cadeia r_i com a qual se comprometeu na etapa anterior.

Isso é feito da seguinte maneira. Para todo $i \in \{2, \cdots, m\}$, P_i invoca o oráculo f_{cma} fornecendo (r_i, s_i) como entrada e fazendo o papel de P_1 nessa chamada. Cada um dos demais participantes fornece c_i (publicado na etapa anterior). Se esse valor for igual a $h^{(E4)}(r_i, s_i) \stackrel{def}{=} \overline{C}_{s_i}(r_i)$, o participante recebe $g^{(E4)}(r_i, s_i) \stackrel{def}{=} r_i$.

Vamos definir a saída recebida por P_j na chamada oracular iniciada por P_i como r_i^j. Se P_i abortar a chamada oracular, então $r_i^j \stackrel{def}{=} \perp$. Caso contrário $r_i^j \stackrel{def}{=} r_i$. Por simplicidade, assumimos que $r_j^j \stackrel{def}{=} r_j$.

E5. (Cálculo da fonte de aleatoriedade:) Nesse etapa, P_1 calcula sua fonte de aleatoriedade $r \stackrel{def}{=} \bigoplus_{i=1}^m r_i^1$ e envia o valor $k(r)$ a todos os demais.

Definição auxiliar: Para todo $j \in \{1,\cdots,m\}$, definimos $r^j \stackrel{def}{=} \bigoplus_{i=2}^m r_i^j$. Note que, se P_j não abortou, temos que $r_i^j = r_i$ para todo $i \in \{1,\cdots,m\}$. Dessa forma, temos que $\bigoplus_{i=1}^m r_i^1 = r_1 \oplus r^1$ e podemos reescrever $r \stackrel{def}{=} r_1 \oplus r^1$.

P_1 invoca o oráculo f_{cma} fornecendo (r_1, s_1, r^1) como entrada. Cada participante P_j, com $j \neq 1$, fornece (c_1, r^j) como entrada. Se (c_1, r^j) for igual a $h^{(E5)}(r_1, s_1, r^1) \stackrel{def}{=} (\overline{C}_{s_1}(r_1), r^1)$, então P_j recebe $g^{(E5)}(r_1, s_1, r^1) \stackrel{def}{=} k(r_1 \oplus r^1) = k(r)$. Caso P_1 aborte ou P_j receba um valor inválido, P_j define sua saída como $\perp$, indicando que P_1 agiu desonestamente.

E6. (Saídas:) Se P_1 não abortou (com saída $\perp$), ele define sua saída como r. Cada participante P_j, com $j \neq 1$, define sua saída de acordo com a etapa anterior, isto é, ou $\perp$ (se P_1 agiu maliciosamente) ou $k(r)$.

A principal dificuldade da implementação de f_{alea}, exibida acima, é garantir que a saída r de P_1 é de fato escolhida uniformemente dentre os valores $\{0,1\}^t$. Para isso, a estratégia empregada é semelhante a da divisão de *bits* que apresentamos na Definição 4.46. Ou seja, o valor r é obtido a partir da soma de valores gerados *independentemente* pelos demais participantes. Se pelo menos um desses valores for escolhido uniformemente, o que equivale a dizer que pelo menos um dos participantes é honesto, o resultado r também será uniforme.

A seguir estabelecemos a segurança do protocolo Π_{alea}.

Lema 4.84 (Segurança do protocolo Π_{alea})**.** *O protocolo Π_{alea} computa seguramente a funcionalidade f_{alea} para geração de fonte de aleatoriedade com o auxílio de um oráculo f_{cma} para a funcionalidade de computação multilateral autenticada.*

Esboço de Prova. Intuitivamente, tanto a correção como a segurança do protocolo Π_{alea} seguem da utilização do oráculo f_{cma}. Em cada uma da etapas descritas, ou os participantes obtém valores corretos das chamadas oraculares, ou ficam cientes de que houve um comportamento malicioso.

Note que o participante P_1 pode trapacear gerando a cadeia r_1 de maneira não uniforme. No entanto, como já argumentamos anteriormente, devido à propriedades da operação $\oplus$, basta que um valor r_i seja uniformemente gerado, por um participante honesto, para que a saída $r = \bigoplus_{i=1}^m r_i^1$ de P_1 tenha distribuição uniforme. □

4.3.5.5 Comprometimento com uma Entrada

Na funcionalidade para comprometimento com uma entrada, o participante P_1 possui uma certa cadeia binária x e quer se comprometer com ela. Isto é, para alguma cadeia aleatória r e um esquema de comprometimento C predeterminado, P_1 quer enviar o valor $\overline{C}_r(x)$ para os demais.

A seguir definimos a funcionalidade f_{comp} para comprometimento com uma entrada.

Definição 4.85 (Funcionalidade f_{comp} para comprometimento com uma entrada)**.** *Seja C um esquema de comprometimento, $x \in \{0,1\}^n$ uma cadeia binária e n um parâmetro de segurança. Definimos a m-funcionalidade f_{comp} como a seguir:*

$$f_{comp}(x, 1^{|x|}, \cdots, 1^{|x|}) \stackrel{def}{=} (r, \overline{C}_r(x), \cdots, \overline{C}_r(x)).$$

em que r é escolhido uniformemente do conjunto $\{0,1\}^{n^2}$.

O nosso objetivo, com a funcionalidade f_{comp}, é forçar um participante a se comprometer com sua entrada para um protocolo, de modo a não conseguir mudá-la sem que os demais participantes percebam. Além disso, também usamos f_{comp} para garantir que a escolha da entrada de um participante é independente das escolhas dos demais. Exploraremos essas duas qualidades de f_{comp} no compilador que apresentaremos mais adiante.

A seguir, apresentamos um protocolo Π_{comp} para computar f_{comp}.

Definição 4.86 (Protocolo Π_{comp} para comprometimento com uma entrada)**.** *Seja $x \in \{0,1\}^n$ uma cadeia binária, n um parâmetro de segurança e f_{cma} e f_{alea} oráculos para as funcionalidades de computação multilateral autenticada e de geração de fonte de aleatoriedade, respectivamente. Seja ainda C um esquema de comprometimento. Definimos o protocolo Π_{comp} para computar f_{comp} como a seguir:*

E1. (Entradas:) O participante P_1 fornece $x \in \{0,1\}^n$ e os demais fornecem 1^n.

E2. (Sorteio auxiliar:) P_1 seleciona uniformemente $r' \in \{0,1\}^{n^2}$.

E3. (Comprometimento inicial:) Nessa etapa P_1 envia $c' \stackrel{def}{=} \overline{C}_{r'}(x)$ para todos os demais participantes.

Para isso, P_1 invoca o oráculo f_{cma} fornecendo como entrada o par (x, r'). Cada um dos demais participantes fornece 1^{n+n^2}. Se esse valor for igual a $h^{(E3)}(x, r') \stackrel{def}{=} 1^{|x|+|r'|}$, o participante recebe $g^{(E3)}(x, r') \stackrel{def}{=} \overline{C}_{r'}(x)$.

E4. (Gerando a fonte de aleatoriedade:) Nessa etapa, P_1 gera um par de valores aleatórios (r, r'') uniformemente distribuídos e envia o valor $c'' \stackrel{def}{=} \overline{C}_{r''}(r)$ para os demais participantes.

Para isso, P_1 invoca o oráculo f_{alea} fornecendo 1^n, assim como os demais participantes. Vamos supor que o tamanho da saída recebida por P_1 seja $t(n) \stackrel{def}{=} n^2 + n^3$ e que essa saída

seja a sequência (r, r''), *com* $r \in \{0,1\}^{n^2}$ *e* $r'' \in \{0,1\}^{n^3}$ *uniformemente distribuídos. Os demais participantes recebem* $k(r, r'') \stackrel{def}{=} \overline{C}_{r''}(r)$.

E5. (Comprometimento com a entrada:) O objetivo dessa etapa é fazer P_1 *se comprometer com sua entrada* x *utilizando a cadeia* r, *da etapa anterior, como fonte de aleatoriedade.*

Para isso, P_1 *invoca o oráculo* f_{cma} *fornecendo* (x, r, r', r'') *como entrada. Cada um dos demais participantes fornece* (c', c''). *Se esse valor for igual a* $h^{(E5)}(x, r, r', r'') \stackrel{def}{=} (\overline{C}_{r'}(x), \overline{C}_{r''}(x))$, *o participante recebe* $g^{(E5)}(x, r, r', r'') \stackrel{def}{=} \overline{C}_r(x)$.

E6. (Saídas:) P_1 *define* r *como sua saída. Cada um dos demais participantes define sua saída como* $\overline{C}_r(x)$ *ou* $\perp$ *caso* P_1 *tenha abortado ou agido maliciosamente.*

Finalmente, registramos a segurança de Π_{comp} a seguir.

Lema 4.87 (Segurança de Π_{comp})**.** *O protocolo* Π_{comp} *computa seguramente* f_{comp} *com auxílio de oráculos para* f_{cma} e f_{alea}.

Esboço de Prova. Informalmente, a segurança de Π_{comp} segue diretamente da utilização dos oráculos f_{cma} e f_{alea}.

Note que em Π_{comp} o participante P_1 se compromete duas vezes com x, gerando os valores $\overline{C}_{r'}(x)$ e $\overline{C}_r(x)$. De fato, se isso fosse feito somente em E3, não seria possível garantir a correção do protocolo, uma vez que P_1 poderia usar um valor r' não uniformemente distribuído para gerar $\overline{C}_{r'}(x)$. Esse comprometimento inicial serve, no entanto, para impedir que P_1 se comprometa, em E5, com um valor x' escolhido *em função* de r. □

4.3.6 O Compilador

O compilador que apresentaremos nesta subseção é um procedimento que transforma um protocolo Π, seguro contra adversários semi-honestos, em um protocolo Π', seguro contra adversários maliciosos. A ideia geral desse compilador é gerar Π' de modo a emular o funcionamento de Π, mas limitando as ações maliciosas que podem ser tomadas pelos participantes. O protocolo resultante funciona em três etapas:

Etapa 1 - *Comprometimento com a entrada:* Cada participante se compromete com sua entrada, tornando-se impossível substituí-la durante o protocolo sem que os demais percebam.

Etapa 2 - *Geração da fonte de aleatoriedade:* Cada participante obtém uma cadeia binária uniformemente escolhida para ser utilizada como fonte de aleatoriedade e se compromete com ela.

Dessa forma, os participantes ficam impossibilitados de mudar suas fontes de aleatoriedade durante o protocolo ou de escolhê-las não uniformemente.

Etapa 3 - *Aderência ao protocolo:* Cada mensagem é enviada conforme a especificação do protocolo Π. Isto é, o emissor gera a mensagem em função (i) da entrada com a qual ele se comprometeu na Etapa 1, (ii) da fonte de aleatoriedade com a qual ele se comprometeu na Etapa 2 e (iii) das mensagem que ele recebeu até o momento. Em outras palavras, cada participante gera sua próxima mensagem de acordo com sua visão atual do protocolo.

Dessa forma, ou cada participante gera suas mensagens honestamente ou sua desonestidade é detectada pelos demais.

A Etapa 1 evita a chamada substituição de entradas. Não podemos evitar que um participante escolha sua entrada arbitrariamente, mas podemos impedir que ele a substitua segundo sua conveniência durante a execução do protocolo. Em particular, podemos obrigá-lo a escolher sua entrada de maneira independente das escolhas dos demais. Para cumprir o objetivo dessa etapa utilizaremos a funcionalidade f_{comp}.

Já a Etapa 2 é bastante semelhante a Etapa 1, mas trata da escolha da fonte de aleatoriedade a ser utilizada por cada participante durante o protocolo. A implementação dessa etapa se baseia na funcionalidade f_{alea}.

A Etapa 3, por sua vez, é a que de fato limita as ações maliciosas que os participantes podem tomar. Um dos maiores objetivos dela é impedir que um participante trapaceie *sem ser percebido*. Entre as possíveis trapaças estão a suspensão da execução (aborto) e o envio de mensagens inadequadas segundo o protocolo. A implementação dessa etapa se baseia na funcionalidade f_{cma}.

A Definição 4.88 a seguir define de maneira formal e mais aprofundada os objetivos de cada uma das três etapas acima e como eles são alcançados.

Definição 4.88 (O Compilador)**.** *Seja Π um protocolo de m participantes seguro contra adversários semi-honestos. O canal de comunicação utilizado é o de broadcast. Seja f_{comp}, f_{alea} e f_{cma} oráculos para as funcionalidades para comprometimento com uma entrada, geração de fonte de aleatoriedade e computação multilateral autenticada, respectivamente. Seja ainda C um esquema de comprometimento. Definimos o compilador como o procedimento que transforma Π em um protocolo Π' como definido a seguir:*

E1. (Entradas:) Cada participante P_i fornece $x_i \in \{0,1\}^n$, para todo $i \in \{1,\cdots,m\}$.

E2. (Etapa de comprometimento com as entradas:) O objetivo desta etapa é fazer com que cada participante P_i se comprometa com sua entrada x_i utilizando a funcionalidade f_{comp}.

Para isso, para todo $i \in \{1,\cdots,m\}$, o participante P_i invoca o oráculo f_{comp} fornecendo x_i como entrada e fazendo o papel de P_1 nessa chamada. Os demais participantes fornecem 1^n como entrada. Como saída, P_i recebe ρ_i e os demais recebem $\gamma_i \stackrel{def}{=} \overline{C}_{\rho_i}(x_i)$.

E3. (Etapa de geração de fonte de aleatoriedade:) O objetivo desta etapa é fornecer a cada participante uma cadeia binária, com distribuição uniforme, para ser usada como fonte de aleatoriedade na emulação de Π.

Para isso, para todo $i \in \{1, \cdots, m\}$, *o participante* P_i *invoca o oráculo* f_{alea} *fornecendo* 1^n *como entrada e fazendo o papel de* P_1 *nessa chamada. Os demais participantes também fornecem* 1^n. *Como saída,* P_i *recebe* (r_i, ω_i) *e os demais recebem* $\delta_i \stackrel{def}{=} \overline{C}_{\omega_i}(r_i)$.

E4. (Etapa de emulação de Π*:) O objetivo desta etapa é emular o protocolo* Π *forçando o comportamento semi-honesto fortalecido a todo participante. Para isso, utilizamos o oráculo* f_{cma} *de forma a garantir que cada mensagem seja gerada de acordo com a especificação de* Π. *Além disso, a própria definição de* f_{cma} *possibilita que os participantes detectem um possível comportamento malicioso do emissor.*

Suponha que P_j *seja o emissor da próxima mensagem a ser publicada no canal de broadcast, de acordo com a especificação de* Π. *Definimos as funções* g, h *e as entradas* $\alpha, \beta_1, \cdots, \beta_m$, *seguindo a nomenclatura da Definição 4.79 (*f_{cma}*), como a seguir:*

- α *é definida como* $(\alpha_1, \alpha_2, \alpha_3)$, *em que* $\alpha_1 \stackrel{def}{=} (x_j, \rho_j)$ *(entrada e saída de* P_j *na chamada a* f_{comp} *para comprometimento com a entrada),* $\alpha_2 \stackrel{def}{=} (r_j, \omega_j)$ *(saída recebida por* P_j *na chamada a* f_{alea} *para geração de fonte de aleatoriedade) e* α_3 *é a sequência de mensagens que* P_j *teve acesso até agora na emulação de* Π.
- *Para todo* $i \in \{1, \cdots, m\}$, β_i *é igual a* $\beta \stackrel{def}{=} (\gamma_j, \delta_j, \alpha_3)$, *em que* γ_j *e* δ_j *são as saídas das chamadas a* f_{comp} *e* f_{alea} *feitas por* P_j *nas etapas 2 e 3. O valor* α_3 *é como no item anterior. Note que todos os participantes possuem* α_3 *porque a comunicação se dá sobre um canal de broadcast.*
- *A função* h *é definida como* $h((v_1, s_1), (v_2, s_2), v_3) \stackrel{def}{=} (\overline{C}_{s_1}(v_1), \overline{C}_{s_2}(v_2), v_3)$. *Em especial,* $h(\alpha_1, \alpha_2, \alpha_3) = \beta$.
- *A função* g *é a que determina a próxima mensagem a ser enviada por* P_j *em* Π. *Ela é computada (em tempo polinomial) a partir da entrada* x_j *(contida em* α_1*), da fonte de aleatoriedade* r_j *(contida em* α_2*) e de todas as mensagens recebidas por* P_j *até o momento (exatamente o significado de* α_3*). Ou seja, a próxima mensagem que* P_j *deve enviar, segundo* Π, *é definida por* $g(\alpha_1, \alpha_2, \alpha_3)$.

Dessa forma, ao invocar f_{cma} *com* α, g *e* h *como definidas acima,* P_j *envia a sua próxima mensagem* $g(\alpha)$ *para todos os demais. Essa mensagem é autenticada pelo valor* $h(\alpha)$. *Ou seja,* $h(\alpha)$ *garante que* $g(\alpha)$ *foi gerada de acordo com o estabelecido por* Π *e usando a entrada e fonte de aleatoriedade com as quais* P_j *se comprometeu. Os demais participantes podem detectar um comportamento malicioso de* P_j *através do recebimento de* $(h(\alpha), g(\alpha))$. *Ainda outra possibilidade é o recebimento de* $\perp$ *caso* P_j *tenha decidido abortar.*

E5. (Saídas:) Se algum participante abortar durante qualquer etapa, todos os demais também suspendem a execução e definem suas saídas como $\perp$. Caso contrário, ao fim da etapa de emulação, cada participante obtém sua saída em Π e a define como saída em Π'.

As etapas E2 e E3 definidas acima podem ser vistas como *auxiliares* para o compilador. Elas fazem com que os participantes se comprometam com suas entradas e fontes de aleatoriedade. No entanto, é na etapa E4 que a aderência a esses valores é de fato exigida.

Em E4 assume-se a existência de uma função g que determina a próxima mensagem a ser enviada no protocolo Π. Essa função recebe como entrada a *visão parcial* da execução do participante emissor. Isto é, a entrada e a fonte de aleatoriedade do participante e as mensagens enviadas e recebidas até o momento. Essas três informações estão contidas em α.

Ao enviar uma mensagem $m_j \stackrel{def}{=} g(\alpha)$, o participante P_j precisa provar aos demais que (i) m_j foi calculada por g e, portanto, está de acordo com as regras de Π, e (ii) g foi aplicada à α para gerar m_j. Naturalmente que P_j não pode revelar α. Para isso, P_j invoca o oráculo para a funcionalidade f_{cma}. Note que o resultado da função h que autentica g é publicamente conhecido, consistindo das mensagem enviadas e recebidas no canal de *broadcast* e os comprometimentos recebidos de P_j nas etapas E2 e E3.

A utilização de f_{cma} na etapa E4 limita as ações do participante P_j, emissor da próxima mensagem em Π. Ele pode apenas se comportar honestamente ou suspender a execução. Não existe outra alternativa, pois se ele tentar enviar uma mensagem fora de ordem, malformada ou que tenha sido calculada sobre outra entrada ou fonte de aleatoriedade, os demais participantes vão detectar seu comportamento desonesto e suspender a execução. É por isso que podemos dizer que o compilador impõe o comportamento semi-honesto fortalecido aos participantes.

O teorema a seguir serve para preencher uma lacuna que deixamos até aqui. O compilador da Definição 4.88 restringe o comportamento dos participantes de um protocolo Π, tornando-o seguro contra adversários semi-honestos fortalecidos. No entanto, não é óbvio se isso é suficiente para tornar Π resistente a adversários maliciosos. Mas esse é exatamente o objetivo que queremos atingir. Registramos isso no Teorema 4.89.

Teorema 4.89 (Computação segura de uma funcionalidade qualquer)**.** *Sob a hipótese da existência de permutações arapuca aprimoradas, qualquer m-funcionalidade f pode ser seguramente computada na presença de um adversário malicioso e sobre um canal de comunicação ponto-a-ponto. Ressaltamos que f não é computada de maneira justa.*

Esboço de Prova. Sabemos, pelo Teorema 4.62, que existe um protocolo Π_{-1} canônico para computar *f privadamente*. Para computarmos f de maneira segura na presença de um adversário malicioso precisamos submeter Π_{-1} aos três compiladores apresentados:

1. *(Pré-compilador:)* Ao submeter Π_{-1} ao pré-compilador, obtemos Π_0 canônico que computa *f privadamente* sobre um canal de *broadcast*, de acordo com o Lema 4.65 (efeito do pré-compilador).

2. *(Compilador principal:)* Ao submeter o protocolo Π_0 ao compilador principal, da Definição 4.88, obtemos um protocolo Π_1 que executa sobre um canal de *broadcast* e cuja segurança é tratada neste teorema.

3. *(Pós-compilador:)* Ao submeter Π_1 ao pós-compilador obtemos o protocolo Π_2 que computa f sobre um canal ponto-a-ponto.

Segundo o Lema 4.69 (efeito do pós-compilador), a segurança de Π_1 é preservada em Π_2. Por isso precisamos apenas provar que o compilador principal produz protocolos seguros contra adversários maliciosos.

A ideia geral dessa prova se baseia nos fatos a seguir:

1. Todo adversário malicioso (real) A para Π_1 pode ser transformado em um adversário semi-honesto fortalecido (real) B para o protocolo Π_0. Tudo o que B precisa fazer é copiar o comportamento de A nas etapas de comprometimento com as entradas, geração de fonte de aleatoriedade e emulação do protocolo. Ressaltamos que B pode inclusive abortar.

2. Se existe um adversário semi-honesto fortalecido (real) para um protocolo Π *canônico*, então existe um adversário malicioso (ideal) para a funcionalidade computada por Π[12].

3. O protocolo Π_0 é canônico.

Por isso, se existe um adversário *malicioso* para o protocolo Π_1, então existe um adversário *semi-honesto fortalecido* equivalente[13] para Π_0 (item 1). Mas, se existe um adversário *semi-honesto fortalecido* para Π_0, como Π_0 é canônico (item 3), então existe um adversário *malicioso* ideal equivalente para a funcionalidade computada por Π_0, ou seja, f (item 3).

Dessa forma, concluímos que se existe um adversário malicioso real para Π_1, existe um adversário malicioso ideal equivalente para f. O que equivale a dizer que Π_1 computa f de maneira segura segundo a Definição 4.37 (segurança contra adversários maliciosos).

□

Por fim, ressaltamos que, ao submeter um procolo Π ao compilador, obtemos um protocolo auxiliado por (três) oráculos. No entanto, podemos utilizar os resultados sobre composição e redução, resumidos no Teorema 4.72, em conjunto com os protocolos para as funcionalidades que utilizamos no compilador (Π_{cma} da Definição 4.80, Π_{alea} da Definição 4.83 e Π_{comp} da Definição 4.86) para gerar um protocolo livre de oráculos. É necessário aplicar esse processo de composição iterativamente, já que alguns protocolos para essas funcionalidades são, eles próprios, auxiliados por oráculos.

[12] Proposição 7.4.27 de GOLDREICH (2004).

[13] Por "equivalente" queremos dizer que as visões conjuntas desses adversários são indistinguíveis, como na Definição 4.37 (segurança contra adversários malicioso).

4.4 Considerações Finais

Neste capítulo, seguimos a apresentação feita no Capítulo 7 (*General Cryptographic Protocols*) de GOLDREICH (2004) que trata da computação segura de uma funcionalidade qualquer.

Inicialmente, chegamos a um procedimento para computar uma funcionalidade qualquer em um modelo de segurança limitado (Seção 4.2). Nele, o adversário é semi-honesto e o canal de comunicação é ponto-a-ponto. A ideia básica desse procedimento é tratar a funcionalidade como um circuito lógico (ou uma série de operações sobre o F_2) e conduzir uma avaliação privada desse circuito, procedendo uma porta lógica por vez.

Um adversário semi-honesto pode modelar comportamentos reais e, por isso, esse primeiro resultado é de interesse por si só. No entanto, nós o utilizamos como etapa auxiliar para alcançar segurança no chamado modelo padrão, no qual não se faz restrições aos poderes do adversário. Partindo de um protocolo seguro contra um adversário semi-honesto aplicamos uma série de três procedimentos, ou compiladores, para gerar um protocolo seguro no modelo padrão.

Dois desses procedimentos, o pré-compilador e o pós-compilador, lidam apenas mudanças dos tipos de canais de comunicação. O resultado central da Seção 4.3, portanto é o compilador principal. Ele tem o papel de forçar o comportamento semi-honesto fortalecido ao adversário malicioso. Esse comportamento reflete o que pode e o que não pode ser esperado da segurança do protocolo final. Em particular, é impossível garantir que a execução não será abortada e, portanto, que o protocolo final é justo.

Por fim, resumimos o efeito dessa restrição de comportamento com o Teorema 4.89, o qual estabelece que qualquer funcionalidade pode ser computada de maneira segura na presença de um adversário malicioso.

5 COMPUTAÇÃO JUSTA

Começaremos este capítulo discutindo o resultado que estabelece a existência de funcionalidades para as quais não existem protocolos *justos*. Uma maneira de contornar esse resultado negativo, como discutiremos na Seção 5.2, é a adoção de definições *relaxadas* de justiça. Esses relaxamentos têm como objetivo possibilitar que protocolos, de forma geral, alcancem pelo menos *algum grau de justiça*.

Na Seção 5.3 nós definiremos a noção de justiça *monetária* que será adotada no restante deste trabalho. Ela se baseia no pagamento de multas como forma de compensar os participantes honestos e penalizar os desonestos quando a justiça *tradicional* de um protocolo é quebrada. A escolha dessa definição possibilita o emprego direto de transações *Bitcoin* na construção de protocolos justos.

5.1 Impossibilidade Geral

No Capítulo 4 apresentamos um importante resultado sobre a computação segura de uma funcionalidade qualquer. No entanto, esse resultado contém uma ressalva importante: a tolerância à quebra de justiça. Isso se deve ao fato de o modelo ideal, que é nosso referencial de segurança, dar ao adversário a possibilidade de privar os participantes honestos de receberem suas saídas[1]. Protocolos seguros nesse modelo (ou seja, seguros segundo a Definição 4.37) possuem o que se chama de segurança com (tolerância ao) aborto (*security with (agreement on) abort*), em oposição à segurança com justiça perfeita (*complete fairness*).

À primeira vista, a falta de justiça dos protocolos do Capítulo 4 poderia ser atribuída *apenas* à capacidade do adversário abortar. No entanto, essa não é uma análise precisa. É possível construir protocolos perfeitamente justos, mesmo na presença de um adversário com tal capacidade, desde que a *maioria dos participantes seja honesta*[2].

Esse resultado positivo, por sua vez, poderia sugerir que as construções utilizadas no Capítulo 4 não são ótimas e que trabalhos futuros poderiam generalizar a justiça perfeita para qualquer cenário. Infelizmente, esse não é o caso, como mostra CLEVE (1986). Esse trabalho demonstra que a funcionalidade de *cara ou coroa* (*coin flipping*), em particular, não pode ser computada com justiça perfeita, concluindo, portanto, que nem toda funcionalidade o pode.

Esse resultado é especialmente forte porque a funcionalidade de cara ou coroa, apesar de simples, é importante na criação de protocolos mais complexos. Nela, dois participantes têm como objetivo escolher um *bit* (ou jogar um moeda) com distribuição uniforme, isto é, com um viés desprezível. Ela pode ser definida formalmente como a seguir.

Definição 5.90 (Funcionalidade cara ou coroa)**.** *Sejam A e B dois participantes e n um parâmetro de segurança. Definimos o viés de um bit b como $viés(b) \stackrel{def}{=} \left|Pr[b=0] - \frac{1}{2}\right|$. A 2-funcionalidade*

[1] O modelo ideal de execução, no caso de adversários maliciosos, é apresentado na Definição 4.34.

[2] Uma forma de construir tais protocolos é descrita por GOLDREICH (2004).

de cara ou coroa, $f_{c/c}$, é definida como a seguir:

$$f_{c/c}(1^n, 1^n) \stackrel{def}{=} (b_A, b_B),$$

com $b_A, b_B \in \{0,1\}$, $Pr[b_A = b_B] = 1$ (ou seja, $b_A = b_B$) e $viés(b_A), viés(b_B)$ desprezíveis em n (ou seja, $viés(b_A), viés(b_B) \leq O\left(\frac{1}{n^k}\right)$ para todo k).

Na verdade, CLEVE (1986) considera uma versão relaxada de $f_{c/c}$, ou seja, uma versão que não pode ser mais difícil que a própria $f_{c/c}$. De maneira geral, provar que a *versão relaxada* de $f_{c/c}$ não admite um protocolo perfeitamente justo fortalece ainda mais seu resultado negativo.

Nessa versão relaxada, as saídas b_A e b_B só precisam ser iguais com probabilidade maior ou igual a 50%. Ou seja, a restrição "$Pr[b_A = b_B] = 1$" da Definição 5.90 é substituída pela restrição "$Pr[b_A = b_B] \geq \frac{1}{2} + \varepsilon$", para algum ε predefinido. Note que isso torna a definição significativamente mais fraca. Em particular, qualquer implementação para $f_{c/c}$ serve também como implementação para sua versão relaxada, mas o contrário não é verdadeiro. Definimos formalmente esse relaxamento a seguir.

Definição 5.91 (Funcionalidade cara ou coroa relaxada)**.** *Sejam A e B dois participantes, n um parâmetro de segurança e $\varepsilon \in [0, \frac{1}{2}]$ um valor predefinido. Definimos a versão relaxada de $f_{c/c}$ como a 2-funcionalidade $f^{\varepsilon}_{c/c}$ tal que:*

$$f^{\varepsilon}_{c/c}(1^n, 1^n) \stackrel{def}{=} (b_A, b_B),$$

com $b_A, b_B \in \{0,1\}$, $\mathbf{Pr}[\mathbf{b_A} = \mathbf{b_B}] \geq \mathbf{\frac{1}{2}} + \varepsilon$ e $viés(b_A), viés(b_B)$ desprezíveis em n.

Um protocolo *seguro* para computar $f^{\varepsilon}_{c/c}$ tem que, em particular, garantir que a saída de um participante honesto tenha sempre um viés desprezível. E é isso que impede essa funcionalidade de ser computada com justiça perfeita. Registramos essa impossibilidade no teorema a seguir.

Teorema 5.92 (Impossibilidade de CLEVE (1986))**.** *Não é possível computar $f^{\varepsilon}_{c/c}$ com justiça perfeita.*

Demonstração. A prova formal deste teorema pode ser encontrada no Apêndice A. Apresentaremos aqui apenas uma argumentação informal.

A ideia geral dessa prova é considerar um protocolo Π qualquer para computar $f^{\varepsilon}_{c/c}$ em $r(n)$ rodadas, em que n é um parâmetro de segurança. A partir daí, cria-se $2r(n)+1$ estratégias maliciosas para o participante A e $2r(n)$ para o participante B. A idéia dessas estratégias é emular o comportamento honesto do participante correspondente até a rodada i, com $i \in \{1, \cdots, r(n)\}$, e então abortar.

Consideramos, em seguida, os protocolos resultantes da substituição de um participante honesto por cada uma das $4r(n)+1$ estratégias desonestas. Com isso, podemos calcular a média Δ dos vieses causados nas saídas dos participantes honestos por cada protocolo resultante. Pode

ser demonstrado que $\Delta \geq \frac{\varepsilon}{4r(n)+1}$. Segue então que pelo menos uma das parcelas da média tem valor maior ou igual a Δ. Isso significa que pelo menos um dos adversários considerados causa um viés de valor *pelo menos* Δ à saída do participante honesto. Como Δ não é desprezível, concluímos que Π não é seguro. □

Como vimos, a prova do Teorema 5.92 envolve criar um adversário que utiliza sua capacidade de abortar para enviesar a saída do participante honesto de forma significativa. Isso mostra que a única maneira de se computar $f_{c/c}^{\varepsilon}$ de forma segura é com tolerância ao aborto. Também em CLEVE (1986) é feita uma generalização da funcionalidade $f_{c/c}$ para mais de dois participantes. Nesse caso, o resultado negativo é de que ela não pode ser computada com justiça perfeita se pelo menos metade dos participantes forem desonestos. Não apresentamos esse resultado porque o Teorema 5.92 já é suficiente para nossa discussão.

Corolário 5.93 (Consequência da impossibilidade de CLEVE (1986))**.** *Nem toda funcionalidade pode ser computada com justiça perfeita.*

A consequência do Teorema 5.92, registrada no Corolário 5.93, é que nem toda funcionalidade pode ser computada por um protocolo perfeitamente justo. Portanto, outra consequência, é a inexistência de uma técnica geral para a criação de protocolos justos.

Além da funcionalidade de cara ou coroa (em suas versões relaxada, com dois participantes ou com vários), outras funcionalidades notáveis também não podem ser computadas com justiça perfeita. É o caso, por exemplo, da funcionalidade para o cálculo do ou-exclusivo (*XOR*). De fato, por algum tempo, acreditou-se que *nenhuma* funcionalidade não-trivial pudesse ser computada com justiça perfeita. No entanto, trabalhos recentes, como os de ASHAROV (2014); GORDON et al. (2011), mostraram que essa crença não é verdadeira e muitas funcionalidades admitem implementações perfeitamente justas.

5.2 Definições Relaxadas de Justiça

Na prática, a justiça é uma característica tão importante quanto a própria segurança. Ela garante, essencialmente, que nenhum participante honesto vai desperdiçar recursos computacionais participando de um protocolo sem receber uma resposta adequada no final. Por isso, e diante da impossibilidade estabelecida pelo Teorema 5.92, definições mais fracas de justiça são consideradas para que ao menos *algum* grau de justiça possa ser alcançado em geral.

Apresentamos algumas a seguir.

- Justiça através da liberação gradual (*gradual release*) - Na ocorrência de um aborto, os participantes podem investir uma quantidade comparável de recursos computacionais para obter suas saídas. Em geral, protocolos justos sob essa definição são conduzidos em duas etapas. A primeira delas é onde ocorre a computação propriamente dita e a segunda é onde os participantes reconstroem a saída *bit*-a-*bit*. Desse modo, quando um participante aborta prematuramente consegue apenas uma pequena vantagem em

relação aos demais. Exemplos dessa técnica podem ser encontrados nos trabalhos de GOLDWASSER e LEVIN (1990) e BONEH e NAOR (2000).

- Justiça no modelo otimista (*optimistic model*) - Na ocorrência de um aborto, a justiça pode ser restabelecida por uma TPC. O modelo resultante da adição dessa TPC é chamado *otimista* porque a TPC precisa ser contactada *apenas* no caso de quebra de justiça. Esse tipo de modelo é provado ser necessário, por exemplo, para a condução de trocas justas (*fair exchange*), como mostrado por PAGNIA e GÄRTNER (1999).

- Justiça com sistemas de reputação - Nesse modelo, todo participante possui uma *reputação*, que pode ser encarada como sua probabilidade de agir honestamente. Ações desonestas nos protocolos são, portanto, punidas com decréscimos de reputação. Esse tipo de justiça é estudado por ASHAROV; LINDELL; ZAROSIM (2013).

- Justiça monetária - Nessa definição de justiça, todo participante que aborta paga uma *multa* aos participantes honestos. Dessa forma, quando a justiça é quebrada, diferentemente do que ocorre no modelo otimista, por exemplo, os participantes honestos não recuperam suas saídas, mas recebem uma compensação monetária. Esse é a definição de justiça que usaremos neste trabalho. Ela também é usada nos trabalhos de BENTOV e KUMARESAN (2014), ANDRYCHOWICZ et al. (2014b), ANDRYCHOWICZ et al. (2014a) e KÜPÇÜ e LYSYANSKAYA (2012)

5.3 Definição Adotada de Justiça

Como mencionamos anteriormente, a noção de justiça adotada no decorrer deste trabalho estabelece que participantes desonestos sejam *monetariamente* penalizados por abortarem. Claro que, para chegar a uma formulação mais precisa, temos antes que definir o conceito de *moeda*.

Definição 5.94 (Moedas, adaptado de BENTOV e KUMARESAN (2014))**.** *Definimos moedas como entidades atômicas, intercambiáveis, mas distinguíveis entre si e que não podem ser duplicadas. Assumimos que somente o dono de uma moeda pode transferí-la ou gastá-la e que uma moeda não pode ter mais de um dono simultaneamente. Também assumimos que a validade de uma moeda pode ser prontamente verificada. Cada moeda possui um valor unitário. Para todo natural n denotamos $valor(n)$ como o conjunto contendo n moedas.*

O objetivo da Definição 5.94 é capturar as características de uma moeda real e, principalmente, permitir a utilização direta do *Bitcoin* em nossas construções.

Formalizamos nossa definição de justiça a seguir.

Definição 5.95 (Justiça monetária, adaptado de BENTOV e KUMARESAN (2014))**.** *Seja Π um protocolo (que pode ser o trivial) para computar uma m-funcionalidade f. Seja c_i o investimento monetário feito pelo participante P_i durante o protocolo e r_i o valor recebido por P_i após o término da execução. O protocolo Π é considerado monetariamente justo se:*

- *Um participante P_i honesto nunca paga qualquer penalidade ($r_i \geq c_i$),*
- *Se um participante P_i aborta após receber sua saída e priva os demais de suas saídas, ou seja, se P_i quebra a justiça tradicional, então P_i é penalizado ($r_i < c_i$) e todo participante honesto é compensado ($r_h > c_h$, para todo P_h honesto).*

Nos protocolos que apresentaremos, justos de acordo com a Definição 5.95, cada participante é obrigado a fazer um *investimento monetário inicial*. No final da computação, cada participante recebe uma quantidade de moedas definida de acordo com seu comportamento. Se ele tiver agido honestamente, receberá de volta pelo menos as moedas que gastou. Além disso, caso a justiça tradicional tenha sido quebrada, ele receberá também uma compensação. De modo geral, o *saldo* de um participante honesto no fim do protocolo é ≥ 0 e do desonesto ≤ 0.

Para tornar a Definição 5.95 o mais geral possível, não especificamos o valor do investimento inicial nem o recebido ao fim da computação. Veremos que esses valores dependem, em geral, de um parâmetro preestabelecido q, que representa a multa por aborto. No entanto, cada protocolo pode definir uma estratégia própria de pagamento, como veremos no capítulo a seguir.

6 PROTOCOLOS JUSTOS COM *BITCOIN*

Adotamos a definição de justiça monetária, discutida no Capítulo 5, com o objetivo de utilizar o *Bitcoin* para a condução de computações justas. Neste capítulo exploraremos como isso será feito.

Começaremos discutindo o modelo de computação no qual as funcionalidades deste capítulo serão definidas. Precisamos adotar um modelo diferente do utilizado no Capítulo 4 principalmente porque necessitamos de mais expressividade para especificar as funcionalidades. Além disso, também precisamos de modificações para lidar com as funcionalidades *especiais*, as que manipulam moedas.

O ponto central deste capítulo é mostrar como o problema de se conduzir uma computação segura e justa se reduz ao problema da reconstrução justa de um segredo dividido. Em linhas gerais, para computar uma funcionalidade qualquer de maneira justa, vamos antes dividir sua saída entre os participantes de modo que ela se mantenha secreta. Em seguida, vamos conduzir uma computação multilateral para reconstruir essa saída de maneira monetariamente justa. Discutiremos essa ideia na Seção 6.2.

Tendo ressaltado a importância do problema da reconstrução justa, apresentaremos, nos próximos capítulos, soluções para ele utilizando o *Bitcoin*.

6.1 Modelo de Computação

No Capítulo 4, o modelo ideal de computação nos permitiu especificar funcionalidades de forma concisa, além de facilitar a análise de segurança dos protocolos. No entanto, esse modelo impõe algumas limitações. Para o restante do nosso trabalho, a principal delas é a falta de expressividade para lidar com funcionalidades *reativas*. Isso se deve ao fato da TPC só interagir com os demais componentes da computação em apenas duas ocasiões: ao receber entradas e ao entregar saídas. Esse tipo de interação é suficiente quando queremos computar *funções* de maneira segura. Para expressar computações mais complexas, no entanto, é necessário considerar interações também mais complexas com a TPC.

Neste capítulo, utilizaremos o modelo de computação de BENTOV e KUMARESAN (2014) que, por sua vez, é baseado no *framework* para *segurança universalmente componível* de CANETTI (2001). Em linhas gerais, esse *framework* tem como objetivo estabelecer um conceito fortalecido e abrangente de segurança, considerando, em especial, o cenário em que vários protocolos são executados simultaneamente em um mesmo ambiente. De fato, CANETTI (2001) define a operação de *composição universal* que é um caso geral das composições mais comuns, como a paralela, a concorrente e a sequencial. Essa operação garante que os protocolos tenham sua segurança preservada mesmo quando compostos com protocolos arbitrários e em situações não previstas. Ela também possibilita a modularização tanto do projeto quanto da análise dos protocolos.

No modelo de computação adotado, assim como em CANETTI (2001), o ambiente de execução é representado por uma entidade algorítmica $\mathscr{E}$. Ela é a responsável por fornecer as entradas dos participantes e, ao longo da execução, pode interagir livremente com o adversário ideal S (ou real A).

Seguindo BENTOV e KUMARESAN (2014), assumimos a existência de canais ponto-a-ponto seguros e síncronos. Isso difere do modelo de comunicação extremamente simples utilizado em CANETTI (2001), o qual é chamado de "modelo despido" (*bare model*). No entanto, CANETTI (2001) discute como outros canais podem ser construídos sobre esse modelo mais simples.

De maneira semelhante ao que fizemos no Capítulo 4, CANETTI (2001) define a execução ideal de uma funcionalidade f como sua execução por uma TPC. A diferença é que a TPC pode interagir livremente com os participantes e com o adversário durante a execução. Isso se dá através da troca de mensagens, do recebimento de entradas e do envio de saídas. O comportamento da TPC pode depender das interações anteriores e do estado corrente do protocolo. Por isso a TPC pode guardar informações a respeito do estado da execução. Na execução ideal não há interação entre os participantes.

Já na execução real, os participantes interagem entre si, cada qual com sua estratégia, para emular o funcionamento da TPC.

A segurança *universalmente componível* se baseia na capacidade do ambiente $\mathscr{E}$ distinguir entre as execuções ideal e real. Registramos essa definição a seguir.

Definição 6.96 (Segurança universalmente componível)**.** *Seja f uma funcionalidade e Π um protocolo que computa f. Dizemos que Π computa f com segurança universalmente componível, ou simplesmente que Π* **uc-realiza** *f se, para todo adversário A não uniforme de tempo polinomial probabilístico (no parâmetro de segurança λ) atacando Π, existe um adversário S não uniforme de tempo polinomial probabilístico atacando f tal que:*

$$\{IDEAL_{f,S,\mathscr{E}}(\lambda,z)\}_{\lambda\in N,z\in\{0,1\}^*} \stackrel{c}{\equiv} \{REAL_{\Pi,A,\mathscr{E}}(\lambda,z)\}_{\lambda\in N,z\in\{0,1\}^*}$$

em que $IDEAL_{f,S,\mathscr{E}}(\lambda,z)$ é a saída de $\mathscr{E}$ na execução ideal com parâmetro de segurança λ e entrada z. O significado de $REAL_{\Pi,A,\mathscr{E}}(\lambda,z)$ é semelhante.

Outro conceito importante definido em CANETTI (2001) é o de *execução híbrida*, análogo ao de *execução com auxílio oracular* que definimos no Capítulo 4. Dizemos que um protocolo Π_f para computar f é g-híbrido se, em sua execução, são feitas chamadas à um oráculo para a funcionalidade g. Se Π_g é um protocolo para computar g, então a operação de composição universal utiliza Π_f e Π_g para construir um protocolo Π'_f que computa f sem usar o oráculo para g. Além disso, se Π_f e Π_g possuem segurança universalmente componível, então o teorema da composição universal garante que Π'_f também possui segurança universalmente componível.

Até agora nós citamos características do *framework* para segurança universalmente componível que são relevantes para o restante do nosso trabalho. No entanto, esse é um

framework muito rico e poderoso, possuindo vários detalhes que serão omitidos. Ele permite, por exemplo, a modelagem de adversários adaptativos (que corrompem participantes durante a execução) e de protocolos com número variável de participantes. Além de conferir um alto grau de segurança a seus protocolos, como já mencionamos.

Além das definições básicas retiradas de CANETTI (2001), no nosso modelo de computação, seguindo a apresentação de BENTOV e KUMARESAN (2014), cada participante possui duas estruturas auxiliares para armazenamento de moedas: a *carteira* e o *cofre*. Para iniciar um protocolo, cada participante deve transferir de sua carteira para seu cofre a quantidade necessária de moedas. O comportamento honesto de um participante que não tem moedas suficientes em sua carteira para fazer esse investimento inicial é simplesmente não tentar participar do protocolo.

Assumimos que apenas o ambiente $\mathscr{E}$ pode alterar o conteúdo das carteiras dos participantes honestos. Os cofres desses participantes, no entanto, estão fora do alcance de $\mathscr{E}$. Durante a execução de um protocolo, possíveis créditos e débitos de moedas são feitos nos cofres dos participantes. No final do protocolo, cada participante transfere suas moedas do cofre de volta para a carteira. Por fim, assumimos que o adversário tem total controle tanto sobre a carteira quanto sobre o cofre dos participantes desonestos.

Chamaremos uma funcionalidade de *especial* se ela lida com moedas. Caso contrário a chamaremos de tradicional. Quando não houver ambiguidade, chamaremos apenas de funcionalidade. Usaremos a mesma definição de segurança universalmente componível (Definição 6.96) em ambos os casos.

Como observação final, ressaltamos que a principal motivação para utilizarmos um modelo de computação semelhante ao apresentado em CANETTI (2001) é a sua expressividade e modularidade. Nós esperamos usufruir de suas garantias de segurança, mas os resultados em CANETTI (2001) não cobrem, naturalmente, os conceitos e estruturas auxiliares com os quais aumentamos o modelo, como *moedas*, *cofres* e *carteiras*. Por isso, embora os protocolos que exibiremos sejam seguros de acordo com a Definição 6.96 e possuam características importantes como a interação *on-line* com o ambiente, não podemos, a princípio, utilizar diretamente os resultados de segurança e composição de CANETTI (2001).

6.2 Computação Justa Através da Reconstrução Justa

6.2.1 Visão Geral

Nesta seção vamos mostrar como a computação segura e monetariamente justa de uma funcionalidade f se reduz à reconstrução justa de um *segredo dividido*. A apresentação dessa redução é baseada naquela que foi feita em BENTOV e KUMARESAN (2014). Um resultado semelhante, para o caso particular em que f é uma 2-funcionalidade, também é discutido em ANDRYCHOWICZ et al. (2014a).

A estratégia geral dessa redução é computar f em duas etapas. Na primeira delas, os participantes se engajam na computação segura (com tolerância ao aborto) de uma *versão*

estendida de f, denotada por $\hat{f}$. A funcionalidade $\hat{f}$ basicamente calcula a saída z de f e a divide em m pedaços, um para cada participante, utilizando um (m,m)-esquema verificável de compartilhamento de segredo. Chamaremos z de *segredo dividido* porque cada participante conhece apenas um *pedaço* desse valor.

Na segunda etapa da computação de f, os participantes se engajam na reconstrução (monetariamente) justa de z. Isto é, eles cooperam para que, no final da execução, cada um possua todos os pedaços de z e seja, portanto, capaz de reconstruir esse valor.

O resultado estabelecido pelo Teorema 4.89 nos garante que $\hat{f}$ pode, de fato, ser computada de maneira segura. Por essa razão, a computação segura e justa de f se reduz à reconstrução justa de um segredo dividido (z).

6.2.2 Definições Preliminares

Como mencionamos introdutoriamente, o restante da nossa apresentação se baseará fortemente no uso de um (m,m)-esquema de compartilhamento de segredo. Por isso, vamos definir uma implementação concreta de tal esquema, complementando a definição puramente teórica que demos na Definição 3.19.

Ressaltamos, inicialmente, que o esquema da Definição 3.19 trata do compartilhamento de um único *bit*. Estaremos interessados, no entanto, em compartilhar (ou dividir) cadeias binárias de tamanho arbitrário. Isso é tratado de maneira natural, como no caso dos esquemas de comprometimento. Ou seja, a divisão de uma cadeia é a sequência formada pelas divisões de cada um de seus *bits*.

A Definição 3.19 utiliza um esquema de compartilhamento de segredo (Definição 3.18) de maneira auxiliar. Não vamos implementar esse esquema diretamente, mas vamos utilizar uma ideia similar à da divisão de um *bit*, da Definição 4.46.

Por fim, a Definição 3.19 utiliza um esquema de comprometimento para alcançar *verificabilidade*. Adicionamos a restrição de que esse esquema seja *equívoco*.

Definimos a implementação de um (m,m)-*esquema verificável de compartilhamento de segredo* ($EVCS_{m,m}$) a seguir.

Definição 6.97 (Implementação de um (m,m)-esquema verificável de compartilhamento de segredo, adaptado de BENTOV e KUMARESAN (2014))**.** *Seja C um esquema de comprometimento equívoco. Definimos a implementação do trio de algoritmos $(Divide, Reconstrói, Verifica)$ como a seguir:*

- *Para todo $z \in \{0,1\}^*$, $Divide(z) \stackrel{def}{=} ((e_1,(p_1,w_1)),\cdots,(e_m,(p_m,w_m)))$ tais que:*
  - *Para todo $i \in \{1,\cdots,m\}$, w_i é escolhido uniformemente do conjunto $\{0,1\}^{|z|}$,*
  - *Para todo $i \in \{1,\cdots,m-1\}$, p_i é escolhido uniformemente do conjunto $\{0,1\}^{|z|}$,*
  - *$p_m = \left(z \oplus \bigoplus_{i=1}^{m-1} p_i\right)$. Desse modo $\bigoplus_{i=1}^{m} p_i = z$.*

- *Para todo* $i \in \{1,\cdots,m\}$, $e_i = C_{w_i}(p_i)$.

- *(Nomenclatura:) Definimos a lista* $Etiquetas \stackrel{def}{=} \{e_1,\cdots,e_m\}$ *e a lista* $Pedaços \stackrel{def}{=} \{(p_1,w_1),\cdots,(p_m,w_m)\}$. *O i-ésimo elemento de Etiquetas será denotado por* $Etiquetas_i$. *Da mesma forma, o i-ésimo elemento de Pedaços será denotado por* $Pedaços_i$. *Cada um dos elementos de Pedaços será chamado de "pedaço (de z)".*

- $Reconstrói(Etiquetas, Pedaços) \stackrel{def}{=} \begin{cases} \bigoplus_{i=1}^{m} p_i & com \quad (p_i,w_i) = Pedaços_i \quad e \quad se \\ & Verifica(Etiquetas_i, Pedaços_i) = 1 \\ & para\ todo\ i \in \{1,\cdots,m\}, \\ \perp & caso\ contrário. \end{cases}$

- $Verifica(E, P=(p,w)) \stackrel{def}{=} \begin{cases} 1 & se\ C_w(p) = E, \\ 0 & caso\ contrário. \end{cases}$

Um $EVCS_{m,m}$ tem dois objetivos principais: dividir um valor z em m pedaços e reconstruir z a partir desses pedaços. A divisão de z, feita pelo algoritmo *Divide*, gera as listas *Etiquetas* e *Pedaços*. Os elementos de *Etiquetas* conferem a *não-maleabilidade* aos elementos correspondentes em *Pedaços*. De fato, como $Etiquetas_i$ é o resultado de se comprometer com $Pedaços_i$, não é possível criar um P tal que $P \neq Pedaços_i$ e o par $(Etiquetas_i, P)$ seja válido $(Verifica(Etiquetas_i, P) = 1)$[1].

Além da não-maleabilidade, a divisão de z gerada por um $EVCS_{m,m}$ possui outra propriedade importante. Como os elementos de *Pedaços* têm distribuição uniforme, nenhum subconjunto próprio deles revela qualquer informação sobre z. Como consequência, o valor z só pode ser reconstruído a partir de toda a lista *Pedaços*, através do algoritmo *Reconstrói*.

A reconstrução de z a partir de seus pedaços é feita de maneira eficiente, bastando, para isso, serem "somados". Nesse caso, a "soma" é a operação de ou-exclusivo.

Como era de se esperar, um $EVCS_{m,m}$ será utilizado pela versão estendida $\hat{f}$ de uma funcionalidade para *dividir* a saída de f entre seus participantes. Registramos como isso é feito na definição a seguir.

Definição 6.98 (Versão estendida de uma funcionalidade)**.** *Seja* f *uma m-funcionalidade e* $(Divide, Reconstrói, Verifica)$ *um* $EVCS_{m,m}$ *como na Definição 6.97. Supomos, por simplicidade, que todos os participantes da computação de* f *recebem a mesma saída. Definimos a versão estendida de* f *como a m-funcionalidade* $\hat{f}$ *tal que, para toda sequência de entrada* $\hat{x} \in (\{0,1\}^*)^m$*:*

$$\hat{f}(\hat{x}) = ((Etiquetas, Pedaços_1), (Etiquetas, Pedaços_2), \cdots, (Etiquetas, Pedaços_m))$$

em que $Divide(m, f(\hat{x})) = (Etiquetas, Pedaços)$.

[1] Pelo menos não com uma chance não desprezível.

A versão estendida de uma funcionalidade f, pela Definição 6.98, divide a saída de f em m pedaços e os distribui de modo que *apenas* o participante P_i conhece $Pedaços_i$. Além disso, ela publica a lista *Etiquetas*. Isso é importante porque, como antecipamos, o nosso objetivo é que os participantes, no final de um protocolo para reconstrução justa, obtenham todos os elementos de *Pedaços*. A lista *Etiquetas* é, portanto, a forma pela qual eles podem verificar a validade dos pedaços recebidos.

6.2.3 Juntando as Peças: a Redução

Com as definições de $EVCS_{m,m}$ e da versão estendida de uma funcionalidade, a redução da computação segura e justa à reconstrução justa de um segredo dividido torna-se intuitiva.

Como antecipamos de maneira introdutória, vamos dividir a computação segura e justa de uma funcionalidade f em duas etapas, de acordo com a definição a seguir.

Definição 6.99 (Computação segura e justa através da reconstrução justa)**.** *Seja f uma m-funcionalidade (tradicional) e z a saída de f para seus participantes quando computada sobre uma entrada qualquer. A computação segura e justa de f pode ser conduzida nas duas etapas a seguir:*

***E1**. (Divisão de z em pedaços.) Os m participantes se engajam na computação segura (com tolerância ao aborto) de $\hat{f}$. Ao final desta etapa, de acordo com a Definição 6.98, para todo $i \in \{1, \cdots, m\}$, P_i recebe $(Etiquetas, Pedaços_i)$.*

***E2**. (Reconstrução de z.) Os m participantes se engajam em uma computação multilateral monetariamente justa para reconstrução de z. A entrada de cada participante nessa computação é seu pedaço de z recebido na etapa **E1**. Como a computação deve ser monetariamente justa, os participantes lidarão com moedas nesta etapa.*

Como $\hat{f}$ é uma funcionalidade *tradicional*, a etapa **E1** da Definição 6.99 pode ser conduzida de acordo com o resultado estabelecido pelo Teorema 4.89. Ressaltamos que, nessa etapa, todo aborto é considerado *prematuro*, isto é, não pode causar a quebra da justiça (tradicional) de f. De fato, um aborto na etapa **E1** pode apenas quebrar a justiça de $\hat{f}$, fazendo com que alguns participantes recebam seus pedaços de z e outros não. No entanto, devido às propriedades do $EVCS_{m,m}$ utilizado em $\hat{f}$, esses pedaços não são suficientes para reconstruir a saída z. Portanto, a justiça de f fica trivialmente protegida porque, ao final de **E1**, nenhum subconjunto de participantes pode reconstruir z.

Por outro lado, um aborto na etapa **E2** pode, de fato, quebrar a justiça tradicional de f. Por isso ela precisa ser conduzida de maneira *monetariamente justa*.

A Definição 6.99 estabelece a redução que nos propomos a apresentar: uma funcionalidade f pode ser computada de maneira segura e justa desde que a etapa **E2** também o possa. Por isso, tudo o que precisamos fazer para computar f de maneira justa é resolver o problema da

reconstrução justa de um segredo. Apresentaremos, nos próximos capítulos, duas estratégias de resolução deste problema tendo o *Bitcoin* como plataforma.

7 RECONSTRUÇÃO JUSTA ATRAVÉS DA CONSTRUÇÃO ESCADA

Como discutimos no Capítulo 6, a computação justa de uma funcionalidade f qualquer se reduz a reconstrução justa de um segredo dividido. Neste capítulo apresentaremos uma estratégia para condução de reconstrução justa, a chamada *construção escada*. Discutiremos também sua correção e segurança e apresentaremos uma implementação baseada no *Bitcoin*.

7.1 Visão Geral

Como discutido no capítulo anterior, na etapa **E2** de reconstrução da Definição 6.99, cada participante P_i de uma m-funcionalidade f possui o elemento $Pedaços_i$ e a lista *Etiquetas*, resultantes da aplicação do algoritmo *Divide* de um $EVCS_{m,m}$ à saída de f. O objetivo dessa etapa é que cada participante consiga toda a lista *Pedaços*.

A construção escada é a proposta para reconstrução justa apresentada em BENTOV e KUMARESAN (2014). A unidade básica desse protocolo é a funcionalidade especial f_{cr} para ***c**rédito-ou-**r**eembolso (claim-or-refund)*. Ela viabiliza o depósito, ou transferência, *condicional* de moedas entre dois participantes. Para *reivindicar* o depósito, o destinatário deve fornecer uma entrada que satisfaça um circuito lógico φ especificado pelo remetente. Nesse caso dizemos que o depósito foi *creditado* ou *reivindicado*. Se isso não acontecer antes de uma rodada τ, especificada pelo remetente, o depósito é *reembolsado* e o remetente recebe suas moedas de volta.

Representamos graficamente uma chamada à f_{cr} para realizar uma transferência de c moedas de P_i para P_j, com uma rodada limite τ e um circuito φ como:

$$\left[P_i \xrightarrow[c,\tau]{\varphi} P_j\right].$$

A construção escada faz diversas chamadas à funcionalidade f_{cr}, divididas em duas etapas: a dos *depósitos-escada (ladder deposits)* e a dos *depósitos-cobertura (roof deposits)*. Se m é a quantidade total de participantes, então $m-1$ depósitos-escada e $m-1$ depósitos-cobertura são realizados.

Nos depósitos-escada, para todo $i \in \{1,\cdots,m-1\}$, o participante P_{i+1} utiliza f_{cr} para transferir $i \cdot q$ moedas para P_i sob a condição de P_i satisfazer o circuito $\varphi_{i+1,i}$ com o conjunto $\{Pedaços_1,\cdots,Pedaços_i\}$ até a rodada τ_i. O valor q é uma multa por aborto preestabelecida. O objetivo desses depósitos é possibilitar a revelação gradual dos elementos de *Pedaços* de modo que, no final da rodada τ_{m-1}, o participante P_m possua todos esses elementos.

Os depósitos-cobertura, por sua vez, devem ser creditados na rodada τ_m. Neles, para todo $i \in \{1,\cdots,m-1\}$, o participante P_i utiliza f_{cr} para transferir q moedas para P_m sob a condição do circuito φ_m ser satisfeito. Esse circuito, por sua vez, só é satisfeito pelo conjunto de *todos* os elementos de *Pedaços*. Por essa razão, quando P_m reivindica esses depósitos, todos os

participantes recebem a lista *Pedaços* completa e conseguem, portanto, reconstruir a saída. Por outro lado, se P_m não reivindicá-los, nenhum participante (honesto) consegue reconstruí-la.

De maneira semelhante ao que já havíamos definido, representamos graficamente uma chamada à f_{cr} em que P_i transfere q moedas para P_j sob a condição de P_j apresentar o conjunto $\{Pedaços_1, \cdots, Pedaços_j\}$ até a rodada τ como

$$\left[P_i \xrightarrow[q,\tau]{Pedaços_1,\cdots,Pedaços_j} P_j\right],$$

e usamos essa notação para esquematizar os depósitos escada e cobertura na Figura 7.1. Nela, os depósitos são enumerados de acordo com a ordem em que são feitos. A ordem dos depósitos-cobertura, no entanto, não é muito importante.

Como esquematizamos na Figura 7.1, cada depósito-escada tem a forma $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{Pedaços_1,\cdots,Pedaços_i} P_i\right]$. Para reivindicá-lo, o participante P_i precisa (i) do seu próprio pedaço ($Pedaços_i$) e (ii) do conjunto $\{Pedaços_1, \cdots, Pedaços_{i-1}\}$. A única maneira honesta de obter (ii) é através do depósito-escada imediatamente "abaixo", isto é, $\left[P_i \xrightarrow[(i-1)\cdot q,\tau_{i-1}]{Pedaços_1,\cdots,Pedaços_{i-1}} P_{i-1}\right]$. Por essa razão, um depósito-escada só pode ser creditado quando o logo abaixo dele também o for. Aplicando essa dependência repetidamente, concluímos que um depósito-escada só pode ser creditado quando *todos* os abaixo dele também o forem.

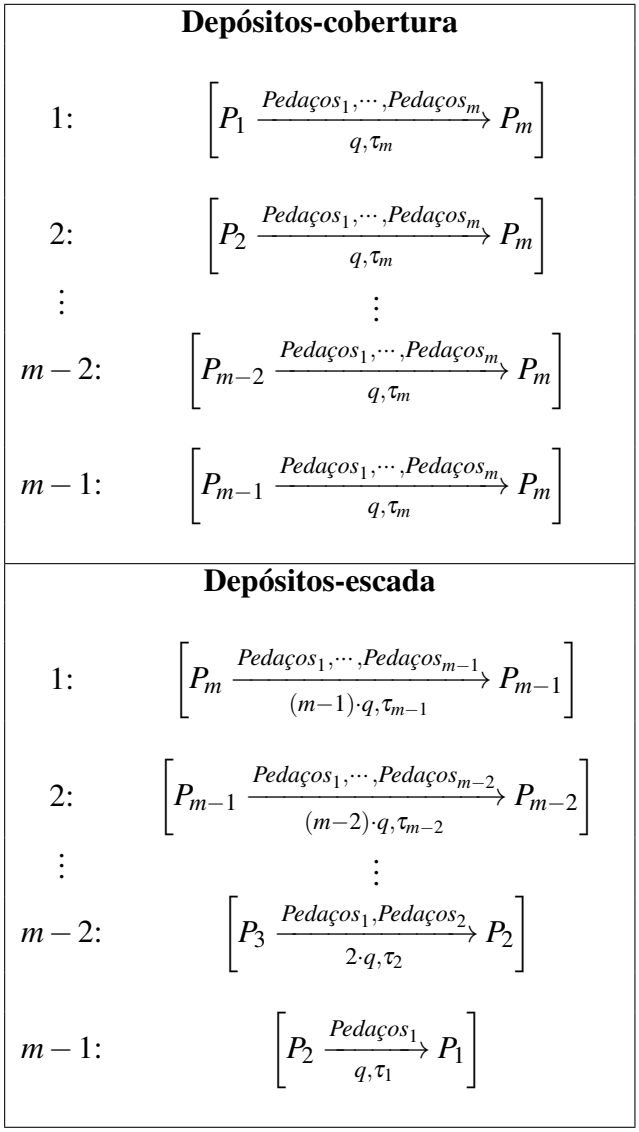
Quando P_i reivindica $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{Pedaços_1,\cdots,Pedaços_i} P_i\right]$, ele fica temporariamente com um saldo positivo de q moedas[1]. Se a reconstrução continuar de maneira honesta, esse saldo é perdido no depósito-cobertura $\left[P_i \xrightarrow[q,\tau_m]{Pedaços_1,\cdots,Pedaços_m} P_m\right]$ e P_i termina a execução com saldo zerado. No entanto, se algum depósito "acima" de $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{Pedaços_1,\cdots,Pedaços_i} P_i\right]$ não for creditado, significando que houve um aborto, então P_i termina a execução com esse mesmo saldo positivo, pois os depósitos-cobertura são reembolsados. Portanto, a noção de justiça da construção escada é compensar, no caso da ocorrência de um aborto, todos os participantes que publicaram seu pedaço do segredo através da reivindicação de um depósito-escada. Isso acontece mesmo que o adversário não tenha conseguido reconstruir a saída e, portanto, não tenha quebrado a justiça tradicional.

Outro ponto relevante da construção escada é que, se P_m for honesto, ou todos recebem o valor reconstruído ou ninguém recebe. Nesse caso, a decisão do adversário de abortar ou não é *independente* da saída do protocolo.

---

[1]Justificativa: se $i = 1$, P_1 recebe q moedas no depósito $\left[P_2 \xrightarrow[q,\tau_1]{Pedaços_1} P_1\right]$ e não tem qualquer gasto. Se $i \neq 1$, P_i gasta $(i-1)$ moedas no depósito $\left[P_i \xrightarrow[(i-1)\cdot q,\tau_{i-1}]{Pedaços_1,\cdots,Pedaços_{i-1}} P_{i-1}\right]$ e recebe $i \cdot q$ moedas no depósito $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{Pedaços_1,\cdots,Pedaços_i} P_i\right]$. Nesse caso, seu saldo é $i \cdot q - (i-1) \cdot q = q$.

Figura 7.1: Os depósitos escada e cobertura feitos na construção escada



Fonte: Elaborada pelo autor.

7.2 A funcionalidade f_{cr}

Agora que temos a intuição de como f_{cr} funciona, vamos definí-la formalmente a seguir.

Definição 7.100 (Funcionalidade f_{cr} para crédito-ou-reembolso - BENTOV e KUMARESAN (2014))**.** *Seja $\{P_1,\cdots,P_m\}$ um conjunto de participantes. A funcionalidade f_{cr} para crédito-ou-depósito é definida como a seguir:*

E1. (Etapa de Depósito.) O objetivo desta etapa é fazer com que os participantes especifiquem depósitos condicionais com o formato $\left[P_i \xrightarrow[c,\tau_{i,j}]{\varphi_{i,j}} P_j\right]$.

Para isso, ao receber a tupla $(depósito,i,j,\varphi_{i,j},\tau_{i,j},valor(c))$ especificando o depósito de c moedas de P_i para P_j, f_{cr} armazena a mensagem $(depósito,i,j,\varphi_{i,j},\tau_{i,j},c)$ e a envia para

todos os participantes.

E2. (Etapa de Crédito.) O objetivo desta etapa é fazer com que os participantes reivindiquem depósitos feitos na etapa anterior.

Para isso, na rodada $\tau_{i,j}$, ao receber a tupla $(crédito, i, j, \varphi_{i,j}, \tau_{i,j}, c, p)$ de P_j para creditar um depósito feito por P_i:

(i) f_{cr} verifica se a mensagem $(depósito, i, j, \varphi_{i,j}, \tau_{i,j}, valor(c))$ está armazenada,

(ii) f_{cr} verifica se $\varphi_{i,j}(p) = 1$.

Se essas duas condições forem atendidas, então:

(i) f_{cr} envia para todos os participantes a mensagem $(crédito, i, j, \varphi_{i,j}, \tau_{i,j}, c, p)$

(ii) f_{cr} envia apenas para P_j a mensagem $(crédito, i, j, \varphi_{i,j}, \tau_{i,j}, valor(c))$

(iii) f_{cr} remove a mensagem $(depósito, i, j, \varphi_{i,j}, \tau_{i,j}, c)$, evitando que esse depósito seja reivindicado uma segunda vez.

E3. (Fase de Reembolso.) O objetivo desta etapa é fazer o reembolso dos depósitos que não foram creditados antes de suas rodadas limite.

Para isso, na rodada $\tau_{i,j} + 1$, se existe uma mensagem $(depósito, i, j, \varphi_{i,j}, \tau_{i,j}, c)$ armazenada, então f_{cr} envia a mensagem $(reembolso, i, j, \varphi_{i,j}, \tau_{i,j}, valor(c))$ para P_j e remove a mensagem $(depósito, i, j, \varphi_{i,j}, \tau_{i,j}, c)$.

A funcionalidade f_{cr} define uma espécie de "ciclo de vida" para um depósito. Ele começa na etapa de depósito, onde f_{cr} recebe especificações de depósitos e as armazena. Uma das informações contidas nessas especificações é a rodada $\tau_{i,j}$ na qual o depósito deve ser creditado. Dessa forma, na rodada $\tau_{i,j}$, o depósito entra na etapa de crédito. Nela, f_{cr} espera que o destinatário do depósito (P_j) apresente uma entrada p que satisfaça o circuito $\varphi_{i,j}$. Se isso acontecer, o valor especificado de moedas ($valor(c)$) é transferido para P_j e o depósito não é mais considerado por f_{cr}. Caso contrário, na rodada $\tau_{i,j} + 1$, o depósito entra na fase de reembolso, onde f_{cr} devolve as c moedas para o emissor do depósito (P_i).

Tanto na fase de depósito quanto na de crédito, f_{cr} faz *broadcasts* atualizando os participantes sobre seu estado. Dessa forma, todo participante fica ciente quando um novo depósito é realizado ou quando uma operação de crédito é bem sucedida. Em particular, quando um depósito é reivindicado, *todo* participante aprende a testemunha que satisfaz o circuito especificado. Essa característica de f_{cr} é especialmente importante para permitir sua implementação através de transações *Bitcoin*.

Ressaltamos a diferença entre as mensagens que enviam $valor(c)$ e as que enviam apenas o natural c. Definimos $valor(c)$ como um conjunto de c moedas e, portanto, mensagens que enviam $valor(c)$ fazem uma transferência de moedas. Por outro lado, as mensagens que enviam

apenas c têm como objetivo atualizar os participantes sobre a criação ou reivindicação de algum depósito.

Por fim, ressaltamos que, ao fazer um depósito, o participante *transfere* o controle das moedas para f_{cr}. Elas permanecem congeladas até que o depósito seja creditado ou reembolsado.

7.3 Construção Escada no Modelo f_{cr}-híbrido

A Figura 7.1 esquematiza como o protocolo que chamamos de *construção escada* utiliza a funcionalidade f_{cr} como uma "caixa preta". Nessa seção vamos formalizar esse esquema, definindo a construção escada no modelo *modelo f_{cr}-híbrido*, isto é, com acesso oracular à f_{cr}.

Definição 7.101 (Construção escada no modelo f_{cr}-híbrido - BENTOV e KUMARESAN (2014))**.** *Seja $\{P_1,\cdots,P_m\}$ um conjunto de participantes, $(Divide, Reconstrói, Verifica)$ um $EVCS_{m,m}$ como implementado na Definição 6.97 e (Etiquetas, Pedaços) o resultado da aplicação do algoritmo Divide a algum segredo z a ser reconstruído. Para todo $i \in \{1,\cdots,m\}$, assumimos que o participante P_i possui a lista Etiquetas e o valor $Pedaços_i$. Também assumimos que q é um valor predefinido, utilizado como multa por aborto. As rodadas $\tau_1,\cdots,\tau_m$ são tais que $\tau_i < \tau_{i+1}$ para todo $i \in \{1,\cdots,m-1\}$ (ou seja, a rodada τ_i começa antes da rodada τ_{i+1}).*

Definições auxiliares: *Definimos $\varphi(et,\cdot)$ como o circuito que é satisfeito por uma entrada p se e somente se $Verifica(et,p) = 1$. Isto é, $\varphi(et,\cdot)$ é satisfeito pelo pedaço p correspondente à etiqueta et. Definimos, para todo $i \in \{1,\cdots,m\}$, $\varphi_i^{esc} \stackrel{def}{=} \varphi(Etiquetas_1,\cdot) \wedge \varphi(Etiquetas_2,\cdot) \wedge \cdots \wedge \varphi(Etiquetas_i,\cdot)$ e $\varphi^{cob} \stackrel{def}{=} \varphi_m^{esc}$.*

Definimos a construção escada como a seguir.

E1. (Depósitos-cobertura.) Nesta etapa os depósitos-cobertura são feitos. Isto é, para todo $i \in \{1,\cdots,m-1\}$, P_i faz o depósito $\left[P_i \xrightarrow[q,\tau_m]{\varphi^{cob}} P_m\right]$. O protocolo só continua se todos os $m-1$ depósitos-cobertura forem feitos.

Mais precisamente, para todo $i \in \{1,\cdots,m-1\}$, o participante P_i faz o seguinte:

- *Envia $(depósito, i, m, \varphi^{cob}, \tau_m, valor(q))$ para f_{cr}.*
- *Se a mensagem $(depósito, e, m, \varphi^{cob}, \tau_m, q)$ não foi recebida de f_{cr} para algum $e \in \{1,\cdots,m-1\}$, então P_i suspende a execução e espera até a rodada $\tau_m + 1$ pela mensagem $(reembolso, i, m, \varphi^{cob}, \tau_m, valor(q))$ de f_{cr}. Nesse caso, P_i define sua saída como $\perp$.*

E2. (Depósitos-escada.) Nesta etapa os depósitos-escada são realizados. O participante P_{i+1} só faz seu depósito-escada $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{\varphi_i^{esc}} P_i\right]$ se todos os depósitos "acima", isto é, aqueles dos participantes $P_{i+2},\cdots,P_m$, tiverem sido feitos.

*Para isso, cada participante P_{i+1}, para $i = m-1$ até 1, envia $(depósito, i+1, i, \varphi_i^{esc}, \tau_i, valor(i \cdot q))$ para f_{cr} **somente se** a mensagem $(depósito, e+1, e, \varphi_e^{esc}, \tau_e, e \cdot q)$ foi recebida de f_{cr} para todo e tal que $i+1 \leq e \leq m-1$.*

E3. (Crédito dos depósitos-escada.) Nesta etapa os depósitos-escada são reivindicados na ordem inversa da que são feitos. Dividimos as ações em dois casos:

Ação de P_1: Na rodada τ_1 o participante P_1 envia a mensagem $(crédito, 2, 1, \varphi_1^{esc}, \tau_1, q, W_1 = \{Pedaços_1\})$ para f_{cr}. Ele então espera receber a mensagem $(crédito, 2, 1, \varphi_1^{esc}, \tau_1, valor(q))$ de f_{cr}.

Ação de P_{i+1}, para todo $i \in \{1, \cdots, m-1\}$: P_{i+1} espera a rodada τ_i para que seu depósito-escada $\left[P_{i+1} \xrightarrow[i\cdot q, \tau_i]{\varphi_i^{esc}} P_i\right]$ seja reivindicado. Ou seja, espera receber a mensagem $(crédito, i+1, i, \varphi_i^{esc}, \tau_i, i\cdot q, W_i)$ de f_{cr}.

- Se essa mensagem for recebida e $i+1 \neq m$, P_{i+1} reivindica o depósito $\left[P_{i+2} \xrightarrow[(i+1)\cdot q, \tau_{i+1}]{\varphi_{i+1}^{esc}} P_{i+1}\right]$. Para isso,
  - P_{i+1} cria o conjunto $W_{i+1} = W_i \cup \{Pedaços_{i+1}\}$,
  - P_{i+1} envia a mensagem $(crédito, i+2, i+1, \varphi_{i+1}^{esc}, \tau_{i+1}, (i+1)\cdot q, W_{i+1})$ para f_{cr},
  - P_{i+1} espera receber de f_{cr} a mensagem $(crédito, i+2, i+1, \varphi_{i+1}^{esc}, \tau_{i+1}, valor((i+1)\cdot q), W_{i+1})$.
- Se essa mensagem não for recebida, P_{i+1} suspende a execução e espera a rodada $\tau_i + 1$ para ter seu depósito-escada reembolsado. Ou seja, P_{i+1} espera a mensagem $(reembolso, i+1, i, \varphi_i^{esc}, \tau_i, valor(i\cdot q))$ de f_{cr}. Além disso, se $i+1 \neq m$, P_{i+1} deve também ter seu depósito-cobertura reembolsado. Para isso, P_{i+1} espera a rodada $\tau_m + 1$ para receber de f_{cr} a mensagem $(reembolso, i+1, m, \varphi^{cob}, \tau_m, valor(q))$. Finalmente, P_{i+1} define sua saída como $\perp$.

E4. (Crédito dos depósitos-cobertura.) Nesta etapa o participante P_m reivindica os depósitos-cobertura.

Para isso, na rodada τ_m, para todo $i \in \{1, \cdots, m-1\}$, o participante P_m reivindica o depósito $\left[P_i \xrightarrow[q, \tau_m]{\varphi^{cob}} P_m\right]$ enviando $(crédito, i, m, \varphi^{cob}, \tau_m, q, W_m = Pedaços)$ para f_{cr} e recebendo de volta a mensagem $(crédito, i, m, \varphi^{cob}, \tau_m, valor(q), W_m)$. Então P_m utiliza o algoritmo *Reconstrói* sobre $W_m = Pedaços$ e reconstrói o valor z.

E5. (Coleta dos pedaços.) Nesta etapa os participantes utilizam as informações da etapa anterior para reconstruir o valor z.

Para todo $i \in \{1, \cdots, m-1\}$, o participante P_i espera a rodada τ_m para receber de f_{cr} a mensagem $(crédito, s, m, \varphi^{cob}, \tau_m, q, W_m)$ para algum $s \in \{1, \cdots, m-1\}$. Então P_i utiliza $W_m = Pedaços$ para reconstruir z. Caso nenhuma dessas mensagens seja recebida, P_i aguarda até a rodada $\tau_m + 1$ para ter seu depósito-cobertura reembolsado através da mensagem $(reembolso, i, m, \varphi^{cob}, \tau_m, valor(q))$ enviada por f_{cr}.

Nós demos uma visão geral, na Figura 7.1, de como os depósitos escada e cobertura são feitos. Prosseguimos com uma descrição mais detalhada baseada na Definição 7.101.

Podemos ver o crédito dos depósitos-cobertura como o objetivo final da construção escada. De fato, é a reivindicação desses depósitos que permite a reconstrução do segredo z. Por essa razão, os depósitos-cobertura devem ser os primeiros a serem feitos, já na etapa E1. Se algum participante não realizar seu depósito-cobertura, a execução deve ser interrompida e os depósitos já feitos, reembolsados.

Os depósitos-escada são feitos na etapa E2. Com exceção de P_1 e P_m, todo participante P_i está envolvido em dois depósitos-escada consecutivos, isto é, $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{\varphi_i^{esc}} P_i\right]$ e $\left[P_i \xrightarrow[(i-1)\cdot q,\tau_{i-1}]{\varphi_{i-1}^{esc}} P_{i-1}\right]$. Como veremos a seguir, é fundamental para a justiça da construção escada que o participante P_i faça seu depósito escada *somente se* o depósito logo acima, do qual ele é o destinatário, tiver sido feito. Isso se justifica pelo fato de que é esse depósito que torna seu saldo temporariamente positivo ($+q$). Devido a isso, os depósitos-escada são feitos de "cima" para "baixo", começando com $\left[P_m \xrightarrow[(m-1)\cdot q,\tau_{m-1}]{Pedaços_1,\cdots,Pedaços_{m-1}} P_{m-1}\right]$ realizado por P_m. Isso permite a cada participante verificar se o depósito do qual ele é destinatário já foi realizado. Essa verificação só é possível porque a funcionalidade f_{cr} faz o *broadcast* dos eventos que atualizam seu estado.

Na etapa E3, os depósitos-escada são creditados de "baixo" para "cima". Cada participante P_{i+1}, com $i \in \{1,\cdots,m-1\}$ espera seu depósito-escada $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{Pedaços_1,\cdots,Pedaços_i} P_i\right]$ ser creditado. Somente assim ele conseguirá reunir os elementos de *Pedaços* necessários para reivindicar o depósito-escada logo acima, do qual ele é o destinatário. Ressaltamos que apenas através da reivindicação do depósito $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{Pedaços_1,\cdots,Pedaços_i} P_i\right]$ o valor $Pedaços_i$ se torna conhecido. O valor $Pedaços_m$, por sua vez, só se torna conhecido nos depósitos-cobertura.

Ainda na etapa E3, se algum participante abortar, não reivindicando um depósito-escada e, portanto, *impedindo os depósitos acima de serem creditados*, a execução é interrompida. Nesse caso, os depósitos acima do ponto de aborto, i.e., os depósitos ainda não creditados, são reembolsados. Além disso, os depósitos-cobertura também são reembolsados, na rodada $\tau_m + 1$.

No entanto, nem todo aborto impossibilita a continuação da etapa E3. Considere o exemplo em que os participantes P_{i+1} e P_i fazem parte de um conluio. Nesse caso, podemos admitir que ambos conhecem os valores $Pedaços_{i+1}$ e $Pedaços_i$. Dessa forma, mesmo que P_i aborte e não reivindique o depósito $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{\varphi_i^{esc}} P_i\right]$, os depósitos acima dele, em particular o $\left[P_{i+2} \xrightarrow[i\cdot q,\tau_{i+1}]{\varphi_{i+1}^{esc}} P_{i+1}\right]$, ainda podem ser creditados. De fato, o depósito $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{\varphi_i^{esc}} P_i\right]$ nem precisa ser feito.

Após o crédito dos depósitos-escada e se não houve aborto, o participante P_m possui todos os elementos da lista *Pedaços*. Com isso, ele é o primeiro participante capaz de usar o algoritmo *Reconstrói* para obter a saída do protocolo. Além disso, com a lista *Pedaços* completa,

P_m é também capaz de reivindicar os depósitos-cobertura, na etapa E4. Como f_{cr} faz o *broadcast* das operações de crédito, P_m só precisa reivindicar *um* depósito cobertura para que todos os demais participantes recebam a lista *Pedaços*. Isso está registrado na etapa E5. No entanto, a estratégia de reivindicar apenas alguns depósitos-cobertura resulta em um prejuízo financeiro para P_m.

Finalmente, o caso em que P_m faz parte de um conluio desonesto é o único em que a justiça (tradicional) do protocolo pode ser quebrada. Ou seja, P_m pode abortar logo após reconstruir a saída. Nesse caso, todos os depósitos-cobertura são reembolsados e os demais participantes acabam com um saldo positivo de q moedas[2].

Ao final do protocolo, se todos os participantes agirem honestamente, nenhuma penalidade é paga. Ou seja, todo investimento feito nos depósitos escada e cobertura são recuperados. Além disso, todos conseguem obter a saída a partir do algoritmo *Reconstrói*. No entanto, caso ocorra um aborto, todo participante que reivindicou depósitos-escada, revelando seu pedaço do segredo, recebe q moedas como compensação. Isso implica que, quando o aborto ocorre após a saída ter sido descoberta por algum conluio, *todos* os participantes honestos recebem essa compensação. Essas características fazem a construção escada ser monetariamente justa.

Analisamos formalmente a justiça da construção escada no lema a seguir e no corolário subsequente.

Lema 7.102 (Estado final da construção escada)**.** *Para todo $i \in \{1, \cdots, m\}$, se P_i é honesto, então, no final da construção escada (i.e., após a rodada τ_m) temos que:*

(i) O depósito-escada $\left[P_{i+1} \xrightarrow[i\cdot q, \tau_i]{Pedaços_1, \cdots, Pedaços_i} P_i\right]$ *foi reivindicado, assim como o depósito logo abaixo* $\left[P_i \xrightarrow[(i-1)\cdot q, \tau_{i-1}]{Pedaços_1, \cdots, Pedaços_{i-1}} P_{i-1}\right]$*, se $i \neq 1$, ou*

(ii) Nenhum depósito-cobertura foi creditado e, para todo $j \in \{i+1, \cdots, m-1\}$, o depósito-escada $\left[P_{j+1} \xrightarrow[j\cdot q, \tau_j]{Pedaços_1, \cdots, Pedaços_j} P_j\right]$ *também não foi creditado (ou seja, os depósitos-escada acima de* $\left[P_{i+1} \xrightarrow[i\cdot q, \tau_i]{Pedaços_1, \cdots, Pedaços_i} P_i\right]$*).*

Demonstração. Para $i = 1$, como P_1 é honesto por hipótese, ele reivindica o depósito $deposit P_2 P_1 q \tau_1 Pedaços_1$ e a propriedade (i) se verifica no final da execução. Para $i \neq 1$, vamos considerar os casos em que o depósito-escada $\left[P_i \xrightarrow[(i-1)\cdot q, \tau_{i-1}]{Pedaços_1, \cdots, Pedaços_{i-1}} P_{i-1}\right]$, que nesse caso existe, foi ou não reivindicado.

- Se P_{i-1} reivindicou esse depósito, então o conjunto $\{Pedaços_1, \cdots, Pedaços_{i-1}\}$ foi publicado e, em particular, é conhecido por P_i. Portanto, tudo o que P_i precisa para montar o conjunto $\{Pedaços_1, \cdots, Pedaços_i\}$ e reivindicar o depósito $\left[P_{i+1} \xrightarrow[i\cdot q, \tau_i]{Pedaços_1, \cdots, Pedaços_i} P_i\right]$ é

[2] Como argumentamos na visão geral do protocolo.

o valor $Pedaços_i$, o qual ele possui. Como P_i é honesto, ele segue o protocolo e reivindica esse depósito. Com isso, no fim da execução, temos que a propriedade (i) se verifica.

- Se P_{i-1} *não* reivindicou esse depósito, P_i não pode reivindicar $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{Pedaços_1,\cdots,Pedaços_i} P_i\right]$ porque não conhece todo o conjunto $\{Pedaços_1,\cdots,Pedaços_{i-1}\}$. Nesse caso, nenhum participante aprende o valor $Pedaços_i$. Sem o conhecimento de $Pedaços_i$, nenhum depósito-cobertura pode ser reivindicado. Além disso, nem o depósito $\left[P_{i+1} \xrightarrow[i\cdot q,\tau_i]{Pedaços_1,\cdots,Pedaços_i} P_i\right]$, nem qualquer outro acima dele pode ser reivindicado. Nesse caso, ao fim da execução, temos que a propriedade (ii) se verifica.

□

Com o lema anterior, podemos estabelecer também o seguinte corolário.

Corolário 7.103 (Justiça da construção escada)**.** *Ao final da execução da construção escada:*

(i) Todos os participantes honestos recebem todos os elementos de Pedaços sem pagar nenhuma penalidade, ou

(ii) Os participantes honestos que não revelaram seus pedaços têm seus investimentos reembolsados. Além disso, cada participante honesto que revelou seu pedaço é compensado com q moedas,

concluindo que a construção escada é monetariamente justa .

Demonstração. Vamos analisar os dois casos:

(i) Um participante honesto P_i só pode receber todos os elementos de *Pedaços* se os depósitos-cobertura e, consequentemente, os escada, forem creditados. Nesse caso, se $i \neq m$, seu saldo final será $valor(i \cdot q)$ (valor recebido no depósito-escada reivindicado) - $valor((i-1)\cdot q)$ (valor pago no seu depósito-escada) - q (valor pago no depósito-cobertura) = 0. Se $i = m$, seu saldo final será $valor((m-1)\cdot q)$ (valor recebido nos $(m-1)$ depósitos-cobertura) - $valor((m-1)\cdot q)$ (valor pago no seu depósito-escada) = 0.

(ii) Se o participante honesto P_i, não revelou seu pedaço ($Pedaços_i$), então, pelo Lema 7.102, nem os depósitos-escada envolvendo P_i nem os depósitos-cobertura foram creditados. Por isso todo o investimento feito por P_i é reembolsado. Por outro lado, se P_i, com $i \neq m$, revelou seu pedaço[3], então seu saldo final será $valor(i \cdot q)$ (valor recebido no depósito-escada reivindicado) - $valor((i-1)\cdot q)$ (valor pago no seu depósito-escada) + (q-q) (valor investido e reembolsado no seu depósito-cobertura[4]) = q.

[3] Se $i = m$, teríamos que propriedade (i) se verificaria, pois se P_m revela seu pedaço então *todos* os pedaços são revelados.

[4] O depósito-cobertura precisar ser reembolsado, pois se não o fosse, a lista *Pedaços* seria revelada e a propriedade (i) se verificaria.

□

O Corolário 7.103 estabelece que a construção escada é monetariamente justa. Uma das exigências da Definição 5.95 é que uma compensação seja paga aos participantes honestos caso a justiça tradicional seja quebrada, ou seja, caso um conluio desonesto obtenha a saída do protocolo e aborte. A construção escada vai um pouco além. Participantes honestos podem ser compensados na ocorrência de qualquer aborto, mesmo que ninguém reconstrua a saída e, portanto, a justiça (tradicional) não seja quebrada. No entanto, a construção escada só garante que *todos* os participantes honestos sejam compensados se a justiça (tradicional) for, de fato, quebrada.

Abordaremos a segurança dessa construção mais adiante.

7.4 A Funcionalidade f_{rec}^{esc} para Reconstrução Justa

O objetivo de uma funcionalidade é especificar como uma determinada tarefa é realizada idealmente, facilitando a análise da correção e segurança de suas possíveis implementações. Por isso, a nossa estratégia geral tem sido a de primeiro definir funcionalidades para só então exibir protocolos para computá-las. Essa ordem, no entanto, foi invertida com a construção escada, que foi apresentada *antes* de uma funcionalidade correspondente. Fizemos essa inversão porque a funcionalidade f_{rec}^{esc} para reconstrução justa, também retirada de BENTOV e KUMARESAN (2014), pode ser melhor compreendida conhecendo-se, de antemão, sua implementação.

Preenchemos essa lacuna na apresentação definindo f_{rec}^{esc} a seguir.

Definição 7.104 (Funcionalidade f_{rec}^{esc} para reconstrução justa)**.** *Seja $\{P_1,\cdots,P_m\}$ o conjunto de participantes, $C \stackrel{def}{=} \{c_1,\cdots,c_t\} \subset \{1,\cdots,m\}$ o conluio dos participantes controlado pelo adversário S e $H \stackrel{def}{=} \{1,\cdots,m\} \setminus C$ o conjunto dos participantes honestos. Seja d um valor de depósito de segurança e q o valor da multa por aborto. Assumimos acesso ao $EVCS_{m,m}$ implementado na Definição 6.97. Definimos a funcionalidade f_{rec}^{esc} para reconstrução justa como a seguir.*

E1. (Entradas:) Nesta etapa os participantes honestos e o adversário, em nome dos desonestos, fornecem seus elementos da lista Pedaços e fazem seus investimentos financeiros iniciais.

Mais precisamente, para todo $h \in H$, P_h envia a mensagem $(entrada, h, Etiquetas, Pedaços_h, valor(d))$ para f_{rec}^{esc}. O adversário S envia a mensagem $(entrada, \{Pedaços_{c_1},\cdots,Pedaços_{c_t}\}, H', valor(|H'|\cdot q))$, com $H' \subseteq H$.

E2. (Retorno de Investimento:) Nesta etapa, o investimento dos participantes honestos é devolvido.

Para isso, f_{rec}^{esc} envia a mensagem $(reembolso, valor(d))$ para P_h para todo $h \in H$.

E3. (Cálculo e Entrega das Saídas:) Nesta etapa, f_{rec}^{esc} reconstrói o segredo a partir das listas Etiquetas e Pedaços fornecidas como entrada. O fluxo da execução desta etapa depende

tanto do conjunto H' quanto da decisão do adversário abortar ou não. Caso o adversário decida interromper a execução, ele paga uma multa a cada participante honesto. Mais precisamente, a etapa é conduzida como a seguir:

- *O valor $z = Reconstrói(Etiquetas, Pedaços)$ é calculado a partir da invocação do algoritmo de reconstrução do $EVCS_{m,m}$.*
- *Se $|H'| = 0$, então a mensagem $(saída, z)$ é enviada para P_i, para todo $i \in \{1, \cdots, m\}$, e a execução é encerrada.*
- *Se $0 < |H'| < |H|$, então a mensagem $(pagamento\text{-}extra, valor(q))$ é enviada para P_h, para todo $h \in H'$, e a execução é encerrada.*
- *Se $|H'| = |H|$, então a mensagem $(saída, z)$ é enviada para S:*
  - *Se o adversário S responder $(continue, H'')$, então:*
    - *(i) a mensagem $(saída, z)$ é enviada para P_h, para todo $h \in H$,*
    - *(ii) a mensagem $(reembolso, valor((|H| - |H''|) \cdot q))$ é enviada para S,*
    - *(iii) a mensagem $(pagamento\text{-}extra, valor(q))$ é enviada para P_h, para todo $h \in H''$.*
  - *Se o adversário S responder $(aborte, \perp)$, a mensagem $(compensação, valor(q))$ é enviada para P_h, para todo $h \in H$.*

O objetivo geral da funcionalidade f_{rec}^{esc} é reconstruir de forma justa um valor z gerado pelo algoritmo *Divide* de um $EVCS_{m,m}$ implementado conforme a Definição 6.97. A etapa E1, conforme a Definição 7.104, existe para que os participantes façam seus investimentos e forneçam suas entradas. Além disso, nessa etapa, o adversário S define o conjunto $H' \subseteq H$, o qual é muito importante para a continuidade da execução. Ele representa o subconjunto de participantes honestos que o adversário está "disposto" a compensar para poder se comportar desonestamente sem quebrar a justiça monetária. Fazendo uma relação de f_{rec}^{esc} com sua implementação (a construção escada), H' pode ser entendido como o conjunto dos participantes honestos que conseguiram reivindicar seus depósitos-escada. Eles são recompensados no caso em que, devido a um aborto, não receberem suas saídas.

Dessa maneira, na etapa E3, se $H' = \emptyset$, a saída z é enviada para todo participante, pois o adversário *não investiu* para agir maliciosamente. Na construção escada, esse cenário ocorre quando a etapa de crédito dos depósitos-escada é concluída sem nenhum aborto e P_m é honesto[5].

Já no caso em que $0 < |H'| < |H|$, o adversário impede que os participantes recebam a saída, embora ele próprio também não a receba. Na construção escada, esse cenário ocorre quando, devido a um aborto, apenas *alguns* participantes honestos conseguem reivindicar depósitos-escadas. Ao final da execução do protocolo, esses participantes possuem um saldo

[5] O caso em que P_m é desonesto corresponde a $|H'| = |H|$, pois o adversário pode abortar *depois* de ver a saída

positivo de q moedas. Por essa razão, eles são compensados da mesma maneira em f_{rec}^{esc}, embora isso não fosse necessário para a justiça monetária da funcionalidade.

Por fim, se $|H'| = |H|$, então o adversário tem o poder de decidir sobre a continuidade da execução *com base na saída* da funcionalidade[6]. Somente dessa forma o adversário pode quebrar a justiça tradicional de f_{rec}^{esc}. No cenário correspondente na construção escada, todos os participantes honestos reivindicaram depósitos-escada. Nesse caso, P_m, responsável por reivindicar os depósitos-cobertura é desonesto[7] e decide abortar ou não com base na saída que ele obteve. Se ele abortar, tanto em f_{rec}^{esc} quanto na construção escada, todo participante honesto recebe uma compensação.

Por outro lado, se ele decidir continuar a execução, f_{rec}^{esc} precisa lidar com outro subconjunto de participantes honestos, o $H'' \subseteq H$. Os componentes de H'' também recebem uma compensação, mas esse conjunto não tem um significado intuitivo quando analisamos f_{rec}^{esc} independentemente. Levando em conta a construção escada, no entanto, podemos encarar H'' como o conjunto dos participantes que tiverem seus depósitos-cobertura *reembolsados*. De fato, se P_m for desonesto, ele pode escolher *alguns* depósitos-cobertura para reivindicar, deixando outros para serem reembolsados. Esse tipo de estratégia não é financeiramente benéfica para o adversário, mas é possível na construção escada. Portanto, o conjunto H'' precisa ser introduzido na Definição 7.104 para possibilitar a prova de que a construção escada é um protocolo seguro para computar f_{rec}^{esc}.

Observamos que a etapa E2 também pode ser melhor compreendida com o auxílio da construção escada. Isoladamente ela parece ser desnecessária. Idealmente, o próprio investimento por parte dos participantes honestos deveria ser desnecessário, uma vez que eles nunca pagam penalidades. No entanto, essa etapa é justificada pelo fato de que, na construção escada, *todos* os participantes fazem investimentos. Sem ela, dificilmente poder-se-ia provar que a construção escada é uma implementação segura de f_{rec}^{esc}.

Por fim, ressaltamos que f_{rec}^{esc} é monetariamente justa. Lembramos que a Definição 5.95 de justiça monetária faz duas exigências em relação ao saldo dos participantes ao fim da execução. A primeira é que nenhum participante honesto pague penalidades. De fato isso é verdade em f_{rec}^{esc} porque, na etapa E2, todo investimento feito por esses participantes é reembolsado. A segunda exigência é que os participantes honestos sejam compensados caso a justiça tradicional seja quebrada. Esse caso é tratado na etapa E3, quando o adversário decide abortar e a mensagem $(\mathit{compensação}, \mathit{valor}(q))$ é enviada para todos os participantes honestos.

Agora que apresentamos a funcionalidade f_{rec}^{esc}, podemos estabelecer que a construção escada é um protocolo que computa f_{rec}^{esc} de maneira segura. Registramos isso no lema a seguir.

Lema 7.105 (Segurança da construção escada)**.** *A construção escada (Definição 7.101) é um protocolo seguro para computar f_{rec}^{esc} no modelo f_{cr}-híbrido. (A construção escada uc-realiza f_{rec}^{esc}.)*

[6] Porque o adversário, nesse caso, está disposto a compensar *todo* participante honesto.

[7] Se P_m fosse honesto, então $|H'| \neq |H|$, pois P_m não reivindica depósitos-escada.

Demonstração. A ideia geral da prova, baseada na apresentação feita em BENTOV e KUMARESAN (2014), é criar um adversário S ideal para f_{rec}^{esc} que *imita* o comportamento de um adversário real A para a construção escada. Isso se baseia na noção de segurança da Definição 6.96. Para tanto, o adversário S precisa *simular* o ambiente de execução da construção escada para A. Isso significa, por exemplo, que S deve agir como o oráculo f_{cr} para enviar e receber mensagens de depósitos, tanto as de *broadcast* quanto as que de fato transferem moedas.

O *simulador* S precisa também "se passar" pelos participantes honestos, simulando suas ações. Isso é um desafio porque S não conhece as entradas desses participantes, as quais precisam ser reveladas para A durante o crédito dos depósitos escada e cobertura. Por fim, S precisa também contactar a funcionalidade f_{rec}^{esc}, definindo os conjuntos H', H'' e decidindo se aborta ou não de acordo com o comportamento de A.

Com a construção desse simulador queremos estabelecer que qualquer adversário (real) para a construção escada tem um correspondente (ideal) que atinge os mesmos efeitos ao atacar f_{rec}^{esc}.

Breve Recapitulação da Nomenclatura Utilizada. Lembramos que as entradas dos m participantes de f_{rec}^{esc} são os pares $(Etiquetas, Pedaços_i)$, gerados a partir da aplicação do algoritmo *Divide* de um $EVCS_{m,m}$, implementado conforme a Definição 6.97, sobre um segredo z. O valor $Pedaços_i$ é um par (p_i, w_i) tal que $Etiquetas_i = \overline{Comp}_{w_i}(p_i)$. Ou seja, $Etiquetas_i$ é o resultado de se comprometer com o pedaço p_i utilizando w_i como fonte de aleatoriedade. O esquema de comprometimento *Comp* para um *bit* e sua extensão para cadeias, $\overline{Comp}$, são, por hipótese, equívocos e, portanto, possuem os algoritmos adicionais *Equivoca* e *Desequivoca* conforme a Definição 3.17. Lembramos ainda que $\bigoplus_{i=1}^{m} p_i = z$.

Ressaltamos que o esquema de comprometimento utilizado, $\overline{Comp}$, lida com *cadeias* de *bits* e, da mesma forma, os algoritmos *Equivoca* e *Desequivoca* a ele associados.

Denotamos por C o conjunto dos índices dos participantes desonestos da execução de f_{rec}^{esc} e por H o conjunto $\{1, \cdots, m\} \setminus C$ dos índices dos participantes honestos.

Visão Geral da Simulação. De maneira geral, a simulação conduzida pelo adversário S, que chamaremos também de *simulador*, pode ser dividida em 4 etapas: uma para simular os depósitos-cobertura, outra para simular os depósitos-escada e outras duas para simular o crédito e reembolso desses depósitos. Nessas etapas, o simulador precisa enviar e receber mensagens como se fosse o oráculo f_{cr}.

Uma das dificuldades de "se fingir" ser o oráculo f_{cr} é ter que lidar com moedas. Todas as moedas que S tem para conduzir a simulação são aquelas recebidas de A nos depósitos-escada, uma vez que o modelo de computação adotado não lhe permite criar (ou forjar) moedas. Para acompanhar o fluxo de moedas e facilitar a apresentação, cada participante P_i possui uma variável associada $Carteira_i$. Ela guarda as moedas que P_i tem para gastar. Além disso, a *CarteiraPenalidade* guarda as moedas que serão utilizadas para contactar a funcionalidade f_{rec}^{esc}.

Outro ponto importante da simulação é o fato de que S precisa simular o comportamento dos participantes honestos, mesmo sem conhecer as entradas deles. Isso é relevante porque A

pode definir sua saída com base no segredo reconstruído z e, portanto, S precisa "forjar" valores para $Pedaços_i$, para todo $i \in H$, de modo que a soma (ou-exclusivo) de todos os pedaços seja igual a z. No entanto, S só vem a conhecer z quando contacta f_{rec}^{esc}.

Para contornar esse problema a simulação se baseia na seguinte observação. O adversário A só descobre o valor z quando o último participante honesto credita um depósito, que pode ser escada ou cobertura. De fato, o pedaço revelado nesse crédito é o último que falta a A para reconstruir z. Essa observação, aliada ao fato de que o esquema de comprometimento utilizado pelo $EVCS_{m,m}$ é equívoco, permite que S aja da seguinte maneira. Seja $MaxH \stackrel{def}{=} \max\{H\}$ o índice do último participante a creditar algum depósito na construção escada. Então, para todo $h \in H \setminus \{MaxH\}$, o simulador S define $Pedaços'_h$ como um valor aleatoriamente gerado e $Etiquetas'_h$ como o resultado de se comprometer com esse valor. Para $MaxH$, o simulador invoca o algoritmo $Equivoca$ sobre um valor aleatório r para obter $(coringa_{MaxH}, info_{MaxH}) = Equivoca(r)$ e define $Etiquetas'_{MaxH} = coringa_{MaxH}$.

Usamos $Etiquetas'$ e $Pedaços'$ (ao invés de $Etiquetas$ e $Pedaços$) para deixar claro que as listas utilizadas na simulação não são as originais, possuídas pelos participantes no início da execução de f_{rec}^{esc}.

Em linhas gerais, o simulador cria uma etiqueta "coringa", ou *ambígua*, para P_{MaxH}. Quando chegar a hora desse participante revelar seu $Pedaços'_{MaxH}$, S deve estar apto a contactar f_{rec}^{esc}, obter a saída z, calcular $p_{MaxH} = \left(z \bigoplus_{i \neq MaxH} p_i\right)$, em que $Pedaços'_i = (p_i, w_i)$, e utilizar $(info_{MaxH}, p_{MaxH})$ para desequivocar $Etiquetas'_{MaxH}$, gerando um valor $Pedaços'_{MaxH} = Desequivoca(info_{MaxH}, p_{MaxH})$ adequado.

Também é preciso lidar, durante a simulação, com a construção adequada dos conjuntos H' e H'' de modo que S tenha moedas suficientes para contactar f_{rec}^{esc}.

Por fim, S precisa lidar também com os casos excepcionais em que A consegue quebrar o sigilo ou a não-ambiguidade de $Comp$ (e $\overline{Comp}$). Esses casos são importantes porque são os únicos em que S não consegue imitar A com perfeição. Eles, no entanto, só podem acontecer com probabilidade desprezível.

Nas seções seguintes apresentaremos como S deve simular as etapas da construção escada.

Inicialização do Simulador. O simulador executa os seguintes passos para inicializar valores importantes:

1. *(Índice do último participante honesto:)* $MaxH = \max\{H\}$.
2. *(Listas $Etiquetas'$ e $Pedaços'$:)* Os pedaços e etiquetas dos participantes desonestos, durante a simulação, são os originais[8]. Os dos participantes honestos, com exceção de P_{MaxH}, são gerados aleatoriamente. Ou seja:

[8]Como já ressaltamos, as listas originais, $Etiquetas$ e $Pedaços$, são as que compõem as entradas dos participantes na execução de f_{rec}^{esc}. As listas $Etiquetas'$ e $Pedaços'$ são as utilizadas durante a simulação.

Para todo $c \in C$, $Etiquetas'_c = Etiquetas_c$ e $Pedaços'_c = Pedaços_c$. Para todo $h \in H \setminus \{MaxH\}$, $Pedaços'_h = (p_h, w_h)$ e $Etiquetas'_h = \overline{Comp}_{w_h}(p_h)$, em que p_h e w_h são cadeias escolhidas aleatoriamente.

3. *(Etiquetas'$_{MaxH}$:)* $Etiquetas'_{MaxH}$ é um valor coringa, gerado pelo algoritmo *Equivoca*.

   $Etiquetas'_{MaxH} = coringa_{MaxH}$, em que $(coringa_{MaxH}, info_{MaxH}) = Equivoca(r)$, com r aleatoriamente escolhido. O valor $info_{MaxH}$ deve ser armazenado e vai servir para desequivocar $coringa_{MaxH}$. Note que o valor $Pedaços'_{MaxH}$ fica indefinido por enquanto.

4. *(Conjuntos H' e H'':)* $H' = H'' = \emptyset$.

5. *(Carteiras:)* Para todo $i \in \{1, \cdots, m\}$, $Carteira_i = \emptyset$. $CarteiraPenalidade = \emptyset$.

Para iniciar a simulação, S envia a lista $Etiquetas'$ e o conjunto $\{Pedaços'_c \mid c \in C\}$ para o adversário A controlando o conluio C.

Simulação dos Depósitos-Cobertura. Para simular os depósitos-cobertura, S inicializa as seguintes variáveis:

1. $CobDepsOK = 1$. Essa variável indica se todos os depósitos-cobertura foram feitos.
2. Para todo $c \in C \setminus \{m\}$, $FezDepCob_c = 0$. Essa variável indica se P_c desonesto fez seu depósito-cobertura.

O simulador prossegue da seguinte maneira:

1. Para todo $h \in H \setminus \{m\}$, S age como o oráculo f_{cr} e envia para A a mensagem $(depósito, h, m, \varphi^{cob}, \tau_m, q)$. Isso serve para simular a ocorrência do depósito-cobertura $\left[P_h \xrightarrow[q,\tau_m]{\varphi^{cob}} P_m\right]$ feito por um participante honesto.
2. Para todo $c \in C \setminus \{m\}$, o simulador S, agindo como o oráculo f_{cr}, espera receber a mensagem $(depósito, c, m, \varphi^{cob}, \tau_m, valor(q))$ de A controlando P_c:
   - Se a mensagem foi recebida, indicando que P_c fez seu depósito-cobertura, o simulador adiciona as q moedas na $Carteira_c$ e atualiza a variável $FezDepCob_c = 1$.
   - Se a mensagem não foi recebida, então S atualiza a variável $CobDepsOK = 0$.

Se $CobDepsOK = 0$, então nem todos os depósitos-cobertura foram feitos e, portanto, os que foram precisam ser reembolsados e a execução, terminada. Para isso S espera até a rodada $\tau_m + 1$, de reembolso dos depósitos-cobertura, para agir como o oráculo f_{cr}:

1. Para todo $c \in C \setminus \{m\}$ com $FezDepCob_c = 1$, S envia para A, controlando P_c, a mensagem $(reembolso, c, m, \varphi^{cob}, \tau_m, valor(q))$. Isso serve para reembolsar P_c. As q moedas são retiradas de $Carteira_c$.

2. Para todo $i \in H$ ou $i \in C \setminus \{m\}$ com $FezDepCob_i = 1$, S envia para A a mensagem $(reembolso, i, m, \varphi^{cob}, \tau_m, q)$, simulando o *broadcast* que f_{cr} faz das operações que modificam seu estado.

O simulador S encerra a execução e define sua saída como a de A. Nesse caso, nenhum participante reconstrói a saída z.

Se, por outro lado, $CobDepsOK = 1$, a execução continua. Nesse ponto, para todo $c \in C \setminus \{m\}$, $Carteira_c$ contém q moedas. A *CarteiraPenalidade* e o conjunto H' continuam vazios.

Simulação dos Depósitos-Escada. Para simular os depósitos-escada, S inicializa as seguintes variáveis:

1. $EscDepsOK = 1$. Essa variável indica se a fase de depósitos-escada foi concluída com sucesso. Isso não implica necessariamente que todos os depósitos tenham sido feitos. Note que, se $i+1 \in H$ é o índice do participante honesto mais "abaixo" na escada de depósitos, ou seja, se todos os depósitos abaixo de $\left[P_{i+1} \xrightarrow[i\cdot q, \tau_i]{\varphi_i^{esc}} P_i\right]$ envolverem apenas participantes desonestos, então P_i, por fazer parte do conluio C, pode já conhecer φ_i^{esc}. Por isso, P_i pode creditar $\left[P_{i+1} \xrightarrow[i\cdot q, \tau_i]{\varphi_i^{esc}} P_i\right]$ mesmo que nenhum outro depósito abaixo desse tenha sido feito. Nesse caso, $EscDepsOK = 1$, mesmo com a falta de alguns depósitos.
2. Para todo $c \in C$, $FezDepEsc_i = 0$. Essa variável indica se P_c fez seu depósito-escada.
3. Para todo $i \in \{1, \cdots, m-1\}$, $\hat{\tau}_i$ é inicializado com a rodada em que P_{i+1} deve fazer seu depósito-escada $\left[P_{i+1} \xrightarrow[i\cdot q, \tau_i]{\varphi_i^{esc}} P_i\right]$. Note que $\hat{\tau}_{m-1} < \hat{\tau}_{m-2} < \cdots < \hat{\tau}_1 < \tau_1 < \cdots < \tau_m$.

A simulação continua e, para $k = m-1, \cdots, 1$ o simulador S espera a rodada $\hat{\tau}_k$:

1. Se $EscDepsOK = 1$ e $k+1 \in H$, então S age como o oráculo f_{cr} e envia a mensagem $(depósito, k+1, k, \varphi_k^{esc}, \tau_k, kq)$ para A. Isso simula a ocorrência do depósito $\left[P_{k+1} \xrightarrow[k\cdot q, \tau_k]{\varphi_k^{esc}} P_k\right]$ de um participante honesto.
2. Se $k+1 \in C$, então o simulador espera receber de A, controlando P_{k+1}, a mensagem $(depósito, k+1, k, \varphi_k^{esc}, \tau_k, valor(k \cdot q))$:
   - Se a mensagem for recebida, então o simulador adiciona as $k \cdot q$ moedas na $Carteira_{k+1}$ e atualiza $FezDepEsc_{k+1} = 1$.
   - Se a mensagem não for recebida e existe $k' < k+1$ com $k' \in H$ (i.e., existe um participante honesto abaixo de $k+1$ na escada de depósitos) então S atualiza $EscDepsOK = 0$. Note que isso previne os próximos participantes honestos (simulados) de fazerem seus depósitos-escada. Esse é um comportamento estabelecido pela Definição 7.101 da construção escada.

Se, neste ponto da simulação, $EscDepsOK = 0$, então a simulação deve terminar sem que A reconstrua o segredo z. De fato, se $MinH \stackrel{def}{=} \min\{H\}$ é o índice do participante honesto mais abaixo na escada de depósitos (i.e., para todo $k < MinH$, $k \in C$), então P_{MinH} não fará seu depósito-escada. Dessa forma, A não terá acesso a $Pedaços'_{MinH}$ e, portanto, só poderá reconstruir o segredo z se conseguir violar o sigilo do esquema de comprometimento utilizado para gerar $Etiquetas'_{MinH}$.

Por isso, se $EscDepsOK = 0$, então o simulador deve cuidar do crédito dos depósitos-escada que podem ser creditados, do reembolso daqueles que não podem, além do reembolso dos depósitos-cobertura. Se durante essa simulação o adversário A conseguir creditar um depósito que ele não deveria, ou seja, acima do de P_{MinH}, então S deve terminar a execução com um erro. O caso de erro é um daqueles excepcionais em que S não consegue imitar A.

A simulação prossegue com S agindo como o oráculo f_{cr} para simular o crédito e o reembolso dos depósitos-escada.

Para todo $k \in \{1, \cdots, m-1\}$, na rodada τ_k, S cuida do crédito:

1. Se $k \in C$ e $k > MinH$, mas S receber de P_k a mensagem $(crédito, k+1, k, \varphi_k^{esc}, \tau_k, k \cdot q, W_k)$ tal que $W_k = \{Pedaços''_1, \cdots, Pedaços''_k\}$ é um conjunto de pedaços válidos[9], então A quebrou o sigilo do esquema de comprometimento. O simulador define sua saída como $Falha_{sigilo}$ e termina a execução. Note que **nenhum** depósito acima do de $MinH$ deveria poder ser creditado.

2. Se $k+1 < MinH$ (portanto $k+1, k \in C$), $FezDepEsc_{k+1} = 1$ e a mensagem $(crédito, k+1, k, \varphi_k^{esc}, \tau_k, k \cdot q, W_k)$ foi recebida, tal que $W_k = \{Pedaços''_1, \cdots, Pedaços''_k\}$ é um conjunto de pedaços válidos, então S envia para A a mensagem $(crédito, k+1, k, \varphi_k^{esc}, \tau_k, valor(k \cdot q), W_k)$. As $k \cdot q$ moedas são retiradas de $Carteira_{k+1}$. O simulador precisa ainda enviar a mensagem de *broadcast* $(crédito, k+1, k, \varphi_k^{esc}, \tau_k, k \cdot q, W_k)$ para A.

   Se $k+1 \neq m$, então $Carteira_{k+1}$ passa a ter q moedas, correspondentes aos depósito-cobertura de P_{k+1}. Senão, $Carteira_{k+1}$ se torna vazia.

Já na rodada $\tau_k + 1$, S cuida dos reembolsos:

1. Se $k+1 \in C$, $FezDepEsc_{k+1} = 1$ e $\left[P_{k+1} \xrightarrow[k \cdot q, \tau_k]{\varphi_k^{esc}} P_k\right]$ não foi reivindicado em τ_k, então S envia a mensagem $(reembolso, k+1, k, \varphi_k^{esc}, valor(k \cdot q))$ para A controlando P_{k+1}. As $k \cdot q$ moedas são retiradas de $Carteira_{k+1}$.

   Se $k+1 \neq m$, então $Carteira_{k+1}$ passa a ter apenas as q moedas dos depósitos-cobertura. Senão, $Carteira_{k+1}$ fica vazia.

2. Se $k+1 \in H$ (portanto $k+1 > MinH$), então S envia a mensagem $(reembolso, k+1, k, \varphi_k^{esc}, k \cdot q)$ para A, simulando o *broadcast* do reembolso de P_{k+1}.

[9] $Valida(Etiquetas_i, Pedaços''_i) = 1$ para todo $i \leq k$.

Na rodada τ_m, S se certifica que nenhum depósito-cobertura é creditado. Caso contrário, ele termina a execução com erro:

1. Na rodada τ_m, se $m \in C$ e, para algum $k \in \{1,\cdots,m-1\}$, P_m enviar a mensagem $(crédito, k, m, \varphi^{cob}, \tau_m, q, W_m)$, tal que $W_m = \{Pedaços''_1, \cdots, Pedaços''_m\}$ é um conjunto de pedaços válidos, então A quebrou o sigilo do esquema de comprometimento. O simulador S define sua saída como $Falha_{sigilo}$ e termina a execução.

Na rodada $\tau_m + 1$, S age como f_{cr} e simula o reembolso dos depósitos-cobertura:

1. Para todo $c \in C$, S envia a mensagem $(reembolso, k, m, \varphi^{cob}, \tau_m, valor(q))$ para A controlando P_c. As q moedas são retiradas de $Carteira_c$ que, depois disso, fica vazia.
2. Para todo $k \in \{1,\cdots,m-1\}$, S envia a mensagem $(reembolso, k, m, \varphi^{cob}, \tau_m, q)$ para A, simulando o *broadcast* feito por f_{cr}.

Até agora tratamos o caso em que $EscDepsOK = 0$. Após simular os reembolsos dos depósitos escada e cobertura, S define sua saída como a saída de A e encerra a execução. Note que, nesse caso, todas as moedas recebidas de A através dos depósitos-escada são devolvidas.

Por outro lado, se $EscDepsOK = 1$, a execução deve continuar. Note que, nesse caso, para todo $c \in C$ com $c > MinH$, é verdade que $FezDepEsc_c = 1$. Além disso, para todo $c \in C \setminus \{m\}$, se $FezDepEsc_c = 1$, então $Carteira_i$ contém $(c-1)\cdot q + q = c \cdot q$ moedas. Por outro lado, se $m \in C$, $Carteira_m$ contém $(c-1)\cdot q$ moedas (porque P_m não faz depósito-cobertura). Lembramos que o conjunto H' continua vazio.

Simulação do Crédito dos Depósitos-Escada. Para simular o crédito dos depósitos-escada, S inicializa as seguintes variáveis:

1. $MaxH \stackrel{def}{=} \max\{H\}$ é o índice do último participante honesto. Note que P_{MaxH} é o último participante honesto a creditar algum depósito (escada ou cobertura) na construção escada.
2. Para todo $k \in \{1,\cdots,m\}$, $Cpred_k$ representa o participante desonesto logo acima de k na escada de depósitos. Ou seja, $Cpred_k = c \in C$ tal que, para todo $c' \in C$, se $c' > k$ então $c' \geq c$. Se tal valor c não existir, então $Cpred_k = \infty$.
3. $CredEscOK = 1$. Essa variável indica se todos os depósitos-escada foram creditados com sucesso.
4. Para todo $k \in \{1,\cdots,m\}$, $CredEscOk_k = 0$. Essa variável indica se P_k creditou o depósito-escada $\left[P_{k+1} \xrightarrow[k\cdot q,\tau_k]{\varphi_k^{esc}} P_k\right]$. Por conveniência, definimos $CredEscOK_0 = 1$.
5. $W_0 = \emptyset$. Definimos W_i como o conjunto de pedaços revelados até o crédito do i-ésimo depósito (inclusive).

Antes de prosseguirmos com a descrição da simulação, vamos ressaltar que é nesta etapa de crédito de depósitos-escada que S contacta a funcionalidade f_{rec}^{esc}. De acordo com a Definição 7.104, existem três resultados possíveis para esse contato:

(i) *A saída reconstruída z é distribuída para todos os participantes:* Isso acontece quando $H' = \emptyset$. Esse tipo de contato é feito quando S percebe que A não agiu desonestamente. Mais precisamente, quando todo participante desonesto tomou parte na simulação do crédito dos depósitos-escada.

(ii) *Os participantes em H' são compensados e ninguém recebe z:* Isso acontece quando $0 < |H'| < |H|$. Na simulação, isso corresponde ao caso em que algum participante desonesto deixa de creditar um depósito feito por um participante honesto. Como consequência, tal participante honesto não pode creditar nenhum outro depósito e o segredo não é reconstruído.

(iii) *A saída reconstruída z é entregue a S que decide se aborta ou não:* Isso acontece quando $H' = H$. Na simulação, isso corresponde ao caso em que todos os depósitos-escada foram creditados e $m \in C$. Dessa forma, A decide se a saída é reconstruída ou não através do crédito dos depósitos-cobertura. S imita esse comportamento para decidir se aborta ou não.

A simulação prossegue com S agindo como f_{cr} para simular o crédito dos depósitos-escada. O que deve ser ressaltado dessa simulação é que, se $MaxH \neq m$ então P_{MaxH} deve creditar o depósito $\left[P_{(MaxH+1)} \xrightarrow[MaxH\cdot q, \tau_{MaxH}]{\varphi^{esc}_{MaxH}} P_{MaxH}\right]$, revelando seu $Pedaços'_{MaxH}$. Mas lembramos que esse valor não foi definido. Por isso, nesse caso, S precisa contactar f_{rec}^{esc} para obter z e desequivocar $Etiquetas'_{MaxH}$ de maneira adequada.

Dessa forma, para todo $k \in \{1, \cdots, m-1\}$ com $FezDepEsc_k = 1$, na rodada τ_k o simulador faz o seguinte:

1. Se $FezCredEsc_{k-1} = 1$, $k \in H$ e $k \neq MaxH$, então S atualiza $FezCredEsc_k = 1$, e envia a mensagem $(crédito, k+1, k, \varphi_k^{esc}, \tau_k, k \cdot q, W_k)$ para A, com $W_k = W_{k-1} \cup \{Pedaços'_k\}$, simulando o *broadcast* feito por f_{cr}. Lembramos que $Pedaços'_k$, nesse caso, é o resultado de se comprometer com uma cadeia aleatoriamente escolhida.

   Se $Cpred_k \neq \infty$, então, S adiciona k em H' e move q moedas da $Carteira_{Cpred_k}$ para a *CarteiraPenalidade*.

   O caso tratado neste item é aquele em que um participante honesto P_k credita um depósito. Ele só pode fazer isso se o depósito logo abaixo foi creditado (i.e., $FezCredEsc_{k-1} = 1$), pois é isso que o habilita a criar o conjunto W_k. Além disso, como $k \neq MaxH$, então $Pedaços'_k$ é um valor definido.

2. Se $FezCredEsc_{k-1} = 1$ e $k = MaxH$ (portanto $k \in H$), então:

   - O simulador S contacta f_{rec}^{esc} enviando a mensagem $(entrada, \{Pedaços'_c \mid c \in C\}, H, valor(|H| \cdot q))$. As $|H| \cdot q$ moedas são retiradas da *CarteiraPenalidade*.
   - O simulador S recebe $(saída, z)$ de f_{rec}^{esc} e tem que decidir se aborta ou não.

- O simulador S escolhe $p_k = z \oplus \bigoplus_{i \neq k} Pedaços'_i$. Então S invoca o algoritmo $w_k = Desequivoca(info_k, p_k)$ e define $Pedaços'_k = (p_k, w_k)$. Finalmente, o simulador credita o depósito-escada de P_k, enviando a mensagem $(crédito, k+1, k, \varphi_k, \tau_k, k \cdot q, W_k)$ para A, com $W_k = W_{k-1} \cup \{Pedaços'_k\}$.

Este é o caso em que P_{MaxH} credita seu depósito. Como já argumentamos anteriormente, S precisa contactar f_{rec}^{esc} para obter z e gerar $Pedaços'_{MaxH}$ adequadamente. Isso significa escolher p_{MaxH} de modo que a soma de todos os $Pedaços'_i$ seja igual a z, e w_k de modo que $Etiquetas'_{MaxH}$ seja um comprometimento válido.

Ressaltamos que, nesse ponto, $H = H' \cup \{MaxH\}$. De fato, todos os outros participantes honestos já creditaram seus depósitos, uma vez que, por definição, P_{MaxH} é o último honesto a fazer isso. De acordo com o passo anterior, no entanto, toda vez que um participante honesto credita um depósito e existe algum desonesto acima dele (i.e., com índice maior), ele é acrescentado em H'. Mas, do fato que $k = MaxH$, podemos concluir que P_m é desonesto pois, se não fosse, $MaxH = m$. Então podemos concluir que H' contém, de fato, todos os índices dos participantes honestos *exceto* $MaxH$.

3. Se $k \in C$:

   - Se S receber uma mensagem $(crédito, k+1, k, \varphi_k, \tau_k, k \cdot q, W'_k)$ de A em que cada pedaço em W'_k é valido, mas $W'_k \neq \{Pedaços'_1, \cdots, Pedaços'_k\}$, então A quebrou a não-ambiguidade do esquema de comprometimento. Então S define sua saída como $Falha_{ambg}$ e encerra a simulação.
   - Se S receber uma mensagem $(crédito, k+1, k, \varphi_k, \tau_k, k \cdot q, W'_k)$ de A em que cada pedaço em W'_k é válido, mas existe algum $h \in H$ tal que $h < k$ e $FezCredEsc_h = 0$, então A quebrou o sigilo do esquema de comprometimento. Então S define sua saída como $Falha_{sigilo}$ como saída e encerra a simulação.
   - Se S receber uma mensagem $(crédito, k+1, k, \varphi_k, \tau_k, k \cdot q, W'_k)$ de A em que cada pedaço em W'_k é válido, então S envia $(crédito, k+1, k, \varphi_k, \tau_k, k \cdot q, W'_k)$ para A, simulando o *broadcast* de f_{cr}, e $(crédito, k+1, k, \varphi_k, \tau_k, valor(k \cdot q))$ para A controlando P_k. As $k \cdot q$ são obtidas como a seguir:

     - Se $Cpred_k \neq \infty$, então as $k \cdot q$ são retiradas da $Carteira_{Cpred_k}$.
     - Senão, q moedas são retiradas de cada $Carteira_c$ com $c \in C$ (os depósitos-cobertura de P_c) e $|H'| \cdot q$ moedas são retiradas da $CarteiraPenalidade$. Note que $|C| + |H'| = k$, por isso S conseguirá as $k \cdot q$ moedas necessárias.

     Além disso, S atualiza $FezCredEsc_k = 1$ e $W_k = W'_k$. Se $Cpred_k = \infty$, então podemos concluir que $m \in H$ (portanto $MaxH = m$). Por isso o simulador S só precisa revelar $Pedaços'_m$ ($Pedaços'_{MaxH}$) na etapa de crédito dos

depósitos-cobertura. Então S prossegue contactando f_{rec}^{esc} através da mensagem $(Entrada, \{Pedaços'_c \mid c \in C\}, \emptyset, valor(0))$. Ele recebe como resposta a mensagem $(saída, z)$. Note que, nesse caso, como $H' = \emptyset$, não cabe a S decidir se vai abortar ou não. Pela Definição 7.104, esse é o caso em que as saídas são distribuídas para todos.

Note que, nesta chamada à f_{rec}^{esc}, $H' = \emptyset$. Isso porque, grosso modo, H' representa os participantes honestos que devem ser compensados caso A aja desonestamente. Mas P_k era o último participante que poderia agir desonestamente, pois, $Cpred_k = \infty$. Dessa forma, nenhum participante precisará ser compensado.

4. Se S não receber nenhuma mensagem de A, então ele atualiza $FezCredEsc_k = 0$. Além disso, se $k+1 \in H$, $H' \neq \emptyset$ e f_{rec}^{esc} ainda não foi contactada, S envia a mensagem $(Entrada, \{Pedaços_c \mid c \in C\}, H', valor(|H'| \cdot q))$ para f_{rec}^{esc}. As $|H'| \cdot q$ moedas são retiradas da *CarteiraPenalidade*.

   Note que, em particular, $k+1 \notin H'$. O simulador contacta f_{rec}^{esc} agora porque o conjunto H' já está completo. Ou seja, todos os participantes honestos que devem ser compensados já foram adicionados em H'. Esses participantes são os que revelaram seus pedaços através dos depósitos-escada. Como $0 < |H'| < H$, pela Definição 7.104, S não obterá a saída z nem terá a chance de escolher se aborta ou não. Essa condição implica que todos os participante de H' serão compensados e a execução, terminada.

   Ressaltamos que S não vai simular o crédito de nenhum participante honesto acima na escada de depósitos (índice $i > k+1$). Isso se justifica porque tal participante, na execução real da construção escada, não conheceria $Pedaços_k$ e não teria como creditar seu depósito. O simulador S precisa imitar esse comportamento.

A simulação prossegue e, na rodada $\tau_k + 1$, S cuida dos possíveis reembolsos dos depósitos-escada:

1. Se $FezCredEsc_k = 0$ e $k+1 \in H$, então S envia a mensagem $(reembolso, k+1, k, \varphi_k^{esc}, \tau_k, k \cdot q)$ para A, simulando o *broadcast* de um reembolso para um participante honesto.
2. Se $FezDepEsc_k = 0$ e $k+1 \in C$, então S envia a mensagem $(reembolso, k+1, k, \varphi_k^{esc}, \tau_k, valor(k \cdot q))$ para A controlando P_{k+1}. As $k \cdot q$ moedas são retiradas da $Carteira_{k+1}$.

Simulação do Crédito dos Depósitos-Cobertura. Nesta parte da simulação, S lida com o crédito dos depósitos-cobertura. É importante ressaltar que, após o crédito dos depósitos-escada, se $m \notin H$ e todos os participantes honestos creditaram depósitos-escada, então f_{rec}^{esc} foi

contactada e está esperando S definir se vai abortar ou não. O simulador vai decidir isso com base no comportamento de A.

A simulação do crédito dos depósitos-cobertura é dividida em dois cenários com base na honestidade de P_m.

Se $m \in H$, então:

1. Se $FezCredEsc_{m-1} = 1$, então S define $p_m = z \oplus \bigoplus_{i \neq m} Pedaços'_i$. Então S invoca o algoritmo $w_m = Desequivoca(info_m, p_m)$ e define $Pedaços'_m = (p_m, w_m)$. Então, agindo como o oráculo f_{cr}, S envia, para todo $i \in \{1, \cdots, m-1\}$, a mensagem $(crédito, i, m, \varphi^{cob}, \tau_m, q, W_m)$ para A, em que $W_m = W_{m-1} \cup \{Pedaços'_m\}$, simulando o *broadcast* feito por f_{cr}.
2. Se $FezCredEsc_{m-1} = 0$, então S simula o reembolso dos depósitos-cobertura. Para todo $i \in \{1, \cdots, m-1\}$, S envia a mensagem $(reembolso, i, m, \varphi^{cob}, \tau_m, q)$ para A e, para todo $c \in C$, S envia ainda a mensagem $(reembolso, c, m, \varphi^{cob}, \tau_m, valor(q))$ para A controlando P_c. As q moedas são retiradas de $Carteira_c$.

Por outro lado, se $m \in C$, S inicializa o conjunto $H'' = H$ e, para todo $k \in \{1, \cdots, m-1\}$, define $FezCredCob_k = 0$ e segue a simulação com os passos abaixo:

1. Para todo $k \in \{1, \cdots, m-1\}$, o simulador espera receber a mensagem $(crédito, k, m, \varphi^{cob}, \tau_m, W')$ de A na rodada τ_m:
   - Se todo $w \in W'$ é um pedaço válido, mas $W' \neq \{Pedaços'_1, \cdots, Pedaços'_m\}$, então A quebrou a não-ambiguidade do esquema de comprometimento. Nesse caso S define sua saída como $Falha_{ambg}$ e termina a simulação.
   - Se $W' = \{Pedaços'_1, \cdots, Pedaços'_m\}$, então S atualiza $FezCredCob_k = 1$ e $H' = H' \setminus \{k\}$.
2. Se existe $k \in \{1, \cdots, m-1\}$ tal que $FezCredCob_k = 1$, significando que *algum* depósito-cobertura foi creditado, então S sinaliza à f_{rec}^{esc} que a execução deve continuar. Ele envia a mensagem $(continue, H'')$ para f_{rec}^{esc} e recebe como resposta a mensagem $(reembolso, valor((|H| - |H''|) \cdot q))$. S adiciona as $(|H| - |H''|) \cdot q$ moedas na $Carteira_m$. Então, para cada k tal que $FezCredCob_k = 1$, S envia $(crédito, k, m, \varphi^{cob}, \tau_m, q, W')$ para A, simulando o *broadcast* dos créditos, e $(crédito, k, m, \tau_m, valor(q))$ para P_m. As q moedas são retiradas de $Carteira_k$, se $k \in C$, ou de $Carteira_m$, se $k \in H$.
3. No entanto, se, para todo $k \in \{1, \cdots, m-1\}$, $FezCredCob_k = 0$ então S contacta f_{rec}^{esc} com a mensagem $(aborte, \perp)$.
4. Na rodada $\tau_m + 1$, S lida com os depósitos-cobertura não creditados. Para isso, para todo $k \in \{1, \cdots, m-1\}$ tal que $FezCredCob_k = 0$, S procede como a seguir:
   - Se $k \in H$, então S envia $(reembolso, k, m, \varphi^{cob}, \tau_m, q)$ para A, simulando o *broadcast* de um reembolso para um participante honesto.

- Se $k \in C$, então S retira q moedas da $Carteira_k$ e envia a mensagem $(reembolso, k, m, \varphi^{cob}, \tau_m, valor(q))$ para A controlando P_k.

No final da simulação, S copia a saída A.

Para encerrar a prova do Lema 7.105, precisamos argumentar que a saída do simulador S é indistinguível da saída do adversário A. De fato, a saída de S é exatamente a de A *exceto* em duas ocasiões: quando A consegue quebrar o sigilo ou a não-ambiguidade do esquema de comprometimento utilizado. Nesses casos, S tem como saída $Falha_{sigilo}$ e $Falha_{ambg}$, respectivamente.

No entanto, os cenários em que S falha em imitar A só podem acontecer com probabilidade desprezível, segundo a Definição 3.14 (sobre esquemas de comprometimento). Por isso, apoiado nas propriedades do esquema de comprometimento utilizado, concluímos que a saída de S é indistinguível da de A, embora elas não sejam idênticas.

Outra propriedade importante do simulador S, e essencial para que a construção escada seja considerada segura, é o fato de que S não precisa criar moedas. Isto é, S consegue conduzir a simulação apenas com as moedas recebidas de A. Lembramos que essa é uma propriedade necessária porque, segundo a definição do modelo de computação, apenas o ambiente $\mathscr{E}$ pode criar moedas.

□

7.5 Implementação de f_{cr} com *Bitcoin*

7.5.1 Visão Geral

O que falta para possibilitar uma implementação concreta da construção escada é apresentarmos uma implementação da funcionalidade f_{cr}. Para isso, utilizamos transações *Bitcoin*.

Um depósito condicional $\left[P_i \xrightarrow[c,\tau]{\varphi} P_j\right]$ é implementado através de duas transações. A primeira delas é a $Txn^{cr}_{i\to j}$, esquematizada na Figura 7.2. Ela é responsável por transferir as c moedas para P_j respeitando as condições de f_{cr}. A condição mais óbvia que f_{cr} impõe é a de que P_j precisa fornecer uma entrada adequada para o circuito φ. Além disso, f_{cr} impõe que *apenas* P_j deva ser capaz de reivindicar o depósito. Essas duas condições são aplicadas através do *script* da saída de $Txn^{cr}_{i\to j}$.

A segunda transação necessária para implementar f_{cr} é a $Txn^{reem}_{i\to j}$, esquematizada na Figura 7.4. Ela serve para reembolsar P_i quando P_j deixar de reivindicar $Txn^{cr}_{i\to j}$ antes da rodada τ especificada. A novidade nessa transação é o uso do campo *locktime* impedir um reembolso prematuro, isto é, antes da rodada τ.

Outra característica importante de f_{cr} é seu *sincronismo*. A estratégia de BENTOV e KUMARESAN (2014) para implementá-lo é usar a *blockchain* como um *relógio global* em que a unidade básica de tempo é o bloco. Ela se sustenta na premissa de que a taxa de geração de

Figura 7.2: A Transação $Txn^{cr}_{i\to j}$

$Txn^{cr}_{i\to j}$

Saída
Valor: c **btc**
Script de Saída: **Parâmetros:** $[Txn]$, p, A_{Sk_i}, A_{Sk_j} **Início:** **retorne** $VrfAssinatura(Pk_j,[Txn],A_{Sk_j}) = 1$ $\wedge\big(VrfAssinatura(Pk_i,[Txn],A_{Sk_i}) = 1 \vee \varphi(p) = 1\big)$

Fonte: Elaborada pelo autor.

blocos é aproximadamente constante[10]. Dessa forma, para representar as rodadas, precisa-se de duas medidas. A primeira delas é a *duração* de uma rodada em blocos, que chamaremos de ΔT. A segunda delas é um parâmetro de segurança T_{seg} que representa a quantidade mínima de confirmações para que um bloco seja considerado pertencer à *blockchain*. Dessa forma, para que uma transação seja feita *dentro* de uma determinada rodada, ela deve aparecer em bloco *antes* dos T_{seg} últimos blocos dessa rodada.

Por conveniência, chamaremos de T_0 a quantidade de blocos presentes na *blockchain* imediatamente antes do início do protocolo. Dessa maneira, a rodada τ_i da implementação *Bitcoin* de f_{cr} começa quando a *blockchain* contiver $T_0 + \tau_i \cdot \Delta T$ blocos e uma transação só é considerada ter sido publicada em τ_i se ela aparecer em um bloco entre os "instantes" $(T_0 + \tau_i \cdot \Delta T)$ e $(T_0 + (\tau_i + 1) \cdot \Delta T - T_{seg})$.

Como um último detalhe dessa visão geral, ressaltamos que os *broadcasts* especificados em f_{cr} são realizados naturalmente pelo protocolo *Bitcoin*. Eles correspondem à própria publicação de transações na *blockchain*.

Nas seções seguintes vamos descrever mais detalhadamente as transações $Txn^{cr}_{i\to j}$ e $Txn^{reem}_{i\to j}$ considerando o cenário em que o depósito $\left[P_i \xrightarrow[c,\tau]{\varphi} P_j\right]$ é feito. Em seguida vamos exibir a estratégia completa da implementação *Bitcoin* de f_{cr}.

7.5.2 As Transações $Txn^{cr}_{i\to j}$

Como ilustrado na Figura 7.2, $Txn^{cr}_{i\to j}$ transfere c moedas para P_j e só pode ser reivindicada se as condições do depósito $\left[P_i \xrightarrow[c,\tau]{\varphi} P_j\right]$ forem respeitadas. Essas condições são aplicadas pelo *script* da única saída de $Txn^{cr}_{i\to j}$. Ele recebe como argumentos a versão simplificada da transação que reivindica $Txn^{cr}_{i\to j}$ ([txn]), uma entrada para o circuito φ (p) e duas assinaturas de [txn] supostamente válidas, uma gerada com a chave secreta de P_j (A_{Sk_j}) e outra com a chave secreta de P_i (A_{Sk_i}).

O *script* de saída é projetado de modo a permitir $Txn^{cr}_{i\to j}$ ser reivindicada de duas maneiras.

[10]Um bloco é gerado, em média, a cada 10 minutos.

Figura 7.3: A Transação $Txn^{cred}_{i\to j}$

$Txn^{cred}_{i\to j}$
Entrada
Saída Referenciada: a única saída de $Txn^{cr}_{i\to j}$
Script de Entrada: **Início:** **forneça** $[Txn^{cred}_{i\to j}], p, \perp, Assina(Sk_j, [Txn^{cred}_{i\to j}])$ para a saída referenciada.

Fonte: Elaborada pelo autor.

A primeiras delas é através do fornecimento de uma assinatura válida feita com a chave secreta Sk_j (do destinatário P_j) e de uma entrada p que satisfaça o circuito φ. Essa é a única maneira pela qual P_j deve ser capaz de reivindicar essa transação e é como a *etapa de crédito* (E2 da Definição 7.100) de f_{cr} é implementada. Ilustramos, na Figura 7.3, como uma transação $Txn^{cred}_{i\to j}$ pode reivindicar $Txn^{cr}_{i\to j}$ dessa maneira. Ela precisa apenas satisfazer $Txn^{cr}_{i\to j}$ fornecendo uma entrada p para o *script* φ e uma assinatura com a chave secreta de P_j. Nenhuma restrição é feita em relação ao seu *script* de saída[11].

A outra maneira de reivindicar $Txn^{cr}_{i\to j}$ é fornecendo duas assinaturas válidas: uma feita com a chave Sk_j e outra com a chave Sk_i. Essa é a forma pela qual a *etapa de reembolso* (etapa E3 da Definição 7.100) de f_{cr} é implementada, como veremos adiante.

Como mostra a Figura 7.2, alguns valores utilizados pelo *script* de saída de $Txn^{cr}_{i\to j}$ não são recebidos como argumentos. É o caso do circuito φ e das chaves públicas Pk_i e Pk_j. De fato, eles devem ser introduzidos diretamente no código (*hardcoded*) durante a criação de $Txn^{cr}_{i\to j}$ e, portanto, precisam ser conhecidos naquele momento.

Ressaltamos que não são impostas restrições sobre a origem das moedas transferidas por $Txn^{cr}_{i\to j}$. Por essa razão, omitimos a especificação da entrada (ou entradas) de $Txn^{cr}_{i\to j}$ na Figura 7.2.

7.5.3 As Transações $Txn^{reem}_{i\to j}$

A segunda transação criada para implementar f_{cr} é a $Txn^{reem}_{i\to j}$, esquematizada na Figura 7.4. Ela serve para reembolsar o criador de $Txn^{cr}_{i\to j}$ (P_i) quando $Txn^{cr}_{i\to j}$ não for creditada até a rodada limite τ. A única entrada de $Txn^{reem}_{i\to j}$ referencia a saída de $Txn^{cr}_{i\to j}$. O *script* dessa entrada fornece as duas assinaturas ($Assina([Txn^{reem}_{i\to j}], Sk_i)$ e $Assina([Txn^{reem}_{i\to j}], Sk_j)$) que possibilitam $Txn^{cr}_{i\to j}$ ser reivindicada sem a apresentação de uma entrada para φ .

A transação $Txn^{reem}_{i\to j}$ possui o campo *locktime* definido com o início da rodada τ, isto é, $T_0 + \tau \cdot \Delta T$. Isso serve para impedir que ela seja publicada antes do esperado e, portanto, que o depósito de P_i seja reembolsado antes da hora.

[11] Isso significa que o destinatário pode gastar essas moedas como quiser.

Figura 7.4: A Transação $Txn^{reem}_{i\to j}$

$Txn^{reem}_{i\to j}$
locktime: $T_0 + \tau \cdot \Delta T$ (rodada τ)
Entrada
Saída Referenciada: a única saída de $Txn^{cr}_{i\to j}$
Script de Entrada: **Início:** **forneça** $[Txn^{reem}_{i\to j}], \perp, Assina(Sk_i, [Txn^{reem}_{i\to j}])$, $Assina(Sk_j, [Txn^{reem}_{i\to j}])$ para a saída referenciada.

Fonte: Elaborada pelo autor.

Os participantes P_i e P_j precisam cooperar para construir $Txn^{reem}_{i\to j}$, já que ela possui assinaturas geradas com as chaves secretas dos dois. Essa cooperação deve acontecer *antes* de $Txn^{cr}_{i\to j}$ ser publicada, pois $Txn^{reem}_{i\to j}$ é a única garantia de P_i ser reembolsado no caso de P_j não reivindicar o depósito.

7.5.4 Juntando as Peças: o Protocolo Π_{cr}

Juntando tudo o que foi discutido sobre as transações Txn^{cnr} e $Txn^{reem}_{i\to j}$, definimos o protocolo Π_{cr} para computar f_{cr} através do protocolo *Bitcoin* a seguir.

Definição 7.106 (Protocolo Π_{cr} para computar f_{cr} baseado no protocolo *Bitcoin*- BENTOV e KUMARESAN (2014))**.** *Seja $\{P_1, \cdots, P_m\}$ um conjunto de participantes.*

E1. (Etapa de Depósito.) O objetivo desta etapa é criar as transações $Txn^{cr}_{i\to j}$ e $Txn^{reem}_{i\to j}$ para implementar o depósito $\left[P_i \xrightarrow[c,\tau]{\varphi} P_j\right]$. Nesta etapa $Txn^{cr}_{i\to j}$ é publicada na rede Bitcoin.

Para isso,

- *P_i recebe de P_j uma chave pública Pk_j,*
- *P_i cria a transação $Txn^{cr}_{i\to j}$, conforme a Figura 7.2, utilizando sua chave pública Pk_i, a chave Pk_j recebida de P_j e o script φ,*
- *P_i cria a transação $Txn^{reem}_{i\to j}$ de acordo com a Figura 7.4 e envia a versão simplificada $[Txn^{reem}_{i\to j}\ ]$ para P_j,*
- *P_j gera a assinatura $A_{Sk_j} = Assina([Txn^{reem}_{i\to j}], Sk_j)$ e a envia para P_i,*
- *P_i verifica se $VrfAssinatura([Txn^{reem}_{i\to j}], Pk_j, A_{Sk_j}) = 1$, acrescenta A_{Sk_j} no script de entrada de $Txn^{reem}_{i\to j}$ e publica $Txn^{cr}_{i\to j}$ na rede Bitcoin.*

E2. (Etapa de Crédito.) O objetivo desta etapa é fazer com que a transação $Txn^{cr}_{i\to j}$ seja reivindicada com o fornecimento de uma entrada p para o script φ.

Para isso, P_j publica na rede Bitcoin a transação $Txn^{cred}_{i\to j}$ reivindicando $Txn^{cr}_{i\to j}$, com p no script da sua entrada, conforme a Figura 7.3.

E3. (Fase de Reembolso.) O objetivo desta etapa é fazer com que P_i tenha seu depósito reembolsado através da transação $Txn^{reem}_{i\to j}$.

Para isso, na rodada τ (quando a blockchain contiver $T_0 + \tau \cdot \Delta T$ blocos) e se $Txn^{cr}_{i\to j}$ não tiver sido reivindicada, P_i adiciona sua assinatura ($Assina([Txn^{reem}_{i\to j}], Sk_i)$) ao script da entrada de $Txn^{reem}_{i\to j}$ e publica essa transação na rede Bitcoin.

A Definição 7.106 não traz grandes novidades em relação ao que já discutimos sobre as transações nas seções anteriores. Ela serve principalmente para registrar o momento em que essas transações são criadas e reforçar o papel que cada uma delas tem no protocolo de maneira geral.

Na etapa de crédito (E1), ambas $Txn^{cr}_{i\to j}$ e $Txn^{reem}_{i\to j}$ devem ser criadas. A transação $Txn^{cr}_{i\to j}$ só deve ser publicada, no entanto, se P_i obtiver uma assinatura válida da versão simplificada de $Txn^{reem}_{i\to j}$ ($[Txn^{reem}_{i\to j}]$) feita por P_j. Como já mencionamos, é isso que garante a P_i a possibilidade de ser reembolsado caso P_j pule a etapa E2. Um ponto importante a ser notado é que, após ser assinada por P_j, a transação $Txn^{reem}_{i\to j}$ torna-se "imutável". Isto é, P_i não pode alterar nenhum dos campos de $Txn^{reem}_{i\to j}$, em particular o *locktime*, sem invalidá-la[12].

Outro ponto importante a ser observado é que, na Definição 7.100, definimos φ como um *circuito*, enquanto que na Definição 7.106 o tratamos como um *script Bitcoin*. De fato, essa conversão de circuito para *script* precisa ser feita para que f_{cr} admita uma implementação baseada no protocolo *Bitcoin*. Por essa razão, como discutiremos mais adiante, a expressividade da linguagem disponibilizada para programar esses *scripts*, bem como a flexibilidade da estrutura das transações *Bitcoin*, são fatores críticos para que essas implementações sejam aplicáveis na prática.

Por fim, ressaltamos que é recomendado que em cada execução do protocolo da Definição 7.106 sejam utilizados pares novos de chaves públicas e secretas ((Pk_i, Sk_i) e (Pk_j, Sk_j)). Isso serve para impedir que informações de execuções passadas sejam reutilizadas. Isso é especialmente importante porque todas as informações trocadas entre P_i e P_j são, a princípio, públicas.

7.6 Análise da Construção Escada e sua Implementação *Bitcoin*

Como vimos, a construção escada permite uma implementação *Bitcoin* bastante simples. Ela é realizada, basicamente, a partir de três transações *Bitcoin* ($Txn^{cr}_{i\to j}$, $Txn^{reem}_{i\to j}$ e $Txn^{cred}_{i\to j}$). Ela atinge a noção de justiça monetária da Definição 5.95 e vai um pouco além, compensando alguns participantes honestos mesmo que a justiça (tradicional) não tenha sido quebrada.

Como aspecto negativo dessa construção, ressaltamos a *assimetria* dos investimentos dos participantes. De fato, o participante P_i precisa pagar até $q + (i-1) \cdot q$ moedas, com $i \neq m$, para fazer seus depósitos escada e cobertura. Mesmo que no final do protocolo os participantes

[12] Isso porque a assinatura contribui para a imutabilidade da transação. No entanto, alguns campos não são considerados pela assinatura, como a entrada na qual ela será incluída.

recebam o mesmo valor de compensação, essa assimetria faz uma distinção arbitrária entre eles logo no início da execução.

Outro ponto a ser ressaltado é a falta de *escalabilidade* da construção escada. Para uma execução com m participantes, o protocolo dura $2 \cdot m$ rodadas[13]. Cada uma dessas rodadas é caracterizada pela publicação de uma ou mais transações. De fato, uma transação só pode ser publicada em *uma rodada específica* e pode depender que outras transações já tenham sido publicadas. Por essa razão, as $2 \cdot m$ rodadas totais do protocolo só podem ser executadas sequencialmente. O *tempo* que isso leva depende dos parâmetros ΔT e T_{seg} mas, se levarmos consideração que $\Delta T \geq T_{seg}$ e que o valor recomendado pela comunidade *Bitcoin* para T_{seg} é de seis blocos, podemos concluir que cada rodada deve durar pelo menos 60 minutos[14]. Nesse cenário, uma execução envolvendo apenas dois participantes ($m = 2$) duraria pelo menos 4 horas.

[13]No caso em que o protocolo é executado honestamente até o fim: 1 (criação dos depósitos-cobertura) + $(m-1)$ (criação dos depósitos-escada) + $(m-1)$ (reivindicação dos depósitos-escada) + 1 (reivindicação dos depósitos-cobertura) = $2 \cdot m$.

[14]No caso em que $\Delta T = T_{seg}$ e $T_{seg} = 6$. Como um bloco é gerado, em média a cada 10 minutos, ΔT dura 60 minutos.

8 RECONSTRUÇÃO JUSTA ATRAVÉS DE COMPROMETIMENTO MONETÁRIO MULTILATERAL

8.1 Visão Geral

Como vimos no capítulo anterior, a construção escada se baseia em chamadas à f_{cr}. Essa funcionalidade, por sua vez, é usada para que os participantes se comprometam em revelar seus elementos da lista *Pedaços*. No entanto, a organização sequencial das chamadas à f_{cr} resulta em um protocolo com número linear (em m) de rodadas, provocando problemas como a falta de escalabilidade.

Nesta seção apresentamos uma alternativa à construção escada baseada na funcionalidade f_{cmm} para **c**omprometimento **m**onetário **m**ultilateral. Uma *única* instância dessa funcionalidade permite a todos os participantes se comprometerem *simultaneamente* a revelar seus pedaços. Em contraste, a construção escada, por exemplo, necessita de várias instâncias de f_{cr} organizadas em sequência para atingir o mesmo efeito.

A funcionalidade f_{cmm} tem 3 etapas principais: a dos depósitos-garantia, a da revelação dos pedaços e a do resgate de depósitos e penalização. Na primeira delas, *todos* os participantes fazem investimentos monetários, através dos chamados depósitos-garantia e, com isso, se comprometem a agir honestamente durante o restante da execução. Agir honestamente significa, na etapa de revelação, publicar os elementos da lista *Pedaços*. Somente assim é possível recuperar os investimentos monetários iniciais através do resgate dos depósitos-garantia, feito na etapa seguinte. Na etapa de penalização, os participantes desonestos, os que não revelaram seus pedaços, perdem a chance de reaver seus investimentos iniciais. Esses investimentos são divididos entre os demais como forma de compensação.

A partir de f_{cmm}, é possível definir uma funcionalidade f_{rec}^{cmm} para reconstrução justa que admita um protocolo Π_{rec}^{cmm} com número *constante* de rodadas. A ideia geral desse protocolo é que cada participante utilize f_{cmm} para investir $(m-1)\cdot q$ moedas e se comprometer a revelar seu pedaço. Essa revelação deve acontecer em uma rodada predefinida τ e é a única maneira pela qual um participante recupera seu investimento. Se ele agir desonestamente, cada um dos $(m-1)$ demais participantes recebe q moedas.

Ressaltamos que nossa proposta é semelhante à funcionalidade $F_{ML}^{\star}$ de KUMARESAN e BENTOV (2014), mas foi desenvolvida de maneira independente. De fato, o trabalho de KUMARESAN e BENTOV (2014) difere do nosso porque, entre outras coisas, admite uma versão modificada do *Bitcoin*[1], a qual contribui para facilitar a implementação de $F_{ML}^{\star}$. Nossa proposta não depende dessa modificação.

A apresentação da nossa proposta será dividida da seguinte forma. Primeiramente definiremos a funcionalidade f_{rec}^{cmm} para reconstrução justa. Dessa forma deixaremos claro como o problema da reconstrução justa será tratado no restante do capítulo. Em seguida exibiremos

[1] A *Proposta 2* discutida na Seção 2.2.

o protocolo Π_{rec}^{cmm} para computar f_{rec}^{cmm} no modelo f_{cmm}-híbrido. Apresentaremos também uma implementação de Π_{rec}^{cmm} utilizando transações *Bitcoin*. Por fim, faremos uma análise da correção e justiça da nossa proposta, bem como de outros pontos relevantes.

8.2 A Funcionalidade f_{rec}^{cmm} para Reconstrução Justa

Nesta seção, consideramos o problema da reconstrução justa da mesma maneira de quando apresentamos f_{rec}^{esc}. Isto é, cada participante possui um pedaço de um valor secreto z e a lista *Etiquetas* é conhecida por todos. O objetivo desses participantes é entrar em uma computação multilateral (monetariamente) justa para possibilitar a reconstrução do segredo z a partir das listas *Etiquetas* e *Pedaços*.

Poderíamos continuar trabalhando sobre a funcionalidade f_{rec}^{esc}, mas consideramos que ela dificulta a especificação e prova de correção de implementações diferentes da construção escada. Isso porque, como comentamos anteriormente, a funcionalidade f_{rec}^{esc} possui algumas características que fazem mais sentindo quando interpretadas sob a luz dessa construção. Portanto, julgamos conveniente para o restante da apresentação definir outra funcionalidade para reconstrução justa. Fazemos isso a seguir.

Definição 8.107 (Funcionalidade f_{rec}^{cmm} para reconstrução justa)**.** *Seja $\{P_1,\cdots,P_m\}$ o conjunto de participantes, $C \stackrel{def}{=} \{c_1,\cdots,c_t\} \subset \{1,\cdots,m\}$ o conluio dos participantes controlado pelo adversário S, $H \stackrel{def}{=} \{1,\cdots,m\} \setminus C$ o conjunto dos participantes honestos e q o valor da multa por aborto. Seja ainda $(Divide, Reconstrói, Verifica)$ um $EVCS_{m,m}$ implementado como na Definição 6.97. Definimos a funcionalidade f_{rec}^{cmm} para reconstrução justa como a seguir.*

E1. (Entradas:) Nesta etapa os participantes honestos e o adversário, em nome dos desonestos, fornecem seus elementos da lista Pedaços e fazem seus investimentos financeiros iniciais.

Mais precisamente, para todo $h \in H$, P_h envia a mensagem $(entrada, h, Etiquetas, Pedaços_h, valor((m-1)\cdot q))$ para f_{rec}^{cmm}. O adversário S envia a mensagem $(entrada, Etiquetas, \{Pedaços_{c_1},\cdots,Pedaços_{c_t}\}, valor(|C|\cdot(m-1)\cdot q))$.

E2. (Retorno de investimento:) Nesta etapa o investimento dos participantes honestos é devolvido.

Para isso, f_{rec}^{cmm} envia a mensagem $(reembolso, h, valor((m-1)\cdot q))$ para P_h para todo $h \in H$.

E3. (Cálculo e entrega das saídas:) Nesta etapa o segredo z é reconstruído a partir das listas Etiquetas e Pedaços fornecidas como entrada. Esse valor é entregue ao adversário S que então decide sobre a continuação da execução. Se S decidir abortar, ele paga um multa a cada participante honesto. Mais precisamente,

- *O valor $z = Reconstrói(Etiquetas, Pedaços)$ é calculado a partir da invocação do algoritmo de reconstrução do $EVCS_{m,m}$,*

- *A mensagem* $(saída, z)$ *é entregue ao adversário* S,
- *Se o adversário* S *responder* $(continue)$, *então:*
  - *(i) a mensagem* $(saída, z)$ *é enviada para* P_h, *para todo* $h \in H$,
  - *(ii) a mensagem* $(reembolso, valor(|C| \cdot (m-1) \cdot q))$ *é enviada para* S,
- *Se o adversário* S *responder* $(aborte, C')$, *com* $C' \subseteq C$, *então:*
  - *(i) a mensagem* $(compensação, valor(|C'| \cdot q))$ *é enviada para* P_i, *para todo* $i \in \{1, \cdots, m\} \setminus C' = H \cup C \setminus C'$.
  - *(ii) a mensagem* $(reembolso, valor(restante))$ *é enviada para* S, *em que* $restante \stackrel{def}{=} (|C| \cdot (m-1) - |C'| \cdot (m - |C'|)) \cdot q$ *é a quantidade de moedas que devem ser devolvidas ao adversário.*

A funcionalidade f_{rec}^{cmm} é dividida em 3 etapas básicas: a da apresentação das entradas (E1), a do retorno dos investimentos dos participantes honestos (E2) e a da entrega das saídas (E3). Na etapa E1 cada participante P_i fornece $Pedaços_i$ e *Etiquetas* como entrada e, além disso, investe $(m-1) \cdot q$ moedas. Esse investimento é a forma de garantir que P_i vai pagar a multa por aborto caso aja desonestamente. Se $i \in C$, então todas essas ações são feitas pelo adversário S.

O montante investido pelos participantes honestos é totalmente devolvido na etapa E2. Como na definição de f_{rec}^{esc}, essa etapa parece ser "artificial", uma vez que o mesmo efeito seria conseguido ao não exigir qualquer investimento de um participante honesto. Isso, no entanto, dificultaria a criação de protocolos seguros para computar f_{rec}^{cmm}.

Na etapa E3 o adversário decide sobre a continuação da execução com base no valor de z. Para que a execução seja interrompida, basta que *apenas um* dos membros do conluio C aborte. No entanto, qualquer subconjunto desses membros pode ser instruído por S a abortar. Esse subconjunto é representado por C' na Definição 8.107. Quando o adversário decide abortar, cada participante *fora de* C' recebe q moedas de cada um dos $|C'|$ desonestos, totalizando $|C'| \cdot q$. Nesse caso, a quantidade de moedas devolvidas ao adversário é calculada de maneira direta: o total de moedas investidas ($|C| \cdot (m-1) \cdot q$) menos a quantidade de moedas usadas para pagar as multas ($|C'| \cdot (m - |C'|) \cdot q$).

No caso em que S decide não abortar, a saída z é entregue também aos participantes honestos e o investimento feito por S é totalmente devolvido.

Por fim, ressaltamos que f_{rec}^{cmm} é monetariamente justa. Lembramos que a Definição 5.95 de justiça monetária exige que, ao fim da execução, (i) nenhum participante pague penalidades e (ii) caso a justiça (tradicional) tenha sido quebrada, cada participante honesto seja recompensado. A etapa E2 faz com que f_{rec}^{cmm} atenda a (i). O requisito (ii) é atendido pelo tratamento do caso de aborto, na etapa E3.

8.3 A Funcionalidade f_{cmm}

A funcionalidade f_{cmm} permite que m participantes se comprometam a revelar entradas que satisfaçam condições codificadas em circuitos, de maneira semelhante à f_{cr}. Mas, como veremos, o comprometimento não é feito apenas entre dois participantes, mas entre todos os m. Além disso, as entradas são todas reveladas simultaneamente.

Inicialmente, f_{cmm} recolhe *depósitos* dos participantes como garantia que eles vão se comportar honestamente. Para fazer seu depósito, cada participante P_i deve fornecer (i) um circuito φ_i como forma de se comprometer com um valor secreto e (ii) uma rodada τ na qual esse segredo deve ser revelado. Ou seja, φ_i deve ser tal que apenas o valor p_i é capaz de satisfazê-lo: $\varphi_i(p_i) = 1$. Na rodada τ, p_i deve ser revelado. Nosso objetivo é utilizar φ_i para que P_i se comprometa com $Pedaços_i$ mas, a princípio, φ_i pode conter uma condição arbitrária.

Na rodada τ, estabelecida pelos participantes durante os depósitos, ocorre a etapa de revelação. Nela, os participantes revelam, simultaneamente, os segredos p_i que satisfazem os circuitos φ_i.

Na rodada $\tau+1$, os participantes que revelaram segredos válidos *resgatam* seus depósitos, recebendo de volta seus investimentos iniciais. Já os que não revelaram, são penalizados. Isso significa que cada um desses participantes paga uma penalidade de q moedas para os demais.

Definimos formalmente f_{cmm} a seguir.

Definição 8.108 (Funcionalidade f_{cmm} para comprometimento monetário multilateral)**.** *Seja $\{P_1, \cdots, P_m\}$ um conjunto de participantes e q o valor da multa por aborto. A funcionalidade f_{cmm} para comprometimento monetário multilateral é definida como a seguir:*

E1. (Depósitos-garantia:) Nesta etapa cada participante deposita moedas como garantia de que vai se comportar honestamente.

Mais precisamente, para todo $i \in \{1, \cdots, m\}$:

- *(i) f_{cmm} espera a mensagem $(depósito, i, \varphi_i, \tau, valor((m-1)\cdot q))$,*
- *(ii) f_{cmm} envia a mensagem $(depósito, i, \varphi_i, \tau, (m-1)\cdot q)$ para todos os participantes e a armazena.*

E2. (Reembolso prematuro:) Nesta etapa, caso algum participante não tenha feito seu depósito-garantia, a execução é interrompida e os depósitos feitos, reembolsados.

Mais precisamente, seja $\mathbb{H} \stackrel{def}{=} \{i \in \{1, \cdots, m\} |$ a mensagem $(depósito, i, \varphi_i, \tau, (m-1)\cdot q)$ está armazenada.$\}$. Ou seja, $\mathbb{H}$ é o conjunto dos índices dos participantes que fizeram os depósitos-garantia. Se $\mathbb{H} \neq \{1, \cdots, m\}$:

- *(i) Para todo $i \in \mathbb{H}$, f_{cmm} envia as mensagens:*
  - *$(reembolso, i, valor((m-1)\cdot q))$ para P_i e,*

- $(reembolso, i, (m-1)\cdot q)$ *para todos os participantes*

(ii) A execução é encerrada.

E3. (Revelação:) Nesta etapa os participantes revelam simultaneamente entradas que satisfaçam os circuitos φ_i da etapa anterior.

Para isso, na rodada τ, ao receber a mensagem $(revelação, i, p)$, f_{cmm} verifica se:

(i) a mensagem $(depósito, i, \varphi_i, \tau, (m-1)\cdot q)$ está armazenada,

(ii) $\varphi_i(p) = 1$.

Se essas duas condições forem atendidas, então, f_{cmm} age como a seguir:

(i) remove a mensagem $(depósito, i, \varphi_i, \tau, (m-1)\cdot q)$,

(ii) envia $(revelação\text{-}sucesso, i, p)$ para todos os participantes e a armazena.

Se pelo menos uma delas não for atendida, então:

(i) envia $(revelação\text{-}falha, i, p)$ para todos os participantes.

E4. (Resgate e penalização:) Nesta etapa os participantes que tomaram parte na etapa anterior são reembolsados e os que não tomaram, penalizados.

Seja $H' \stackrel{def}{=} \{i \in \{1,\cdots,m\} \mid$ a mensagem $(revelação\text{-}sucesso, i, p)$ está armazenada $\}$ o conjunto dos índices dos participantes que revelaram valores válidos na etapa anterior de revelação. Seja $D \stackrel{def}{=} \{1,\cdots,m\} \setminus H'$ o conjunto dos participantes que ou não tomaram parte na etapa anterior ou tentaram revelar um valor inválido.

Na rodada $\tau+1$:

(i) Para todo $h' \in H'$, f_{cmm} envia a mensagem $(resgate, h', valor((m-1)\cdot q))$ para $P_{h'}$, fazendo a devolução do investimento desse participante. f_{cmm} envia a mensagem $(resgate, h', (m-1)\cdot q)$ para todos os participantes.

(ii) Para todo $d \in D$ e $i \in \{1,\cdots,m\} \setminus \{d\}$, f_{cmm} envia a mensagem $(penalidade, d, i, valor(q))$ para P_i e $(penalidade, d, i, q)$ para todos.

A funcionalidade f_{cmm} da Definição 8.108 se comporta da maneira que discutimos anteriormente. Ressaltamos, no entanto, um efeito das etapas E3 e E4. Quando um participante se comporta desonestamente, não tomando parte na etapa de revelação, *todos* os demais participantes são compensados. Isso pode incluir outros que *também agiram desonestamente*. Apesar disso, um participante desonesto sempre tem saldo final ≤ 0: se D é o conjunto de participantes desonestos, então o saldo final de cada um deles será o quanto foi recebido em E4 menos o que foi investido em E1, ou seja, $(|D|-1)\cdot q - (m-1)\cdot q = (|D|-m)\cdot q$. Esse valor é negativo se houver *pelo menos um* participante honesto.

Por fim, ressaltamos que f_{cmm} realiza o *broadcast* das mensagens que alteram seu estado. Isso tem como objetivo facilitar a implementação dessa funcionalidade a partir de transações *Bitcoin*.

8.4 O Protocolo Π_{rec}^{cmm} no Modelo f_{cmm}-híbrido

Prosseguimos nossa apresentação exibindo o protocolo Π_{rec}^{cmm} para computar a funcionalidade f_{rec}^{cmm} no modelo f_{cmm}-híbrido (ou seja, com acesso oracular a f_{cmm}). Esse protocolo é uma alternativa à construção escada para condução de reconstrução monetariamente justa de um segredo. Ele é bem simples e se baseia em apenas uma chamada a f_{cmm}.

De modo geral, precisamos apenas definir a relação entre os circuitos φ_i dos depósitos feitos através de f_{cmm} e as listas *Etiquetas* e *Pedaços*. Essa relação, no entanto, é a mesma que vimos na construção escada. Isto é, os circuitos φ_i são definidos de modo que um participante só consiga resgatar seu depósito ao revelar o elemento de *Pedaços* que possui.

Definimos Π_{rec}^{cmm} a seguir.

Definição 8.109 (Protocolo Π_{rec}^{cmm} para computação da funcionalidade f_{rec}^{cmm} no modelo f_{cmm}-híbrido)**.** *Seja $\{P_1,\cdots,P_m\}$ um conjunto de participantes, $(Divide, Reconstrói, Verifica)$ um $EVCS_{m,m}$ conforme implementado na Definição 6.97 e $(Etiquetas, Pedaços)$ o resultado da aplicação do algoritmo Divide a um segredo z a ser reconstruído. Para todo $i \in \{1,\cdots,m\}$, assumimos que P_i possui a lista Etiquetas e o valor $Pedaços_i$. Também assumimos que q é um valor predefinido, utilizado como multa por aborto. Representamos por $\varphi(et,\cdot)$ o circuito que é satisfeito por uma entrada p se e somente se $Verifica(et,p) = 1$. Isto é, $\varphi(et,\cdot)$ é satisfeito pelo pedaço p correspondente à etiqueta et.*

Definimos as etapas Π_{rec}^{cmm} a seguir.

E1. (Depósitos-garantia:) Nesta etapa os depósitos-garantia são feitos conforme a definição de f_{cmm}.

Mais precisamente, para todo $i \in \{1,\cdots,m\}$, o participante P_i envia para f_{cmm} a mensagem $(depósito, i, \varphi(Etiquetas_i,\cdot), \tau, valor((m-1)\cdot q))$.

E2. (Reembolso de aborto prematuro:) Se algum participante deixou de fazer um depósito na etapa anterior, os que fizeram recebem os investimentos de volta nesta etapa.

Ou seja, se para algum $j \in \{1,\cdots,m\}$ a mensagem $(depósito, j, \varphi(Etiquetas_j,\cdot), \tau, (m-1)\cdot q)$ não foi recebida de f_{cmm}, cada participante P_i que fez seu depósito-garantia na etapa anterior espera a mensagem $(reembolso, i, valor((m-1)\cdot q))$ e então interrompe a execução.

E3. (Revelação dos pedaços:) Nesta etapa os participantes revelam seus elementos da lista Pedaços.

Na rodada τ, para todo $i \in \{1,\cdots,m\}$, o participante P_i envia para f_{cmm} a mensagem $(revelação, i, Pedaços_i)$ e espera receber a mensagem $(revelação\text{-}sucesso, i, Pedaços_i)$ como resposta.

Se todos os participantes revelarem seus pedaços, então todos serão capazes de reconstruir o segredo z.

E4. (Resgate dos depósitos-garantia e penalização:) Nesta etapa os participantes que revelaram seus pedaços na etapa anterior resgatam seus depósitos. Já os que pularam a etapa anterior são penalizados.

Seja $H' \stackrel{def}{=} \{j \in \{1,\cdots,m\} \mid P_j$ recebeu a mensagem $(revelação\text{-}sucesso, j, Pedaços_j)$ na etapa anterior$\}$ o conjunto dos índices dos participantes que revelaram seus pedaços na etapa anterior. Então, para cada $h' \in H'$, o participante $P_{h'}$ recebe a mensagem $(resgate, h', valor((m-1)\cdot q))$ de f_{cmm}.

Seja $D \stackrel{def}{=} \{1,\cdots,m\} \setminus H'$ o conjunto dos índices dos participantes que pularam a etapa anterior. Então, para todo $d \in D$ e todo $d \in \{1,\cdots,m\} \setminus \{j\}$, o participante P_i recebe a mensagem $(penalidade, d, i, valor(q))$ de f_{cmm}.

Como mostra a Definição 8.109, o protocolo Π_{rec}^{cmm} se baseia fortemente na comunicação com uma instância de f_{cmm}.

Na etapa E1 os participantes fazem depósitos de $(m-1)\cdot q$ moedas como garantia de que vão revelar seus valores da lista *Pedaços* na rodada τ. Caso algum desses depósitos não seja feito, a execução é interrompida na etapa E2, após os participantes serem reembolsados. Esse cenário corresponde a um aborto prematuro e não põe em risco a justiça do protocolo, já que os participantes honestos são reembolsados e ninguém é capaz de reconstruir a saída z.

Na etapa E3 ocorre a revelação dos elementos da lista *Pedaços*. Nesta etapa os participantes precisam satisfazer os circuitos $\varphi(Etiquetas_i, \cdot)$, o que, por definição, só acontece com a apresentação dos $Pedaços_i$. Os participantes só podem resgatar seus investimentos se completarem a etapa E3 com sucesso.

Por fim, na etapa E4 os participantes que revelaram seus pedaços na etapa E3 têm seus depósitos resgatados. Ou seja, eles recebem as $(m-1)\cdot q$ moedas de volta. Por outro lado, cada participante P_i que não completou a etapa E3 com sucesso é penalizado. Isso significa que suas $(m-1)\cdot q$ moedas são divididas entre os demais, fazendo com que cada participante do conjunto $\{P_1,\cdots,P_m\} \setminus \{P_i\}$ receba q moedas como compensação.

Existem duas ocasiões em que um participante pode abortar: as etapas E1 e E3. Elas são as etapas em que o protocolo especifica ações para os participantes, enquanto que nas demais etapas as ações são tomadas pelo oráculo f_{cmm}. Na etapa E1, como já mencionamos, um aborto é considerado prematuro. Nesse caso, todo participante termina a execução com saldo igual a 0. Isso não fere a definição de justiça monetária porque não é possível para qualquer conluio

desonesto reconstruir a saída nesse momento[2].

Um aborto na etapa E3, no entanto, pode comprometer a justiça (tradicional) de Π_{rec}^{cmm}. Isso porque admitimos que, embora os elementos da lista *Pedaços* sejam todos revelados na mesma rodada τ, o adversário tem a habilidade de observar as mensagens dos participantes honestos antes de criar as suas próprias. Por essa razão, o adversário pode abortar a etapa E3 *após* receber os *broadcasts* de f_{cmm} relacionados às revelações dos pedaços dos participantes honestos. Dessa forma, o adversário torna-se capaz de reconstruir a saída z antes dos participantes honestos, como de fato também acontece na Definição 8.107 da funcionalidade f_{rec}^{cmm}.

Essa habilidade do adversário é especialmente importante quando consideramos o *Bitcoin* como plataforma para implementar Π_{rec}^{cmm}. Como veremos adiante, as rodadas definidas em Π_{rec}^{cmm} serão baseadas na publicação de transações na *blockchain*. Essas transações, no entanto, tornam-se públicas na rede *Bitcoin* antes de serem adicionadas na *blockchain*. É nesse momento que participantes desonestos têm a possibilidade de observar as transações dos honestos, mesmo antes criar e publicar as suas próprias.

Se na etapa E3 a justiça tradicional pode ser quebrada, a etapa E4 garante que a justiça *monetária* seja preservada. Isso é feito através da penalização dos participantes que abortaram a etapa E3 e da compensação dos demais.

Registramos a justiça de Π_{rec}^{cmm} no lema a seguir.

Lema 8.110 (Justiça de Π_{rec}^{cmm})**.** *O protocolo Π_{rec}^{cmm} é monetariamente justo.*

Demonstração. Lembramos que para a justiça monetária ser alcançada, duas propriedades precisam ser verdadeiras ao fim da execução:

(i) Nenhum participante honesto pagou qualquer penalidade (saldo ≥ 0),

(ii) Se a justiça tradicional foi quebrada, todo participante honesto é compensado (saldo > 0).

A propriedade (i) segue do fato que um participante honesto sempre recupera suas moedas investidas, ou pelo reembolso na etapa E2 ou pelo resgate de seu depósito na etapa E4.

A propriedade (ii) segue diretamente da etapa E4. Se o adversário abortar a etapa E3, única maneira pela qual a justiça tradicional pode ser quebrada, então suas moedas são utilizadas para compensar os demais participantes e, em particular, os honestos. □

Por fim, registramos também que o protocolo Π_{rec}^{cmm} computa de maneira segura a funcionalidade f_{rec}^{cmm}.

Lema 8.111 (Segurança de Π_{rec}^{cmm})**.** *O protocolo Π_{rec}^{cmm} (Definição 8.109) computa f_{rec}^{cmm} de maneira segura no modelo f_{cmm}-híbrido. (Π_{rec}^{cmm} uc-realiza f_{rec}^{cmm}.)*

[2]Sob a hipótese de que pelo menos um dos participantes é honesto.

Demonstração. A ideia geral da prova é criar um adversário S ideal para f_{rec}^{cmm} que *imita* o comportamento de um adversário real A para Π_{rec}^{cmm}. Isso se baseia na noção de segurança da Definição 6.96. Para tanto, o adversário S precisa *simular* o ambiente de execução de Π_{rec}^{cmm} para A. Isso significa, por exemplo, que S deve agir como o oráculo f_{cmm} para lidar com a criação e resgate de depósitos.

O *simulador* S precisa também "se passar" pelos participantes honestos, simulando suas ações. Isso é um desafio porque S não conhece as entradas desses participantes, as quais precisam ser reveladas para A durante a etapa de revelação de pedaços em Π_{rec}^{cmm}. Por fim, S precisa também contactar a funcionalidade f_{rec}^{cmm} e decidir se aborta ou não de acordo com o comportamento de A.

Com a construção desse simulador queremos estabelecer que qualquer adversário (real) para Π_{rec}^{cmm} tem um correspondente (ideal) que atinge os mesmos efeitos ao atacar f_{rec}^{cmm}. Por isso, o protocolo Π_{rec}^{cmm} é tão seguro quanto f_{rec}^{cmm}.

Breve Recapitulação da Nomenclatura Utilizada. Lembramos que as entradas dos m participantes de f_{rec}^{cmm} são os pares $(Etiquetas, Pedaços_i)$, gerados a partir da aplicação do algoritmo *Divide* de um $EVCS_{m,m}$, implementado de acordo com a Definição 6.97, sobre um segredo z. O valor $Pedaços_i$ é um par (p_i, w_i) tal que $Etiquetas_i = \overline{Comp}_{w_i}(p_i)$. Ou seja, $Etiquetas_i$ é o resultado de se comprometer com o pedaço p_i utilizando w_i como fonte de aleatoriedade. O esquema de comprometimento *Comp* para um *bit* e sua extensão para cadeias, $\overline{Comp}$, são, por hipótese, equívocos e, portanto, possuem os algoritmos adicionais *Equivoca* e *Desequivoca* conforme a Definição 3.17. Lembramos ainda que $\bigoplus_{i=1}^{m} p_i = z$.

Ressaltamos que o esquema de comprometimento utilizado, $\overline{Comp}$, lida com *cadeias* de *bits* e, da mesma forma, os algoritmos *Equivoca* e *Desequivoca* a ele associados.

Denotamos por C o conjunto dos índices dos participantes desonestos da execução de f_{rec}^{esc} e por H o conjunto $\{1, \cdots, m\} \setminus C$ dos índices dos participantes honestos.

Visão Geral da Simulação. De maneira geral, a simulação conduzida pelo adversário S, que chamaremos apenas de *simulador*, pode ser dividida em 3 etapas fundamentais:

1. Uma para simular a realização dos depósitos-garantia,

2. Outra para simular a revelação dos pedaços dos participantes,

3. E uma última para simular o resgate dos depósitos-garantia e a penalização dos participantes que abortaram.

Nessas etapas, o simulador precisa enviar e receber mensagens fingindo ser o oráculo f_{cmm}. Uma das dificuldades em se fazer isso é ter que lidar com moedas. De fato, lembramos que o modelo de computação que adotamos no Capítulo 6 não permite que S crie ou forje moedas. Por isso, ele deve conduzir toda a simulação e, além disso, contactar a funcionalidade f_{rec}^{cmm}, utilizando apenas as moedas fornecidas pelos participantes desonestos controlados por A.

Um ponto fundamental da simulação é que S precisa simular o comportamento dos participantes honestos mesmo sem conhecer as entradas deles. Isso é relevante porque o objetivo de S é imitar A, o qual pode definir seu comportamento com base no segredo reconstruído z. Mas A só pode obter z através da simulação de Π_{rec}^{cmm} se, de alguma forma, S conseguir forjar valores $Pedaços_h$, para todo $h \in H$, tais que a soma (ou-exclusivo) de todos os pedaços seja igual a z. O problema é, portanto, que S precisa definir *Etiquetas* e *Pedaços* para a simulação muito antes de poder contactar f_{rec}^{cmm}, maneira pela qual ele obtém z.

Para contornar esse problema, S se baseia no fato de que o esquema de comprometimento utilizado pelo $EVCS_{m,m}$ é, por hipótese, *equívoco*. Essa característica, aliada ao fato de que A só obtém z quando o *último* participante honesto revela seu pedaço, permite que S aja da seguinte maneira. Seja $MaxH \stackrel{def}{=} \max\{H\}$ o maior índice de um participante honesto. Então, para todo $h \in H \setminus \{MaxH\}$, S define $Pedaços'_h$ como um valor aleatoriamente gerado e $Etiquetas'_h$ como o resultado de se comprometer com tal valor. Para $MaxH$, por sua vez, S invoca o algoritmo *Equivoca* para obter $(coringa_{MaxH}, info_{MaxH})$ e definir $Etiquetas'_{MaxH} = coringa_{MaxH}$.

Em linhas gerais, o simulador cria uma etiqueta coringa, ou *ambígua*, para P_{MaxH}. Quando chegar a hora desse participante revelar $Pedaços'_{MaxH}$, S deve estar apto a:

1. contactar f_{rec}^{cmm},
2. obter a saída z,
3. calcular $p_{MaxH} = z \bigoplus_{i \neq MaxH} p'_i$, em que $Pedaços'_i = (p'_i, w'_i)$,
4. e utilizar $(info_{MaxH}, p_{MaxH})$ para desequivocar $Etiquetas'_{MaxH}$, gerando um valor $Pedaços'_{MaxH} = Desequivoca(info_{MaxH}, p_{MaxH})$ adequado.

Ressaltamos que em f_{rec}^{cmm} os participantes honestos revelam pedaços de maneira simultânea, de modo que faz pouco sentido se falar em *um último participante* a revelar seu pedaço. Escolhemos gerar uma etiqueta ambígua para P_{MaxH} de maneira arbitrária. Poderíamos ter escolhido qualquer outro participante honesto.

Utilizamos $Etiquetas'$ e $Pedaços'$ (ao invés de *Etiquetas* e *Pedaços*) para deixar claro que as listas utilizadas na simulação não são as originais, possuídas pelos participantes no início da execução de f_{rec}^{cmm}. A diferença entre $Etiquetas'$ (resp. $Pedaços'$) e *Etiquetas* (resp. $Etiquetas'$) são os valores correspondentes aos participantes *honestos*.

Nas seções seguintes apresentaremos como S deve conduzir a simulação de Π_{rec}^{cmm}.

Inicialização do Simulador. Para começar a simulação, S inicializa algumas variáveis importantes:

1. *(Índice do último participante honesto:)* $MaxH = \max\{H\}$.
2. *(Listas $Etiquetas'$ e $Pedaços'$:)* Os pedaços e etiquetas dos participantes desonestos, durante a simulação, são os originais[3]. Os dos participantes honestos, com exceção de

[3]Como já ressaltamos, as listas originais, *Etiquetas* e *Pedaços*, são as que compõem as entradas dos participantes na execução de f_{rec}^{esc}. As listas $Etiquetas'$ e $Pedaços'$ são as utilizadas durante a simulação.

P_{MaxH}, são gerados aleatoriamente. Ou seja:

Para todo $c \in C$, $Etiquetas'_c = Etiquetas_c$ e $Pedaços'_c = Pedaços_c$. Para todo $h \in H \setminus \{MaxH\}$, $Pedaços'_h = (p'_h, w'_h)$ e $Etiquetas'_h = \overline{Comp}_{w'_h}(p'_h)$, em que p'_h e w'_h são cadeias escolhidas aleatoriamente.

3. *($Etiquetas'_{MaxH}$:)* $Etiquetas'_{MaxH}$ é um valor coringa, gerado pelo algoritmo *Equivoca*.

   $Etiquetas'_{MaxH} = coringa_{MaxH}$, em que $(coringa_{MaxH}, info_{MaxH}) = Equivoca(r)$, com r aleatoriamente escolhido. O valor $info_{MaxH}$ deve ser armazenado e vai servir para desequivocar $coringa_{MaxH}$. Note que o valor $Pedaços'_{MaxH}$ fica indefinido por enquanto.

4. Para todo $c \in C$, $FezDepGarantia_c = 0$. A variável $FezDepGarantia_c$ indica se P_c fez seu depósito-garantia.

5. Para todo $c \in C$, $Revelação_c = 0$. A variável $Revelação_c$ indica se P_c revelou o valor $Pedaços'_c$ na etapa de revelação.

Simulação dos Depósitos-Garantia. Para simular os depósitos-garantia, S precisa agir como o oráculo f_{cmm} para coletar os depósitos dos participantes desonestos e para fazer os *broadcasts* de mensagens necessários. Note que os participantes honestos são simulados por S e, portanto, não fazem depósitos de verdade. Mas S precisa simular o *broadcast* desses depósitos para A.

A simulação é feita como a seguir:

1. Para todo $h \in H$ o simulador S envia a mensagem $(depósito, h, \varphi(Etiquetas'_h, \cdot), \tau, (m-1) \cdot q)$ para A. Isso simula o *broadcast* do depósito-garantia de um participante honesto.

2. Para todo $c \in C$, o simulador espera receber de A, controlando P_c, a mensagem $(depósito, c, \varphi(Etiquetas'_c, \cdot), \tau, valor((m-1) \cdot q))$. S atualiza $FezDepGarantia_c = 1$ e envia a mensagem $(depósito, c, \varphi(Etiquetas'_c, \cdot), \tau, (m-1) \cdot q)$ para A, simulando um *broadcast*.

Se algum depósito-garantia não foi feito, i.e., existe algum $c \in C$ tal que $FezDepGarantia = 0$, então S precisa simular o reembolso dos depósitos feitos. Depois disso a execução é encerrada sem que f_{rec}^{cmm} tenha sido contactada.

Se existe $c \in C$ tal que $FezDepGarantia = 0$, então S procede como a seguir:

1. Para todo $c \in C$ tal que $FezDepGarantia = 1$, S se passa pelo oráculo f_{cmm} e envia as mensagens:

   - $(reembolso, c, valor((m-1) \cdot q))$ para A controlando P_c,
   - $(reembolso, c, (m-1) \cdot q)$ para A simulando o *broadcast* da mensagem acima.

2. Encerra a execução e copia a saída de A. Nesse caso, não houve contato de S com f_{rec}^{cmm}.

Simulação da Revelação dos Pedaços. Se a simulação chegar até este ponto, então todos os depósitos-garantia foram feitos. Isto é, para todo $c \in C$, $FezDepGarantia_c = 1$. Por isso o próximo passo de S é simular a revelação dos elementos de $Pedaços'$. Para tanto, uma das dificuldades que devem ser enfrentadas é a de que $Pedaços'_{MaxH}$ permanece indefinido. Para definir esse valor, S precisa contactar a funcionalidade f_{rec}^{cmm} e obter o segredo reconstruído z. Em seguida, S calcula o valor de $Pedaços'_{MaxH}$ de modo que a soma de todos os pedaços resulte em z. Além disso, S precisa utilizar o algoritmo *Desequivoca* para garantir que $Etiquetas'_{MaxH}$ seja um comprometimento válido para $Pedaços'_{MaxH}$. Definimos os detalhes a seguir.

Na rodada τ:

1. S contacta f_{rec}^{cmm} através da mensagem $(entrada, h, Etiquetas, \{Pedaços_c \mid c \in C\}, valor(|C| \cdot (m-1) \cdot q))$, utilizando as moedas recebidas durante a simulação dos depósitos-garantia. O simulador então procede como a seguir:
   - Recebe de f_{rec}^{cmm} a mensagem $(saída, z)$,
   - Calcula $p'_{MaxH} = z \oplus \left(\bigoplus_{i \neq MaxH} p'_i \right)$, em que $(p'_i, w'_i) = Pedaços'_i$. Note que, com isso, $\bigoplus_{i=1}^{m} p'_i = z$,
   - Calcula $w'_{MaxH} = Desequivoca(info_{MaxH}, p'_{MaxH})$,
   - Define $Pedaços'_{MaxH} = (p'_{MaxH}, w'_{MaxH})$.
2. Para todo $h \in H$, S age como o oráculo f_{cmm} e envia para A a mensagem $(revelação\text{-}sucesso, h, Pedaços'_h)$, simulando a revelação do pedaço do participante honesto P_h. Com isso, o adversário A já é capaz de reconstruir o valor z.
3. Para todo $c \in C$, S espera a mensagem $(revelação, c, p)$ de A controlando P_c. Se ela for recebida:
   - Se S já enviou a mensagem $(revelação\text{-}sucesso, c, Pedaços'_c)$ antes, então o simulador envia a mensagem $(revelação\text{-}falha, c, p)$ para A controlando P_c.
   - Se $Verifica(Etiquetas'_c, p) = 1$, mas $p \neq Pedaços'_c$, então A quebrou a não-ambiguidade do esquema de comprometimento utilizado. S responde $(aborte, \perp)$ para f_{rec}^{cmm} e encerra a execução com saída $Falha_{ambg}$.
   - Se $Verifica(Etiquetas'_c, p) = 1$ e $p = Pedaços'_c$, então S envia a mensagem $(revelação\text{-}sucesso, c, Pedaços'_c)$ para A e atualiza $Revelação_c = 1$.

Na rodada $\tau + 1$:

1. Seja $C' \stackrel{def}{=} \{c \in C \mid Revelação_c = 0\}$ o conjunto dos índices dos participantes desonestos que não revelaram seus pedaços na etapa anterior.
2. Se $C' \neq \emptyset$, então A decidiu abortar:

- Para imitar essa decisão, S envia $(aborte, C')$ para f_{rec}^{cmm}.
- Para todo $c \in C \setminus C'$, S recebe de f_{rec}^{cmm} a mensagem $(resgate, c, valor((m-1) \cdot q))$ de resgate do investimento do participante desonesto P_c que não abortou. Agindo como o oráculo f_{cmm}, S repassa essa mensagem para A controlando P_c.
- Para todo $h \in H$, S envia para A a mensagem $(resgate, h, (m-1) \cdot q)$. Isso simula o *broadcast* dos resgates dos participantes honestos.
- Para todo $c' \in C'$ e todo $c \in C \setminus C'$, S recebe a mensagem $(penalidade, c', c, valor(q))$ que penaliza o participante $P_{c'}$ e compensa P_c. Agindo como o oráculo f_{cmm}, S repassa essa mensagem para A controlando P_c.

  Note que, após as mensagens de penalidade e resgate serem recebidas, a quantidade de moedas de cada participante desonesto P_c vai depender de se ele abortou ou não:

  - Se P_c abortou (i.e., $c \in C'$) ele terá $(|C'|-1) \cdot q$ moedas, originadas da penalização dos demais participantes em C',
  - Senão (i.e., $c \in C \setminus C'$), P_c terá $((m-1)+|C'|) \cdot q$ moedas, originadas do resgate de seu investimento e da penalização dos participantes em C'.

3. Se $C' = \emptyset$, então A decidiu não abortar:

   - Para imitar essa decisão, S envia $(continue)$ para f_{rec}^{cmm}.
   - Para todo $c \in C$, S recebe a mensagem $(resgate, c, valor((m-1) \cdot q))$ de resgate do investimento do participante desonesto P_c. Agindo como o oráculo f_{cmm}, S repassa essa mensagem para A controlando P_c.
   - Para todo $h \in H$, S envia para A a mensagem $(resgate, h, (m-1) \cdot q)$. Isso simula o *broadcast* dos resgates dos participantes honestos.

4. O simulador S copia a saída de A.

Para encerrar a demonstração que Π_{rec}^{cmm} computa seguramente a funcionalidade f_{rec}^{cmm}, precisamos argumentar que as saídas de S e A são indistinguíveis. A saída de S é exatamente igual à de A *exceto* em uma ocasião: quando A, durante a etapa de revelação dos pedaços, quebra a não-ambiguidade do esquema de comprometimento utilizado. Nesse caso, S encerra a execução com saída $Falha_{ambg}$. No entanto, essa situação só pode acontecer com probabilidade desprezível, segundo o item relativo à não-ambiguidade da Definição 3.14 (esquemas de comprometimento).

De maneira geral, a segurança de Π_{rec}^{cmm} se baseia fortemente na segurança do esquema de comprometimento utilizado. Note que S revela a A pares $(Etiquetas'_h, Pedaços'_h)$ gerados sobre valores *aleatórios*. Mais que isso, uma das etiquetas (a de P_{MaxH}) é *ambígua*. Se A fosse capaz de perceber essa manipulação de S então não seria possível provar que Π_{rec}^{cmm} é um protocolo seguro da maneira com que fizemos.

Outro aspecto importante da prova apresentada é que S é capaz de conduzir toda a simulação apenas com as moedas obtidas de A. Essa é uma característica essencial para a segurança de Π_{rec}^{cmm} porque o modelo de computação adotado só permite que o ambiente $\mathcal{E}$ crie moedas, e não o adversário S.

□

Ressaltamos que, como f_{rec}^{cmm} é monetariamente justa, o Lema 8.111 implica o Lema 8.110. Explicitamos o Lema 8.110 apenas porque ele estabelece um resultado importante e possui uma prova mais sucinta e intuitiva.

8.5 Implementação de f_{cmm} com *Bitcoin*

8.5.1 Visão Geral

O que falta para concretizarmos o protocolo Π_{rec}^{cmm} é apresentar uma implementação da funcionalidade f_{cmm}. Utilizaremos, para isso, o protocolo *Bitcoin* como suporte.

A nossa implementação da funcionalidade f_{cmm} se baseia em uma série de transações, as quais são resumidas na Tabela 8.1. A Txn^{comp} é a principal delas, permitindo que os participantes (i) especifiquem *scripts* φ_i, (ii) se comprometam a revelar entradas que satisfaçam o *script* φ_i adequado e (iii) sejam penalizados caso não revelem. Essa lógica é especificada por suas $m \cdot (m-1)$ saídas. Cada participante P_i é *responsável* por um conjunto S_i contendo $(m-1)$ dessas saídas. Cada uma delas disponibiliza q moedas que podem ser resgatadas pelo próprio P_i, através da transação Txn_i^{resg}, ou por P_j, com $j \neq i$, através da transação $Txn_{i\to j}^{pen}$.

A transação Txn_i^{resg} resgata as moedas de todas as saídas de Txn^{comp} contidas em S_i e, para isso, revela uma entrada para o *script* φ_i. Se Txn_i^{resg} não aparecer na *blockchain* antes de uma rodada predefinida, então cada P_j, com $j \neq i$, pode utilizar $Txn_{i\to j}^{pen}$ para reivindicar uma das saídas em S_i. Dessa forma, P_j transfere para si q moedas que originalmente pertenciam a P_i.

Outra transação envolvida na implementação de f_{cmm}, a Txn_i^{dep}, é criada por P_i para realizar o depósito-garantia de $(m-1) \cdot q$ moedas. Ela tem o papel fundamental de restringir *quando* a transação Txn^{comp} pode ser publicada. De fato, Txn^{comp} possui uma entrada para cada transação Txn_i^{dep} e, portanto, só pode aparecer na *blockchain* quando *todas* as transações Txn_i^{dep} também tiverem aparecido. Dessa forma, P_i pode bloquear a publicação de Txn^{comp} escondendo sua Txn_i^{resg}. Isso deve ser feito até que todas as transações $Txn_{j\to i}^{pen}$, que garantem uma compensação a P_i caso P_j aja desonestamente, sejam criadas. Essa estratégia de esconder Txn_i^{dep} é o que nos permite implementar f_{cmm} sem admitir alterações no protocolo *Bitcoin*.

Utilizaremos, na sequência, as mesmas noções de *tempo* e *rodada* adotadas pela construção escada. Isto é, vamos encarar a *blockchain* como um relógio global em que o tempo é medido em blocos. Definimos T_0 como o tempo inicial do protocolo, isto é, a quantidade de blocos na *blockchain* imediatamente antes do protocolo ser iniciado. Definimos ainda ΔT como sendo a duração de uma rodada em blocos e T_{seg} como o parâmetro de segurança discutido durante a

Tabela 8.1: As transações *Bitcoin* envolvidas na implementação da funcionalidade f_{cmm}.

Transação	Descrição
Txn_i^{dep}	Transação que corresponde ao **dep**ósito-garantia do participante P_i. Sua única saída só pode ser gasta com uma assinatura do próprio P_i. Deve permanecer secreta até que P_i receba as transações $Txn_{j\to i}^{pen}$.
Txn^{comp}	Esta transação permite que os participantes especifiquem os *scripts* φ_i e se **comp**rometam a revelar entradas para eles. Ela gasta as moedas das transações Txn_i^{dep} e deve ser criada através da cooperação entre os participantes. Suas saídas podem ser usadas tanto para resgate dos depósitos iniciais quanto para o recebimento de compensações.
Txn_i^{resg}	Esta transação permite a P_i **resg**atar seu depósito-garantia revelando uma entrada p tal que $\varphi_i(p) = 1$.
$Txn_{i\to j}^{pen}$	Esta transação **pen**aliza P_i em q moedas, as quais são usadas para compensar P_j. Ela só pode ser publicada na rodada $\tau + 1$ e se Txn_i^{resg} não foi publicada.

apresentação da construção escada[4]. Dessa forma, a rodada τ começa no instante $T_0 + \tau \cdot \Delta T$.

Em seguida vamos definir detalhadamente as transações necessárias para a criação de um protocolo para computar de f_{cmm} baseado no *Bitcoin*. Utilizaremos a nomenclatura estabelecida pela Definição 8.108. Ou seja, vamos admitir que m participantes vão se engajar na execução do protocolo. Cada P_i deve especificar um *script* φ_i e fazer um investimento inicial. O valor q de multa para aborto é conhecido por todos antes da execução, assim como a rodada τ em que os depósitos-garantia devem ser resgatados.

8.5.2 A Transação Txn_i^{dep}

A transação Txn_i^{dep} transfere $(m-1) \cdot q$ moedas do participante P_i para ele mesmo. Ela é composta de uma única saída que disponibiliza essa quantia e só pode ser reivindicada com a apresentação de uma assinatura feita com uma chave secreta de P_i. A transação Txn_i^{dep} é esquematizada na Figura 8.1. Não fazemos restrições sobre a origem das moedas e, por isso, não especificamos a entrada (ou entradas) de Txn_i^{dep}.

O *script* de saída de Txn_i^{dep} é de um tipo bastante comum nas transações *Bitcoin*. Ele recebe a versão simplificada da transação que reivindica Txn_i^{dep} ([txn]) e uma assinatura dessa versão gerada com a chave secreta de P_i (A_{Sk_i}). O *script* então verifica, utilizando a chave pública de P_i (Pk_i), se essa assinatura é válida. Essa chave deve ser conhecida no momento da criação de Txn_i^{dep} para ser introduzida no código como uma constante.

[4] T_{seg} é a quantidade mínima de confirmações exigidas para que um bloco seja considerado pertencente à cadeia principal da *blockchain*

Figura 8.1: A Transação Txn_i^{dep}

Txn_i^{dep}
Saída
Valor: $(m-1)\cdot q$ **btc**
Script de Saída: **Parâmetros:** $[Txn]$, A_{Sk_i} **Início:** **retorne** $VrfAssinatura(Pk_i, [Txn], A_{Sk_i}) = 1$

Fonte: Elaborada pelo autor.

8.5.3 A Transação Txn^{comp}

A transação Txn^{comp} possui m entradas, cada uma reivindicando uma das transações Txn_i^{dep}. Ela possui $m\cdot(m-1)$ saídas, denotadas por $Saída_{i\to j}$, para todo $i, j \in \{1,\cdots,m\}$ e $i \neq j$. Cada participante P_i é responsável pelo conjunto de saídas $S_i \stackrel{def}{=} \{Saída_{i\to j} | j \in \{1,\cdots,m\} \setminus \{i\}\}$, as quais são construídas levando-se em conta o *script* φ_i especificado por P_i. Isso significa que P_i pode reivindicar as saídas de S_i exibindo (i) uma assinatura gerada com sua chave secreta, para provar sua identidade, e (ii) uma entrada para o *script* φ_i. Tendo em vista a funcionalidade f_{cmm}, P_i participa da *etapa E3 de revelação* quando reivindica as saídas de S_i até uma determinada rodada τ.

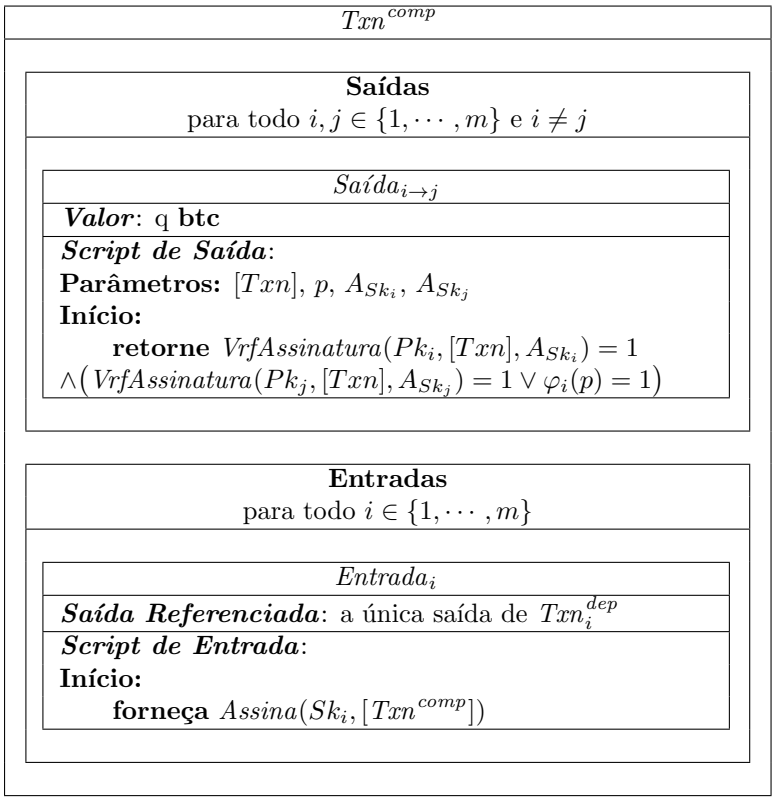
Se P_i não reivindica uma saída $Saída_{i\to j}$ a tempo, então, na rodada $\tau+1$, o participante P_j pode fazê-lo. Para isso ele precisa apresentar duas assinaturas: uma feita como sua chave secreta (Sk_j) e outra feita com a chave secreta de P_i (Sk_i). De forma intuitiva, isso significa que P_j precisa provar sua identidade e mostrar que P_i, ao fornecer uma assinatura sua, concordou que essa reivindicação fosse feita caso ele se comportasse desonestamente.

Na Figura 8.2 esquematizamos *modelos* de entradas e saídas de Txn^{comp}. Ressaltamos que Txn^{comp} contém m entradas e $m\cdot(m-1)$ saídas. Mas como elas são semelhantes e parametrizáveis por índices, podemos representá-las de uma forma resumida, ao invés de listá-las exaustivamente.

A transação Txn^{comp} deve ser construída incrementalmente, com a cooperação de todos os participantes. De fato, cada $Entrada_i$ referencia uma transação Txn_i^{dep} e fornece uma assinatura gerada com a chave Sk_i. Isso implica que $Entrada_i$ só pode ser criada por P_i. Já as saídas de Txn^{comp}, pelo menos no contexto da funcionalidade f_{cmm}, são compostas por informações públicas e, portanto, todas podem ser construídas por qualquer participante. Se isso não fosse verdade, no entanto, os participantes precisariam cooperar primeiro para preencher as saídas e só então criar as entradas[5].

[5]Isso porque as assinaturas presentes nas entradas são geradas sobre a versão simplificada da transação, a qual contém as saídas. Por isso, quando as entradas são criadas, as saídas já precisam estar prontas.

Figura 8.2: A Transação Txn^{comp}



Fonte: Elaborada pelo autor.

8.5.4 A Transação Txn_i^{resg}

Ao descrever a transação Txn^{comp}, definimos o conjunto $S_i \stackrel{def}{=} \{Saída_{i \to j} | j \in \{1, \cdots, m\} \setminus \{i\}\}$ das saídas pelas quais o participante P_i é responsável. O papel da transação Txn_i^{resg} é permitir a P_i reivindicar essas saídas e, consequentemente, tornar pública uma entrada que satisfaça o *script* φ_i.

A transação Txn_i^{resg} é esquematizada na Figura 8.3. Ela possui $(m-1)$ entradas que referenciam as saídas em S_i. Essas entradas são exatamente iguais entre si: elas fornecem uma assinatura gerada com a chave secreta de P_i e uma entrada p que satisfaz φ_i[6]. Não impomos restrições sobre como as $(m-1) \cdot q$ moedas transferidas por Txn_i^{resg} são usadas e, por isso, não especificamos a saída (ou saídas) dessa transação.

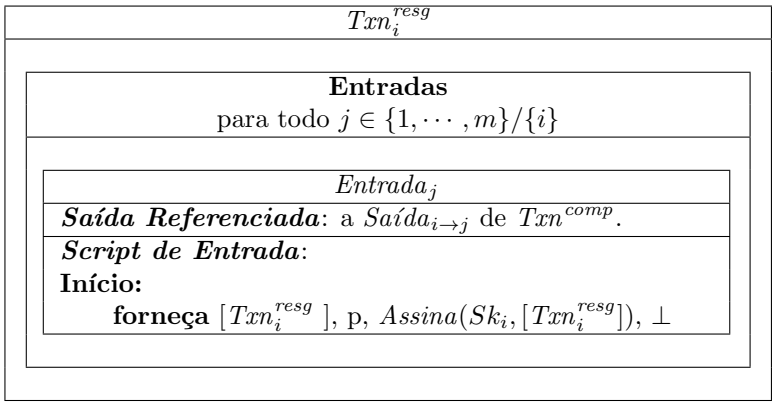
8.5.5 A Transação $Txn_{i \to j}^{pen}$

Mais uma vez faremos referência ao conjunto $S_i \stackrel{def}{=} \{Saída_{i \to j} | j \in \{1, \cdots, m\} \setminus \{i\}\}$ das saídas pelas quais o participante P_i é responsável.

A transação $Txn_{i \to j}^{pen}$ permite que, na rodada $\tau + 1$, o participante P_j reivindique $Saída_{i \to j} \in S_i$ se ela não tiver sido reivindicada por P_i através de Txn_i^{resg}. Para isso, a única entrada de $Txn_{i \to j}^{pen}$

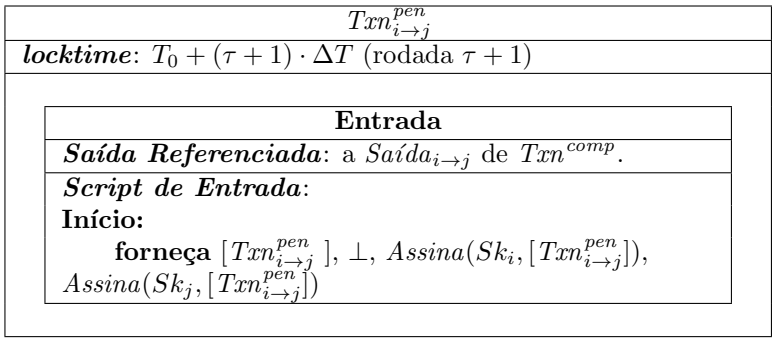
[6] O objetivo dessas entradas é o mesmo mas, na prática, elas podem diferir nas assinaturas que contém. No entanto isso é só um detalhe de implementação.

Figura 8.3: A Transação Txn_i^{resg}



Fonte: Elaborada pelo autor.

Figura 8.4: A Transação $Txn_{i\to j}^{pen}$



Fonte: Elaborada pelo autor.

explora uma das formas de se reivindicar $Saída_{i\to j}$: apresentando uma assinatura feita com a chave secreta de P_i (Sk_i) e outra com a de P_j (Sk_j). Por essa razão, $Txn_{i\to j}^{pen}$ só pode ser construída através da cooperação entre P_i e P_j. Além disso, como $Txn_{i\to j}^{pen}$ referencia $Saída_{i\to j}$, ela só pode ser criada depois da transação Txn^{comp}.

Outro aspecto importante de $Txn_{i\to j}^{pen}$, como mencionamos, é que ela só pode aparecer na *blockchain* após a rodada τ. Essa propriedade é implementada através de seu campo *locktime*, o qual deve ter seu valor definido para o instante $T_0 + (\tau+1)\cdot\Delta T$ (início da rodada $\tau+1$). A transação $Txn_{i\to j}^{pen}$ é esquematizada na Figura 8.4.

8.5.6 Juntando as Peças: o Protocolo Π_{cmm}

Nessa seção vamos apresentar o protocolo Π_{cmm} para computar f_{cmm} utilizando o *Bitcoin*. Nós precisamos descrever como os participantes devem proceder para criar as transações que descrevemos anteriormente. Além disso, precisamos deixar claro como as fases de f_{cmm} são conduzidas com a publicação dessas transações.

Definimos Π_{cmm} a seguir.

Definição 8.112 (Protocolo Π_{cmm} para computar f_{cmm} utilizando o *Bitcoin*)**.** *Seja $\{P_1, \cdots, P_m\}$ um conjunto de participantes. Assumimos que o valor q da multa por aborto e os scripts φ_i são conhecidos por todos. Definimos as etapas do protocolo Π_{cmm} a seguir.*

E0. (Criação das transações:)

- *Txn_i^{dep}: Na rodada preliminar τ_0', para todo $i \in \{1, \cdots, m\}$, o participante P_i cria a transação Txn_i^{dep} conforme a Figura 8.1. É importante que P_i conserve Txn_i^{dep} secreta.*
- *Txn^{comp}: Para a criação da transação Txn^{comp} os participantes devem cooperar da seguinte forma:*
  - *(i) Na rodada preliminar τ_1', P_1 cria Txn^{comp} e preenche todas as saídas da transação. Ele consegue fazer isso porque o código do script de saída de cada $Saída_{i\to j}$ opera sobre valores que são parametrizados ou que são publicamente conhecidos (Pk_i, Pk_j e φ_i). Os scripts das entradas de Txn^{comp}, no entanto, são deixados vazios.*
  - *(ii) Na rodada preliminar τ_i', para $i = 1, \cdots, (m-1)$, P_i verifica se a versão parcial de Txn^{comp} que ele possui está de acordo com a Figura 8.2, considerando apenas as $i-1$ entradas que essa versão possui. Ele então preenche $Entrada_i$ colocando sua assinatura nela. Por fim, P_i envia a transação Txn^{comp} atualizada com $Entrada_i$ para P_{i+1}.*
  - *(iii) Na rodada preliminar τ_m', P_m verifica se a versão parcial de Txn^{comp} está de acordo com a Figura 8.2 e, em caso positivo, preenche $Entrada_m$. Ele então envia Txn^{comp}, agora completa, para todos os demais.*
  - *(iv) Se em algum passo anterior um participante receber uma versão parcial de Txn^{comp} que não esteja nos conformes da Figura 8.2, ele deve prontamente interromper a execução.*
- *$Txn_{i\to j}^{pen}$: Na rodada preliminar τ_{m+1}' o participante P_j, para todo $j \in \{1, \cdots, m\}$, cria as transações $Txn_{i\to j}^{pen}$, para todo $i \neq j$, conforme a Figura 8.4. A única informação que ele não pode preencher na (entrada da) transação $Txn_{i\to j}^{pen}$ é a assinatura gerada com a chave secreta Sk_i de P_i. Por isso, ele envia a versão parcial de $Txn_{i\to j}^{pen}$ para P_i e espera receber de volta a versão final devidamente assinada. Ressaltamos que essa versão parcial enviada a P_i **não** pode conter a assinatura de P_j. Se para algum i a versão final de $Txn_{i\to j}^{pen}$ não for recebida na rodada τ_{m+2}', P_j deve abortar a execução.*

E1. (Depósito:) Nesta etapa os participantes fazem seus depósitos-garantia. Isso significa publicar as transações Txn_i^{dep}, para todo $i \in \{1, \cdots, m\}$, e a transação Txn^{comp}.

Para isso, na rodada τ_0, para todo $i \in \{1,\cdots,m\}$, P_i publica Txn_i^{dep}. Quando todas as transações Txn_i^{dep} aparecerem na blockchain, P_1 publica Txn^{comp}.

E2. (Reembolso prematuro:) Nesta etapa os participantes reivindicam as transações Txn_i^{dep} caso Txn^{comp} não tenha aparecido na blockchain.

Ou seja, na rodada τ_1, se Txn^{comp} não estiver na blockchain, cada participante P_i reivindica a transação Txn_i^{dep}. Para isso P_i cria uma transação com uma entrada semelhante à $Entrada_i$ de Txn^{comp}.

E3. (Revelação e resgate:) Nesta etapa os participantes revelam entradas que satisfazem os circuitos φ_i e resgatam seus investimentos iniciais. Isso significa publicar as transações Txn_i^{resg}, para todo $i \in \{1,\cdots,m\}$.

Na rodada $\tau_1 = \tau$,[7] com Txn^{comp} na blockchain, cada participante P_i cria Txn_i^{resg}, conforme descrito na Figura 8.3, e a publica.

E4. (Penalização:) Nesta etapa os participantes que pularam a etapa anterior são penalizados. Isso significa que as transações $Txn_{i\to j}^{pen}$ são publicadas.

Na rodada $\tau_2 = \tau + 1$, para cada Txn_i^{resg} que não foi publicada, todo participante P_j, com $j \in \{1,\cdots,m\} \setminus \{i\}$, publica $Txn_{i\to j}^{pen}$ para receber q moedas que deveriam ter sido resgatadas por P_i.

Uma das etapas mais críticas do protocolo Π_{cmm} é a etapa E0 de criação das transações. Nela, m^2+1 transações são criadas ao todo (m transações Txn_i^{dep}, 1 transação Txn^{comp} e $m\cdot(m-1)$ transações $Txn_{i\to j}^{pen}$). As transações Txn_i^{dep} são as mais simples, pois podem ser criadas por cada participante sem qualquer interação com os demais. Um ponto importante é que cada transação Txn_i^{dep} deve permanecer secreta até que P_i receba todas as transações $Txn_{i\to j}^{pen}$. Essa é a nossa estratégia para impedir que Txn^{comp} seja publicada antes que todos os participantes tenham a garantia de serem compensados em caso de comportamentos desonestos dos demais. De fato, se Txn_i^{dep} fosse prontamente publicada, Txn^{comp} poderia ser publicada sem que P_i recebesse qualquer $Txn_{i\to j}^{pen}$. Dessa forma os demais participantes poderiam pular a etapa E3 de revelação sem sofrer penalização.

Uma forma mais natural de impedir que Txn^{comp} fosse publicada antes dos participantes receberem $Txn_{i\to j}^{pen}$ seria deixá-la incompleta. Ou seja, um participante P_i só preencheria a $Entrada_i$ quando recebesse todas as transações $Txn_{i\to j}^{pen}$. No entanto, isso não é possível na versão atual do *Bitcoin*. Isso porque, no protocolo *Bitcoin*, as entradas de $Txn_{i\to j}^{pen}$ referenciam as saídas de Txn^{comp} através do campo *id* de Txn^{comp}. Esse campo, por sua vez, é o valor *hash* calculado sobre Txn^{comp}, *incluindo sua lista de entradas*. Por essa razão, $Txn_{i\to j}^{pen}$ só pode ser criada quando Txn^{comp} estiver completa, uma vez que ao ter suas entradas modificadas, o *id* de Txn^{comp} mudaria

[7] O valor τ foi especificado na criação das transações $Txn_{i\to j}^{pen}$.

e invalidaria todas as transações $Txn^{pen}_{i\to j}$ criadas. Por isso precisamos manter as transações Txn^{dep}_i escondidas.

Outra transação criada na etapa E0 é a Txn^{comp}. Ela é inicializada por P_1 com suas saídas completas e entradas "em branco". A partir daí, cada participante preenche a entrada correspondente com sua assinatura e passa essa versão parcial adiante. Dessa forma, quando P_m preenche $Entrada_m$, a transação Txn^{comp} se torna completa e é enviada aos demais participantes.

Quando a transação Txn^{comp} está completa, os participantes criam as transações $Txn^{pen}_{i\to j}$. Elas são criadas pelos participantes que podem publicá-las (P_j), pois são eles que definem como devem ser as saídas dessa transação. A entrada de $Txn^{pen}_{i\to j}$ possui informações secretas de P_i e P_j exigindo, portanto, que esses participantes cooperem. A cooperação se dá de forma semelhante à da criação de Txn^{comp}, com a diferença de ser bem mais simples e envolver apenas dois participantes. Dessa cooperação ressaltamos apenas que P_j só deve assinar a transação $Txn^{pen}_{i\to j}$ *depois* de recebê-la assinada de P_i. Caso contrário, o próprio P_i poderia publicá-la.

Se um participante detectar qualquer comportamento desonesto na etapa E0, o que pode ocorrer durante a criação de Txn^{comp} e $Txn^{pen}_{i\to j}$, ele deve abortar a execução. Nesse caso, ninguém ganha ou perde moedas e nem descobre qualquer coisa sobre os valores que seriam revelados na etapa E3.

O restante da Definição 8.112 é dedicado a fazer o mapeamento entre as etapas da funcionalidade f_{cmm} e a publicação das transações criadas.

Os depósitos-garantia, da etapa E1 de f_{cmm}, fazem com que os participantes percam o controle sobre suas moedas enquanto a execução durar. Isso é alcançado, em Π_{cmm}, com a publicação de Txn^{comp}. Isso exige, por sua vez, que *todas* as transações Txn^{dep}_i tenham sido publicadas.

A etapa E2 de reembolso prematuro acontece sempre que algum participante age desonestamente na etapa E1. No protocolo Π_{cmm} isso significa dizer que a transação Txn^{comp} não foi publicada. Isso pode acontecer, por exemplo, se alguma transação Txn^{dep}_i for inválida ou não tiver sido publicada[8].

Na etapa E3 os participantes criam suas transações Txn^{resg}_i sem precisar interagir com os demais. Ao publicar Txn^{resg}_i, ainda em E3, o participante P_i revela uma entrada que satisfaz φ_i. Como as transações aparecem publicamente na *blockchain*, essa entrada torna-se conhecida por todos os demais participantes. É dessa forma que o protocolo Π_{cmm} implementa a etapa de revelação definida em f_{cmm}.

Na etapa E4, os participantes que pularam a etapa E3 são penalizados. Isso se dá através da publicação das transações $Txn^{pen}_{i\to j}$. Uma transação $Txn^{pen}_{i\to j}$ tem o efeito de impedir P_i de recuperar suas moedas investidas, fazendo com que ele tenha um saldo final negativo. Tal participante P_i é considerado desonesto porque $Txn^{pen}_{i\to j}$ só pode ser publicada se Txn^{resg}_i não tiver aparecido na *blockchain*.

[8]Note que, até a etapa E1, as transações Txn^{dep}_i são secretas e, portanto, ninguém conhece a validade delas exceto seus criadores.

Finalmente, ressaltamos que, na rodada E0, fazemos menção a "rodadas preliminares", denotadas por τ'_i. Com isso pretendemos explicitar a diferença entre as rodadas dessa etapa e as das demais. As rodadas preliminares não são atreladas ao protocolo *Bitcoin* e, portanto, não precisam ser medidas em blocos. Elas servem para sincronizar o processo de criação das transações. As rodadas das demais etapas, por sua vez, são projetadas para tirar proveito de uma característica do protocolo *Bitcoin*. Na prática, as rodadas preliminares podem ser curtas e, por isso, o tempo de execução de Π_{cmm} é dominado pelas "rodadas *Bitcoin*" das etapas E1-E4, embora elas sejam menos numerosas[9].

8.6 Análise de Π_{rec}^{cmm} e sua Implementação *Bitcoin*

O protocolo Π_{rec}^{cmm} é uma proposta para se alcançar justiça na reconstrução de um segredo com uma quantidade *constante* de rodadas. Ele se baseia na funcionalidade f_{cmm}, que tem objetivo similar ao da f_{cr} utilizada pela construção escada. A diferença essencial entre essas funcionalidades é que f_{cmm} lida com depósitos de *vários* participantes simultaneamente, ao passo que f_{cr} lida com depósitos de apenas dois participantes por vez.

Apesar de f_{cmm} ter uma quantidade constante de rodadas, sua implementação Π_{cmm} baseada no *Bitcoin* é conduzida em $O(m)$ rodadas. Isso ocorre porque, além das rodadas necessárias para a realização das etapas definidas em f_{cmm}, precisamos das chamadas *rodadas preliminares* em Π_{cmm}. Essas rodadas servem para coordenar os participantes na criação das transações, de modo a tornar factíveis as ações previstas para a rodada τ'_i ($i > 0$) sempre que as previstas para a rodada τ'_{i-1} tiverem sido tomadas.

Contrastando com as rodadas preliminares temos as "rodadas *Bitcoin*". Como o nome sugere, é durante essas rodadas que os participantes de Π_{cmm} interagem com o *Bitcoin* e, em especial, publicam transações na *blockchain*. Elas ajudam a estabelecer uma ordem de publicação, de modo que uma transação pode garantidamente ser publicada na rodada τ_i ($i > 0$) se a rodada τ_{i-1} tiver transcorrido sem problemas. No entanto, mais do que essa ordenação, as rodadas *Bitcoin* garantem a resistência de Π_{cmm} contra ataques de gasto-duplo. De fato, nós já argumentamos que essas rodadas, medidas em blocos, precisam ser longas o suficiente para comportar uma quantidade segura de confirmações (T_{seg}).

A diferença essencial entre as rodadas preliminares e as *Bitcoin* é que as preliminares não precisam ser medidas em blocos. Isso é essencial para a nossa argumentação de que Π_{cmm} e, consequentemente, Π_{rec}^{cmm} são escaláveis. Esperamos que as rodadas preliminares sejam *muito mais curtas* que as *Bitcoin*. De fato, o tamanho mínimo de uma rodada *Bitcoin*, de modo a comportar as 6 confirmações recomendadas, é de 6 blocos, ou seja, de 1 hora em média. Enquanto isso, uma rodada preliminar deve apenas ser longa o suficiente para permitir a transmissão de transações e o cálculo de algumas funções criptográficas, como as relacionadas a assinaturas

[9] De fato, existem O(m) rodadas preliminares e O(1) rodadas *Bitcoin*. No entanto, como já argumentamos anteriormente, cada rodada *Bitcoin* dura, pelo menos, 1 hora.

digitais. Por isso, o tempo total de execução de Π_{cmm} deve ser dominado por suas 3 rodadas *Bitcoin*, mesmo para valores grandes de m.

A escalabilidade de Π_{cmm}, em relação ao tempo de execução, segue principalmente do fato de que a quantidade de rodadas *Bitcoin* necessárias não depende de m. Além disso, ressaltamos que, embora existam $O(m)$ rodadas preliminares, cada participante só toma parte em até 4 delas: 1 para criar Txn_i^{dep}, 1 para ajudar a criar Txn^{comp} e 2 para criar $Txn_{i\to j}^{pen}$.

Outra característica importante de Π_{rec}^{cmm} é sua simetria em relação ao investimento inicial feito pelos participantes. Nele, é necessário que todos façam um depósito de $(m-1)\cdot q$ moedas no início da execução. Isso contrasta com a construção escada em que, por exemplo, P_1 investe somente q moedas enquanto P_m investe $(m-1)\cdot q$ e os demais, um valor intermediário. Usamos essa observação também para ressaltar que o nosso investimento inicial, além de simétrico, é compatível com os da construção escada. Isso é importante porque, embora o investimento inicial seja imprescindível para se alcançar justiça monetária, para os participantes *honestos* ele funciona essencialmente como uma "barreira" inicial. Isso porque eles precisam juntar moedas apenas para, no final, recebê-las de volta. Nosso protocolo, em relação à construção escada, não aumenta essa barreira.

O investimento inicial exigido por Π_{rec}^{cmm} também tem impacto sobre sua escalabilidade, uma vez que, para um número grande de participantes, tal investimento pode se tornar proibitivo. Seu valor absoluto, no entanto, pode ser regulado pelo parâmetro q de multa por aborto. De maneira geral, o valor do investimento inicial tem que acompanhar a "importância" do resultado da computação para fazer com que comportamentos maliciosos não valham a pena. Sob essa perspectiva, a quantidade de participantes pode ser um indicativo dessa importância e, portanto, sua influência no valor do investimento pode ser justificável.

Como ponto negativo de Π_{cmm} ressaltamos o tamanho da transação Txn^{comp}, que é da ordem de $O(m^2)$. Essa é uma característica ruim porque o protocolo *Bitcoin* impõe um tamanho máximo aos blocos e, consequentemente, às transações. Além disso, o tamanho de uma transação, como especificado em WIKI (2015e), influencia diretamente o valor mínimo da gorjeta que deve ser dada aos mineradores[10] . Por outro lado, esse é um problema que pode ser amenizado em versões futuras do *Bitcoin*, com o ajuste dos parâmetros que regulam o tamanho dos blocos e a gorjeta mínima.

Outra maneira de contornar esse problema seria considerar versões um pouco mais complexas das transações envolvidas em Π_{cmm}. Poderíamos considerar uma versão de Txn^{comp} tal que todas as saídas pertencentes ao mesmo conjunto S_i fossem transformadas em uma só. Para reivindicar tal saída seria necessário apresentar a assinatura dos m participantes. Por isso seria necessário a cooperação de todos os participantes para construir cada Txn_i^{resg}. Além disso, para cada i, $Txn_{i\to j}^{pen}$ seria transformada em uma única transação com $m-1$ saídas. Ou seja, basicamente transferiríamos o conjunto S_i de Txn^{comp} para a nova Txn_i^{pen}. A publicação de

[10]Uma transação que não dê a quantidade mínima de gorjeta tem menor prioridade para os mineradores. Por isso sua publicação pode levar mais tempo.

Txn_i^{pen} penalizaria imediatamente P_i, mas os demais participantes precisariam de mais uma transação para reivindicar a saída adequada de Txn_i^{pen}. Apesar de envolver mais transações e necessitar de mais cooperação entre os participantes, cada transação teria, no máximo, um tamanho proporcional a m. Consideramos a descrição e análise mais detalhadas dessa extensão de Π_{cmm} como um possível desenvolvimento deste trabalho.

9 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

9.1 Considerações Finais

Ao longo dos Capítulos 6, 7 e 8 tratamos da reconstrução justa e de duas estratégias para conduzí-la que podem ser implementadas tendo o *Bitcoin* como plataforma. Justamente o emprego do *Bitcoin* por essas estratégias merece algumas considerações adicionais às que já foram feitas.

A primeira dessas considerações diz respeito à representação dos circuitos φ_i que aparecem tanto na construção escada quanto no protocolo Π_{rec}^{cmm}. Esses circuitos precisam ser representados como *scripts* para serem utilizados pelas transações *Bitcoin* e, portanto, têm sua complexidade limitada pela expressividade da linguagem de *script* do *Bitcoin*. De fato, essa linguagem não é Turing-completa e, além disso, alguns de seus comandos originais foram removidos por questões de segurança, como descrito em WIKI (2015g). Portanto, a complexidade de φ_i é limitada pela expressividade dos *scripts Bitcoin*.

O segundo ponto que precisamos discutir diz respeito ao problema da maleabilidade das transações do *Bitcoin*. Lembramos que esse problema foi discutido na Seção 2.2. Vamos considerar seus efeitos na nossa proposta, mais especificamente sobre as transações Txn^{comp} e $Txn_{i\to j}^{pen}$. Lembramos que $Txn_{i\to j}^{pen}$ referencia Txn^{comp} *antes* que ela apareça na *blockchain*. Depois que todas as transações $Txn_{i\to j}^{pen}$ forem criadas, um participante malicioso pode, devido à maleabilidade das transações, publicar uma versão Txn_2^{comp} equivalente a Txn^{comp} mas com *id* diferente. Dessa forma, todas as transações $Txn_{i\to j}^{pen}$ tornam-se inválidas.

Como mencionamos anteriormente, todas as transações que referenciam outras ainda não publicadas na *blockchain* estão vulneráveis a esse problema. Em particular, também a construção escada sofre desse problema, pois $Txn_{i\to j}^{reem}$ referencia uma transação ainda não publicada ($Txn_{i\to j}^{cr}$).

Além do problema da maleabilidade, a maneira pela qual os *ids* das transações são gerados tem outro efeito indesejado nas construções apresentadas. Ela obriga que uma transação só possa referenciar uma outra que já esteja *completa*, isto é, com todas as entradas devidamente preenchidas. De fato, se um transação ainda não está completa, então seu *id* ainda não é o final. Como comentamos, contornamos essa característica, na nossa proposta, mantendo as transações Txn_i^{dep} secretas. Outra possibilidade é admitir uma versão modificada do *Bitcoin*, como as discutidas na Seção 2.2. Ressaltamos que nossa proposta pode ser conduzida da mesma maneira nessas versões. De fato, ela pode até mesmo ser simplificada.

Outro ponto revelante sobre as construções apresentadas é que elas caracterizam o aborto de um participante através de sua demora em tomar parte em alguma etapa. Isso é alcançado através do campo *timelock* das transações. No entanto, como as comunicações na rede *Bitcoin* se dão sem garantias, no esquema de melhor esforço (*best effort*), não há como de fato ter certeza se um participante abortou ou se simplesmente sua transação *ainda* não apareceu na *blockchain*.

De maneira geral, podemos ajustar o tempo de espera de acordo com a taxa média de geração de blocos que, no *Bitcoin*, é de um a cada 10 minutos.

Esse tipo de estratégia pode sofrer com ataques de negação de serviço, como aponta BEEKMAN (2015). Isso porque um participante sob um desses ataques pode não conseguir enviar suas transações a tempo. Por outro lado, não apenas um participante, mas a própria rede *Bitcoin* pode sofrer com ataques desse tipo, como o descrito em BITSCAN (2015).

Por outro lado, como argumenta KUMARESAN; MORAN; BENTOV (2015), podemos admitir, em geral, que tais ataques são incomuns e que os participantes possuem uma conexão de boa qualidade com a *Internet*. Além disso, o próprio protocolo *Bitcoin* possui mecanismos para tornar ataques de negação de serviço caros e difíceis de serem conduzidos, como é argumentado em WIKI (2015h).

9.2 Trabalhos Futuros

Acreditamos que o trabalho atual pode ser continuado e aprimorado de diversas maneiras. Consideramos interessante, por exemplo, a possibilidade de se reduzir a distância entre os resultados altamente teóricos do Capítulo 4 e as construções predominantemente práticas do Capítulo 6. Como a computação segura de uma funcionalidade *qualquer* exige, naturalmente, um grande e complexo ferramental criptográfico, pode-se investigar a possibilidade de utilizar as construções propostas para a computação de funcionalidades simples.

No Capítulo 6 comentamos algumas limitações que o *Bitcoin* impõe às construções apresentadas. Entre elas, citamos a possibilidade de ataques de negação de serviço, a maleabilidade de transações e a falta de expressividade da linguagem de *script* utilizada. Por isso, uma continuação frutífera deste trabalho seria o estudo e adaptação das formalizações da *blockchain* que têm sido propostas. Ou seja, trazer a *blockchain* para o "mundo matemático" para que as construções propostas não dependam diretamente do *Bitcoin* ou de qualquer outra criptomoeda ou sistema similar específico.

Além disso, apontamos outro benefício de se adotar uma formalização da *blockchain*: a prova formal da segurança das construções propostas. Ressaltamos que a apresentação dessas construções seguiu um padrão em que (i) primeiramente especificamos uma funcionalidade ideal, (ii) em seguida propomos um protocolo para implementá-la, e, finalmente, (iii) apresentamos uma implementação do protocolo em (ii) utilizando o *Bitcoin* como plataforma. No entanto, fomos capazes de provar apenas que os protocolos propostos em (ii) eram de fato realizações seguras das funcionalidades em (i). Não foi possível garantir formalmente que a implementação *Bitcoin* era também segura.

Por fim, ressaltamos que existem outros trabalhos, inclusive citados por nós, como ANDRYCHOWICZ et al. (2014b) e ANDRYCHOWICZ et al. (2014a), que não foram detalhados aqui mas que podem, posteriormente, vir a ser objetos de estudo mais aprofundado. Por exemplo, um possível desenvolvimento deste trabalho poderia começar com a disponibilização de provas

para os Lemas 1 e 2 de ANDRYCHOWICZ et al. (2014a)[1].

[1]Em maio/2015 entramos em contato com os autores de ANDRYCHOWICZ et al. (2014a) porque, nesse trabalho, eles afirmam que *"The security proofs will appear in an extended version of this paper."* No entanto, a resposta que recebemos foi da forma: *"unfortunately the extended version with the proofs does not exist."*

REFERÊNCIAS

ANDRYCHOWICZ, M. et al. Fair Two-Party Computations via Bitcoin Deposits. In: BöHME, R. et al. (Ed.). **Financial Cryptography and Data Security**. [S.l.]: Springer Berlin Heidelberg, 2014. p.105–121. (Lecture Notes in Computer Science, v.8438).

ANDRYCHOWICZ, M. et al. Secure Multiparty Computations on Bitcoin. In: IEEE SYMPOSIUM ON SECURITY AND PRIVACY (SP'14), Washington. **Proceedings...** IEEE Computer Society, 2014. p.443–458.

ANDRYCHOWICZ, M. et al. On the Malleability of Bitcoin Transactions. In: BRENNER, M. et al. (Ed.). **Financial Cryptography and Data Security**. [S.l.]: Springer Berlin Heidelberg, 2015. p.1–18. (Lecture Notes in Computer Science, v.8976).

ASHAROV, G. Towards Characterizing Complete Fairness in Secure Two-Party Computation. In: THEORY OF CRYPTOGRAPHY CONFERENCE (TCC 2014), 11., San Diego. **Proceedings...** Springer Berlin Heidelberg, 2014. p.291–316.

ASHAROV, G.; LINDELL, Y.; ZAROSIM, H. Fair and Efficient Secure Multiparty Computation with Reputation Systems. In: INTERNATIONAL CONFERENCE ON THE THEORY AND APPLICATION OF CRYPTOLOGY AND INFORMATION SECURITY (ASIACRYPT 2013), 19., Bengaluru. **Proceedings...** Springer Berlin Heidelberg, 2013. p.201–220.

BEEKMAN, J. G. A Denial of Service attack against fair computations using Bitcoin deposits. **Information Processing Letters**, [S.l.], v.116, n.2, p.144–146, February 2015.

BENTOV, I.; KUMARESAN, R. How to Use Bitcoin to Design Fair Protocols. In: ANNUAL CRYPTOLOGY CONFERENCE (CRYPTO 2014), 34., Santa Barbara. **Proceedings...** Springer Berlin Heidelberg, 2014. p.421–439.

BITSCAN. **Clogged**: the anatomy of a bitcoin attack. Disponível em: <https://bitscan.com/articles/clogged-the-anatomy-of-a-bitcoin-attack>. Acesso em: 28 nov. 2015.

BONEH, D.; NAOR, M. Timed commitments (Extended Abstract). In: ANNUAL INTERNATIONAL CRYPTOLOGY CONFERENCE (CRYPTO 2000), 20., Santa Barbara. **Proceedings...** Springer Berlin Heidelberg, 2000. p.236–254.

CANETTI, R. Universally composable security: a new paradigm for cryptographic protocols. In: IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE (FOCS), 42., Las Vegas. **Proceedings...** IEEE, 2001. p.136–145.

CLEVE, R. Limits on the Security of Coin Flips when Half the Processors Are Faulty. In: ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING (STOC'86), 18., New York. **Proceedings...** ACM, 1986. p.364–369.

FOUNDATION, B. **Input, Transaction Input, TxIn**. Disponível em: <https://bitcoin.org/en/glossary/input>. Acesso em: 9 jul. 2015.

FOUNDATION, B. **Output, Transaction Output, TxOut**. Disponível em: <https://bitcoin.org/en/glossary/output>. Acesso em: 9 jul. 2015.

FOUNDATION, B. **Contracts**. Disponível em: <https://bitcoin.org/en/developer-guide#contracts>. Acesso em: 26 jun. 2015.

GOLDREICH, O. **Foundations of Cryptography**: basic tools. New York: Cambridge University Press, 2001. v.1.

GOLDREICH, O. **Foundations of Cryptography**: basic applications. New York: Cambridge University Press, 2004. v.2.

GOLDWASSER, S.; LEVIN, L. A. Fair Computation of General Functions in Presence of Immoral Majority. In: ANNUAL INTERNATIONAL CRYPTOLOGY CONFERENCE (CRYPTO'90), 10., Santa Barbara. **Proceedings...** Springer Berlin Heidelberg, 1990. p.77–93.

GORDON, S. D. et al. Complete Fairness in Secure Two-Party Computation. **Journal of the ACM (JACM)**, New York, v.58, n.6, p.1–37, 2011.

KATZ, J.; LINDELL, Y. **Introduction to Modern Cryptography**: principles and protocols. 2.ed. New York: Chapman and Hall/CRC, 2014.

KUMARESAN, R.; BENTOV, I. How to Use Bitcoin to Incentivize Correct Computations. In: ACM SIGSAC CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY (CCS'14), Scottsdale. **Proceedings...** ACM, 2014. p.30–41.

KUMARESAN, R.; MORAN, T.; BENTOV, I. How to Use Bitcoin to Play Decentralized Poker. In: ACM SIGSAC CONFERENCE ON COMPUTER AND COMMUNICATIONS SECURITY (CCS'15), 22., Denver. **Proceedings...** ACM, 2015. p.195–206.

KÜPÇÜ, A.; LYSYANSKAYA, A. Usable Optimistic Fair Exchange. **Computer Networks**, New York, v.56, n.1, p.50–63, 2012.

NAKAMOTO, S. **Bitcoin**: a peer-to-peer electronic cash system. Disponível em: <https://bitcoin.org/bitcoin.pdf>. Acesso em: 7 de dez. 2015.

PAGNIA, H.; GÄRTNER, F. C. **On the impossibility of fair exchange without a trusted third party**. Darmstadt, Germany: University of Technology, Department of Computer Science, 1999. (Technical Report TUD-BS-1999-02).

WIKI, B. **Transacion**. Disponível em: <https://en.bitcoin.it/w/index.php?title=Transaction&oldid=56666>. Acesso em: 9 jul. 2015.

WIKI, B. **Contracts**. Disponível em: <https://en.bitcoin.it/w/index.php?title=Contracts&oldid=57023>. Acesso em: 26 jun. 2015.

WIKI, B. **Block**. Disponível em: <https://en.bitcoin.it/w/index.php?title=Block&oldid=56508>. Acesso em: 14 jul. 2015.

WIKI, B. **Block hashing algorithm**. Disponível em: <https://en.bitcoin.it/w/index.php?title=Block_hashing_algorithm&oldid=56896>. Acesso em: 14 jul. 2015.

WIKI, B. **Transaction Fees**. Disponível em: <https://en.bitcoin.it/w/index.php?title=Transaction_fees&oldid=56859>. Acesso em: 30 jun. 2015.

WIKI, B. **Confirmation**. Disponível em: <https://en.bitcoin.it/w/index.php?title=Confirmation&oldid=51011>. Acesso em: 9 jul. 2015.

WIKI, B. **Script**. Disponível em: <https://en.bitcoin.it/w/index.php?title=Script&oldid=59356>. Acesso em: 6 dez. 2015.

WIKI, B. **Weaknesses**. Disponível em: <https://en.bitcoin.it/w/index.php?title=Weaknesses&oldid=57353#Denial_of_Service_.28DoS.29_attacks>. Acesso em: 28 nov. 2015.

YAO, A. C. Protocols for Secure Computations. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE (SFCS '82), 23., Washington. **Proceedings...** IEEE, 1982. p.160–164.

APÊNDICE A - PROVA DA IMPOSSIBILIDADE DE CLEVE (1986)

Por conveniência vamos reescrever a definição da funcionalidade cara ou coroa relaxada, $f^{\varepsilon}_{c/c}$ a seguir:

Definição A.113 (Funcionalidade cara ou coroa relaxada)**.** *Sejam A e B dois participantes, n um parâmetro de segurança e $\varepsilon \in [0, \frac{1}{2}]$ um valor predefinido. Definimos a versão relaxada de $f_{c/c}$ como a 2-funcionalidade $f^{\varepsilon}_{c/c}$ tal que:*

$$f^{\varepsilon}_{c/c}(1^n, 1^n) \stackrel{def}{=} (b_A, b_B),$$

com $b_A, b_B \in \{0,1\}$, $Pr[b_A = b_B] \geq \frac{1}{2} + \varepsilon$ e $viés(b_A), viés(b_B)$ desprezíveis em n.

Antes de provarmos o Teorema 5.92, vamos definir alguns conceitos auxiliares, com base na apresentação feita em CLEVE (1986).

Definição A.114 (Protocolo para computação de $f^{\varepsilon}_{c/c}$)**.** *Seja n um parâmetro de segurança. Um protocolo para computação de $f^{\varepsilon}_{c/c}$ é definido como um par de participantes (A^n, B^n) que se comunicam entre si para gerar os bits b_A, b_B em tempo $poly(n)$. Assumimos que essa computação se dê em $r(n)$ rodadas, para alguma função r limitada por $poly(n)$. Cada uma dessas rodadas consiste nos seguintes eventos: (i) A^n conduz alguma computação e (ii) envia uma mensagem para B^n que então (iii) conduz alguma computação e (iv) envia uma mensagem de volta para A^n. Dessa forma, as estratégias de A^n e B^n podem ser vistas como duas séries de $r(n)+1$ circuitos, cada um limitado por $poly(n)$ em tamanho. A saída dos primeiros $r(n)$ circuitos são informações de estado do protocolo e mensagens para o outro participante. Os últimos circuitos, por sua vez, apenas calculam o bit de saída do participante. Assumimos que cada participante tem acesso a uma fonte (privada) de aleatoriedade.*

De acordo com a Definição 5.91, reescrita na Definição A.113, um protocolo que implementa a funcionalidade $f^{\varepsilon}_{c/c}$ precisa garantir duas propriedades: a de *consenso* ($Pr[b_A = b_B] \geq \frac{1}{2} + \varepsilon$) e a de *aleatoriedade* ($viés(b_A), viés(b_B)$ desprezíveis em n). Em relação à propriedade de consenso apresentamos a seguinte definição.

Definição A.115 (ε-consistência)**.** *Seja n um parâmetro de segurança. Definimos que um protocolo (A^n, B^n) para $f^{\varepsilon}_{c/c}$ é ε-consistente se (A^n, B^n) atende à propriedade de consenso.*

Como argumentado em CLEVE (1986), não se pode esperar que um protocolo garanta consenso se um de seus participantes for desonesto, uma vez que ele pode gerar um *bit* qualquer como saída. Por isso, sempre que necessário, tomaremos a propriedade de consenso por hipótese.

A seguir definiremos a segurança de um protocolo para $f^{\varepsilon}_{c/c}$. Note que ela não precisa ser geral, como as definições do Capítulo 4, porque precisamos lidar apenas com uma funcionalidade específica.

Definição A.116 (Segurança de um protocolo (A^n, B^n) para $f^{\varepsilon}_{c/c}$)**.** *Seja n um parâmetro de segurança e (A^n, B^n) um protocolo para computar $f^{\varepsilon}_{c/c}$. Definimos que (A^n, B^n) é seguro se o viés da saída do participante honesto for desprezível em n mesmo que uma das estratégias A^n, B^n seja substituída por uma estratégia desonesta. Ou seja, se b_H é a saída do participante honesto então, para toda função f, com $f(n) \leq O\left(\frac{1}{n^k}\right)$ para todo k, temos que $viés(b_H) \leq f(n)$. Por consequência, para um n grande, torna-se infactível enviesar a saída de um participante honesto em um protocolo seguro.*

Com as duas definições anteriores, vamos enunciar e provar uma versão preliminar do Teorema 5.92.

Teorema A.117 (Impossibilidade de justiça perfeita sob a hipótese de ε-consistência CLEVE (1986))**.** *Se (A^n, B^n) é um protocolo ε-consistente para computar $f^{\varepsilon}_{c/c}$, então (A^n, B^n) não é seguro. Mais especificamente, para todo n existe uma estratégia desonesta que faz com que o viés da saída do participante honesto seja pelo menos $\frac{\varepsilon}{4r(n)+1}$, em que $r(n)$ é o número de rodadas do protocolo.*

Demonstração. A ideia geral da prova é investigar o impacto de várias estratégias desonestas no viés do participante honesto. Essas estratégias utilizam a capacidade do adversário abortar.

Definimos que um participante *aborta* o protocolo na rodada i se a partir dela em diante ele passa a enviar mensagens nulas (cadeias binárias consistindo apenas de zeros). Se B^n aborta na rodada i, por exemplo, então A^n tem como saída um *bit* a_i, o qual chamaremos de *saída padrão*. Naturalmente A^n consegue calcular a_i no início da rodada i, mesmo antes de receber qualquer mensagem de B^n.

Definimos a_i como a saída padrão de A^n para a rodada i, para $i \in \{1, \cdots, r(n)\}$. De maneira semelhante, b_i é a saída padrão de B^n para a rodada $i+1$, para $i \in \{0, \cdots, r(n)-1\}$. Se B^n não abortar o protocolo, a saída de A^n será $a_{r(n)+1}$. Se A^n não abortar o protocolo, a saída de B^n será $b_{r(n)}$.

Vamos definir o viés de um *bit b em direção a 0* como $Pr[b = 0] - \frac{1}{2}$. O *viés em direção a 1* é definido de maneira semelhante.

Apresentaremos agora $4r(n)+1$ estratégias desonestas, derivadas das estratégias honestas A^n e B^n. São elas $\tilde{A}^n$, $A^n_{i,0}$, $A^n_{i,1}$, $B^n_{i,0}$ e $B^n_{i,1}$, para todo $i \in \{1, \cdots, r(n)\}$.

A estratégia $\tilde{A}^n$ é tal que $\tilde{A}^n$ aborta na rodada 1. Nesse caso a saída de B^n é b_0 e o $viés(b_0)$ é:

$$viés(b_0) = max\{Pr[b_0 = 0], Pr[b_0 = 1]\} - \frac{1}{2}. \tag{A.1}$$

As demais estratégias são definidas na Tabela A.1 e na Tabela A.2.

Nosso objetivo é mostrar que pelo menos uma dessas estratégias desonestas causa um viés na saída do participante honesto de valor pelo menos $\frac{\varepsilon}{4r(n)+1}$. Vamos começar calculando o viés de cada uma dessas estratégias[1]. O viés em direção a 0 da saída de B^n quanto o protocolo

[1] Note que o viés de um *bit* é o maior entre o seu viés em direção a 0 e em direção a 1.

Tabela A.1: Estratégias $A^n_{i,0}$ e $A^n_{i,1}$

Estratégia $A^n_{i,0}$:	**Estratégia $A^n_{i,1}$:**
Simula A^n nas rodadas $1,2,\cdots,i-1$	Simula A^n nas rodadas $1,2,\cdots,i-1$
Computa a_i	Computa a_i
Se $a_i = 0$ então:	Se $a_i = 1$ então:
Simula A^n na rodada i	Simula A^n na rodada i
Aborta na rodada $i+1$	Aborta na rodada $i+1$
Senão:	Senão:
Aborta na rodada i	Aborta na rodada i

Tabela A.2: Estratégias $B^n_{i,0}$ e $B^n_{i,1}$

Estratégia $B^n_{i,0}$:	**Estratégia $B^n_{i,1}$:**
Simula B^n nas rodadas $1,2,\cdots,i-1$	Simula B^n nas rodadas $1,2,\cdots,i-1$
Computa b_i	Computa b_i
Se $b_i = 0$ então:	Se $b_i = 1$ então:
Simula B^n na rodada i	Simula B^n na rodada i
Aborta na rodada $i+1$	Aborta na rodada $i+1$
Senão:	Senão:
Aborta na rodada i	Aborta na rodada i

$(A^n_{i,0}, B^n)$ é executado é:

$$Pr[a_i = 0 \wedge b_i = 0] + Pr[a_i = 1 \wedge b_{i-1} = 0] - \frac{1}{2} \tag{A.2}$$

O viés em direção a 1 da saída de B^n quando o protocolo $(A^n_{i,1}, B^n)$ é executado é:

$$Pr[a_i = 1 \wedge b_i = 1] + Pr[a_i = 0 \wedge b_{i-1} = 1] - \frac{1}{2} \tag{A.3}$$

O viés em direção a 0 da saída de A^n quando o protocolo $(A^n, B^n_{i,0})$ é executado é:

$$Pr[b_i = 0 \wedge a_{i+1} = 0] + Pr[b_i = 1 \wedge a_i = 0] - \frac{1}{2} \tag{A.4}$$

O viés em direção a 1 da saída de A^n quando o protocolo $(A^n, B^n_{i,1})$ é executado é:

$$Pr[b_i = 1 \wedge a_{i+1} = 1] + Pr[b_i = 0 \wedge a_i = 1] - \frac{1}{2} \tag{A.5}$$

Vamos definir Δ como a média dos $4r(n)+1$ vieses introduzidos pelas estratégias desonestas apresentadas. Ela é calculada como a soma das Equações A.1 à A.5 dividida pelo total de valores

$(4r(n)+1)$:

$$\begin{aligned}\Delta = \frac{1}{4r(n)+1}\big[&max\{Pr[b_0=0], Pr[b_0=1]\} - \frac{1}{2} \\ &+ \sum_{i=1}^{r(n)} (Pr[a_i=0 \wedge b_i=0] + Pr[a_i=1 \wedge b_{i-1}=0] - \frac{1}{2}) \\ &+ \sum_{i=1}^{r(n)} (Pr[a_i=1 \wedge b_i=1] + Pr[a_i=0 \wedge b_{i-1}=1] - \frac{1}{2}) \\ &+ \sum_{i=1}^{r(n)} (Pr[b_i=0 \wedge a_{i+1}=0] + Pr[b_i=1 \wedge a_i=0] - \frac{1}{2}) \\ &+ \sum_{i=1}^{r(n)} (Pr[b_i=1 \wedge a_{i+1}=1] + Pr[b_i=0 \wedge a_i=1] - \frac{1}{2})\big]\end{aligned} \tag{A.6}$$

Vamos agora fazer uma série de manipulações algébricas para definir um limite inferior para o valor de Δ. Começaremos removendo as parcelas $\frac{1}{2}$ dos somatórios e usando o fato de que $Pr[a_i=0 \wedge b_i=0] + Pr[a_i=0 \wedge b_i=1] + Pr[a_i=1 \wedge b_i=0] + Pr[a_i=1 \wedge b_i=1] = 1$ para todo $i \in \{1, \cdots, r(n)\}$:

$$\begin{aligned}\Delta = \frac{1}{4r(n)+1}\big[&r(n) - 4r(n)\frac{1}{2} + max\{Pr[b_0=0], Pr[b_0=1]\} - \frac{1}{2} \\ &+ \sum_{i=1}^{r(n)} Pr[a_i=1 \wedge b_{i-1}=0] + \sum_{i=1}^{r(n)} Pr[a_i=0 \wedge b_{i-1}=1] \\ &+ \sum_{i=1}^{r(n)} Pr[b_i=0 \wedge a_{i+1}=0] + \sum_{i=1}^{r(n)} Pr[b_i=1 \wedge a_{i+1}=1]\big]\end{aligned} \tag{A.7}$$

Podemos retirar dos quatro somatórios acima as somas da forma $Pr[a_{i+1}=1 \wedge b_i=0] + Pr[a_{i+1}=0 \wedge b_i=1] + Pr[b_i=0 \wedge a_{i+1}=0] + Pr[b_i=1 \wedge a_{i+1}=1]$, para $i \in \{1, \cdots, r(n)-1\}$. Cada uma delas resulta em 1. Portanto temos:

$$\begin{aligned}\Delta = \frac{1}{4r(n)+1}\big[&r(n) - 4r(n)\frac{1}{2} + r(n) - 1 + max\{Pr[b_0=0], Pr[b_0=1]\} - \frac{1}{2} \\ &+ Pr[a_1=1 \wedge b_0=0] + Pr[a_1=0 \wedge b_0=1] + Pr[a_{r(n)+1}=0 \wedge b_{r(n)}=0] \\ &+ Pr[a_{r(n)+1}=1 \wedge b_{r(n)}=1]\big]\end{aligned} \tag{A.8}$$

Substituindo $Pr[a_1=1 \wedge b_0=0] + Pr[a_1=0 \wedge b_0=1]$ por $(1 - Pr[a_1=b_0])$ e $Pr[a_{r(n)+1}=0 \wedge b_{r(n)}=0] + Pr[a_{r(n)+1}=1 \wedge b_{r(n)}=1]$ por $Pr[a_{r(n)+1}=b_{r(n)}]$ temos que:

$$\begin{aligned}\Delta = \frac{1}{4r(n)+1}\big[&-1 + max\{Pr[b_0=0], Pr[b_0=1]\} - \frac{1}{2} + 1 \\ &- Pr[a_1=b_0] + Pr[a_{r(n)+1}=b_{r(n)}]\big]\end{aligned} \tag{A.9}$$

Note que como a_1 e b_0 podem ser calculados *antes* da rodada 1, portanto antes de qualquer interação entre os participantes, podemos considerá-los como independentes. Por isso $P[a_1 = b_0] = P[a_1 = 0]Pr[b_0 = 0] + Pr[a_1 = 1]Pr[b_0 = 1] \leq max\{Pr[b_0 = 0], Pr[b_0 = 1]\}$. Dessa forma, podemos transformar a igualdade acima na seguinte desigualdade:

$$\Delta \geq \frac{1}{4r(n)+1}\left[Pr[a_{r(n)+1} = b_{r(n)}] - \frac{1}{2}\right] \tag{A.10}$$

Usando o fato de que (A^n, B^n) é ε-consistente por hipótese, temos que:

$$\Delta \geq \frac{[\frac{1}{2} + \varepsilon - \frac{1}{2}]}{4r(n)+1} = \frac{\varepsilon}{4r(n)+1} \tag{A.11}$$

Como a média Δ dos vieses das estratégias desonestas é tal que $\Delta \geq \frac{\varepsilon}{4r(n)+1}$, podemos concluir que pelo menos uma delas gera um viés $\geq \frac{\varepsilon}{4r(n)+1}$ na saída do participante honesto. Esse valor *não* é desprezível em n.

Com isso concluímos que existe uma estratégia desonesta para (A^n, B^n) que introduz um viés não desprezível na saída do participante honesto e, portanto (A^n, B^n) não pode ser seguro. □

Agora podemos provar o Teorema 5.92, que reescrevemos a seguir.

Teorema A.118 (Impossibilidade de CLEVE (1986))**.** *Não é possível computar $f^{\varepsilon}_{c/c}$ com justiça perfeita.*

Demonstração. Seja (A^n, B^n) um protocolo para computar $f^{\varepsilon}_{c/c}$. Se (A^n, B^n) não for ε-consistente, então ele não computa $f^{\varepsilon}_{c/c}$ corretamente. Caso ele seja ε-consistente, então este teorema segue diretamente do Teorema A.117. □