Como funciona o Bitcoin?

Anatómia da Blockchain do `Bitcoin`

Enno Nagel

Biblioteca Central da UFAL – Maceió, 22 de Agosto de 2018

1. Cadeia
2. Laço
3. Bloco
4. Mineração
5. Transação

Moeda Comum versus Criptomoeda

Moeda Comum

Em uma moeda comum os negociantes confiam em um terceiro, o **banco**, que mantém um livro-razão de todas as transações.

Criptomoeda

Em uma *criptomoeda* os negociantes confiam em uma **blockchain**, literalmente *cadeia de blocos*, um livro-razão público de todas as transações; mantido (e replicado) por uma rede de milhares de computadores e praticamente infalsificável.

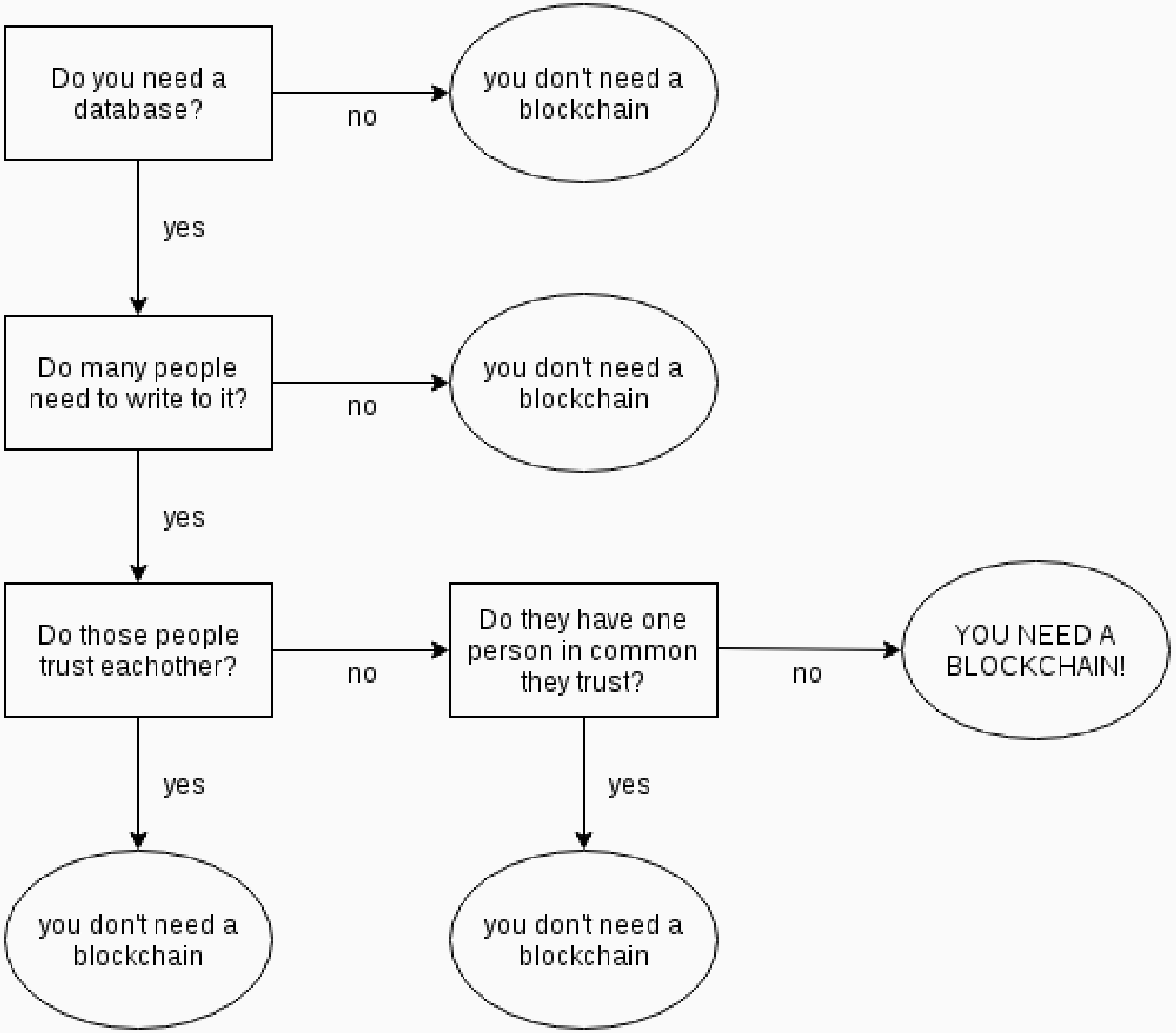


Figura 1: Necessidade de uma cadeia de blocos

Blockchain

A *blockchain* é uma cadeia de blocos que são ligados, isto é, cada bloco tem um indicador (ou *endereço*, mais exatamente, o hash) ao bloco *anterior*, isto é, ao bloco mais recente antes dele.

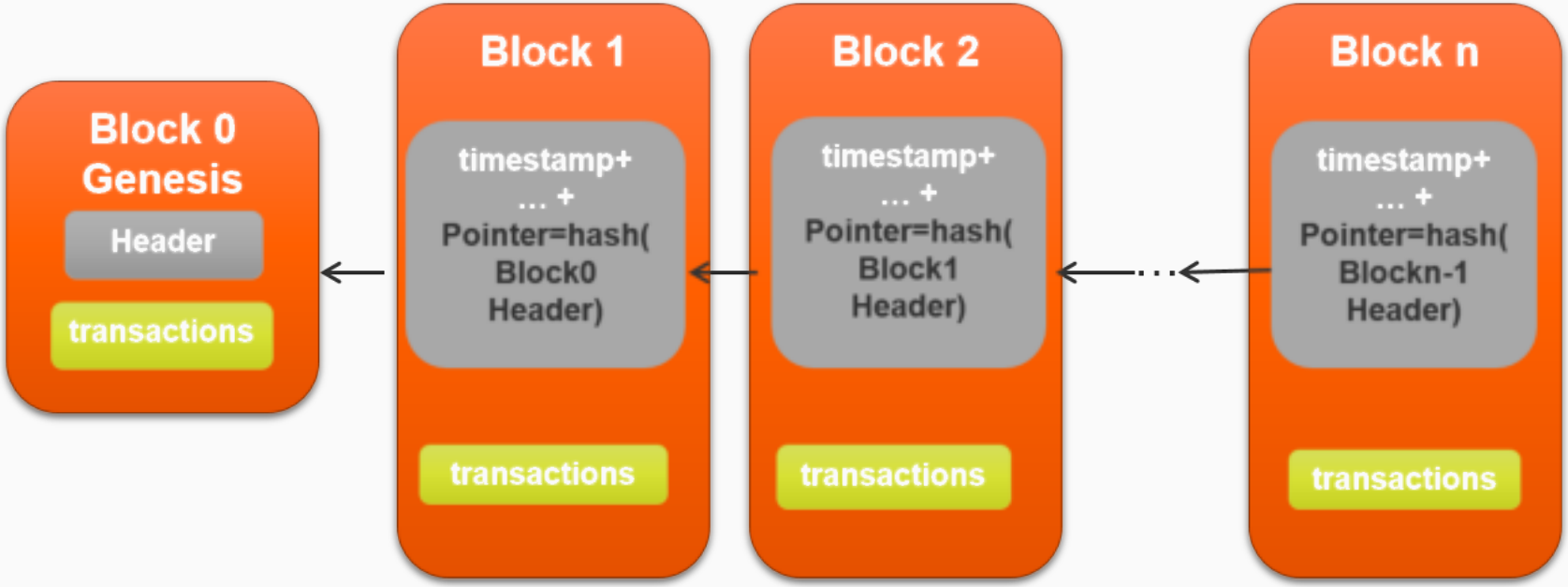


Figura 2: Blockchain

Ramificações

Ocorrem ramificações: A única cadeia considerada válida é a mais comprida; todos os blocos fora dela são considerados *orfanados*. Raramente as ramificações têm mais de um bloco (porque é dificílimo estender a cadeia por outro bloco).

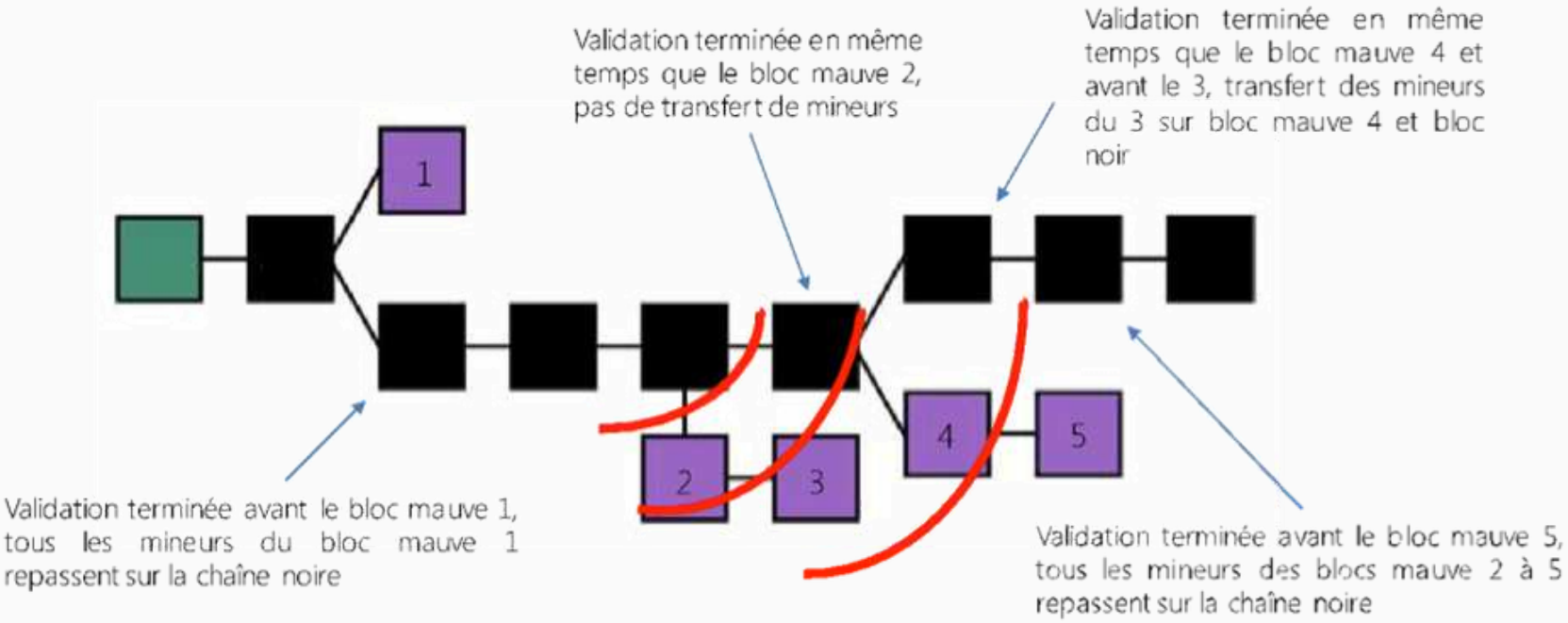


Figura 3: Gerenciamento de Bifurcações

Blocos

A blockchain do Bitcoin começou no dia 3 de janeiro de 2009 às 18:15 UTC, provavelmente por Satoshi Nakamoto, com o primeiro bloco, o *bloco de Génesis*.

O bloco consiste

- de um *cabeçalho*, os meta-dados, que contém
  - o indicador para o bloco anterior, e
  - informações sobre a sua criação.
- do seu *conteúdo*, os dados, cerca de 2000 *transações* (agrupadas em uma *árvore de Merkle* de cerca de um megabyte) entre os usuários do Bitcoin.

Reação em Cadeia

O endereço (mais exatamente, o hash) do bloco depende do seu conteúdo inteiro. Isto é, qualquer alteração, implica a alteração do seu endereço, em particular,

- a alteração de uma de suas transações,
- a alteração do endereço ao bloco anterior!

Por exemplo:

1. Se uma das suas transações muda, então o seu endereço.
2. Logo, o indicador do bloco posterior muda, logo o endereço do bloco posterior.
3. Igual para o endereço do bloco posterior ao bloco posterior, e … assim por diante.

A alteração de uma singela transação em um bloco invalida todos os endereços dos seus blocos posteriores!

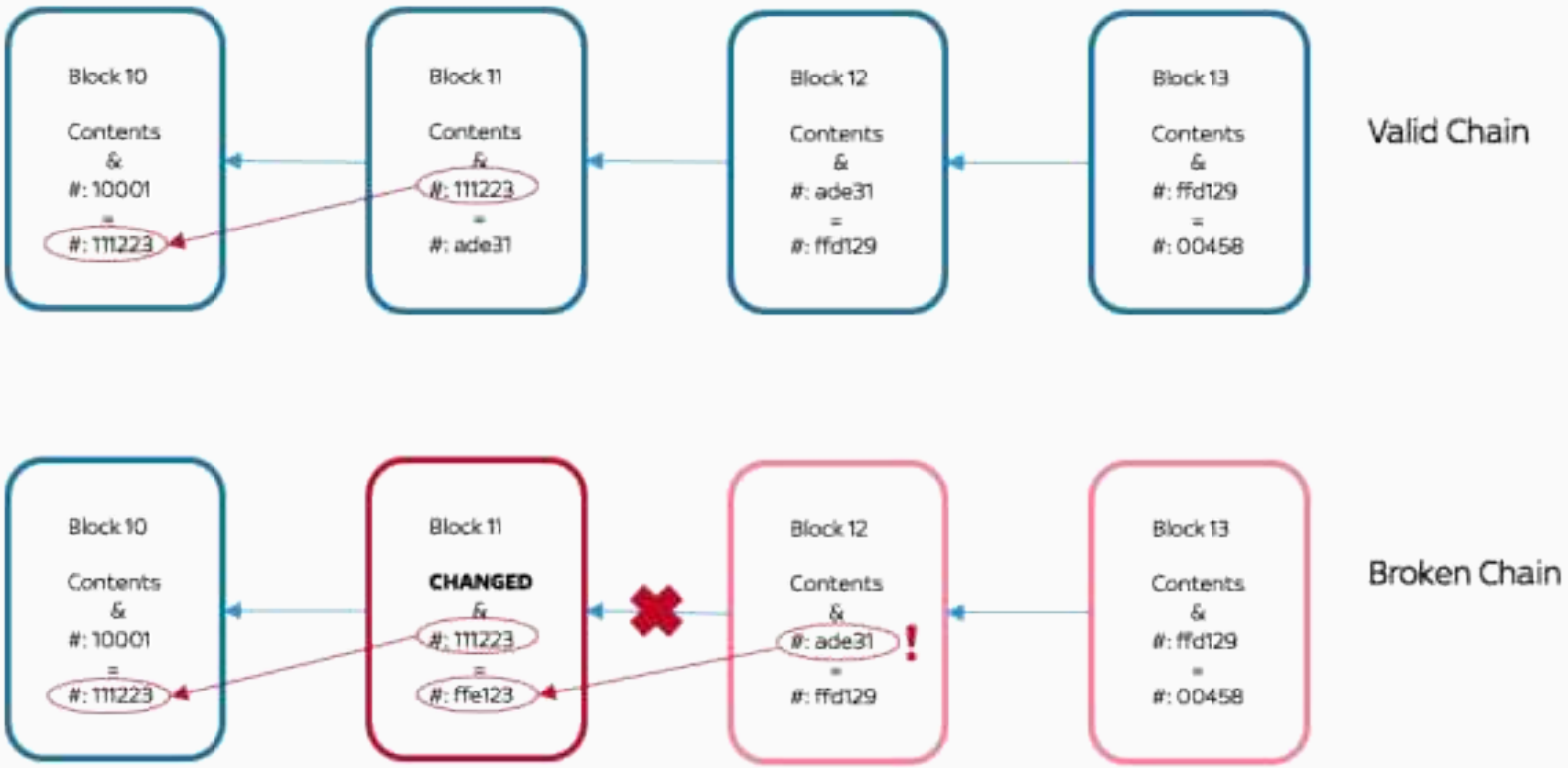


Figura 4: Bloco inválido

Comprimento *Arbitrário* $\longmapsto$ Comprimento *Fixo*

Uma função **hash** (ou de *dispersão*, ou de *espalhamento*) é uma função de *compressão com perda de informação*, isto é, um algoritmo que envia

- entradas (= sequências de bits) de tamanhos (= número de bits) **variáveis**
- a saídas (= sequências de bits) de um tamanho (= número de bits) **fixo**, usualmente entre 128 e 512 bits.

O valor de uma função *hash* é o *hash* ou *checksum* (= *soma de verificação*). Usualmente é representado na base hexadecimal cujos algarismos são 0, 1, …, 9, `A`, `B`, …, `F`. Por exemplo, o valor da função hash `SHA-256` da entrada `Oi!` começa com

`E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA4...`

Valor do Hash = Identificação

Colisões

Como o comprimento da saída é *limitada* (usualmente ≤ 512 bits) enquanto o comprimento da entrada é *ilimitada*,
$\implies$ existem **colisões** = entradas diferentes com hashes iguais!

Aleatoriedade Uniforme

Porém, o algoritmo minimiza a probabilidade de colisões pela distribuição **mais uniforme possível** dos seus valores: Por exemplo, se a saída tem 256 bits, então cada valor tem, idealmente, a mesma probabilidade 2^{-256}.

Hash = Identidade

Por isso, convém pensar de um hash de dados, por exemplo, de um arquivo, como a sua (carteira de) **identidade** (ou como CPF).

CNH

Figura 5: Identidade

Hashes Criptográficos

A função de hash é *criptográfica* (ou uma *função de embaralhamento*) se a saída praticamente independe da entrada, isto é, ela resiste (em ordem à dificuldade):

- contra a criação de **uma imagem inversa**: dada uma saída, a maneira mais rápida para achar uma entrada com esta saída é *força bruta* (isto é, provar todas as possíveis entradas);
- contra a criação de **uma segunda imagem inversa**: dada uma entrada, a maneira mais rápida para achar *outra* entrada com a mesma saída é força bruta;
- contra a criação de **colisões**: a maneira mais rápida para achar duas entradas (arbitrárias) com a mesma saída é força bruta.

Exemplos

- CRC é uma função de hash que não é criptográfica;
- MD4 (Message-Digest algorithm): Desenvolvido em 1991 por Ron Rivest, um dos três inventores da criptografia RSA; *rápido*, mas vulnerável à criação de imagens inversas.
- MD5: Desenvolvido pela RSA Data Security. Enquanto é vulnerável a colisões, mas não à criação de uma segunda imagem inversa. Muito utilizado para a verificação de integridade, por softwares com protocolo par-a-par (= P2P).
- SHA-1 (*Secure Hash Algorithm*): Desenvolvido pelo NIST, o National Institute for Standards and Technology. É vulnerável a colisões, mas não à criação de uma segunda imagem inversa.

Atualmente recomendáveis são, entre outros, SHA-256 e SHA-3.

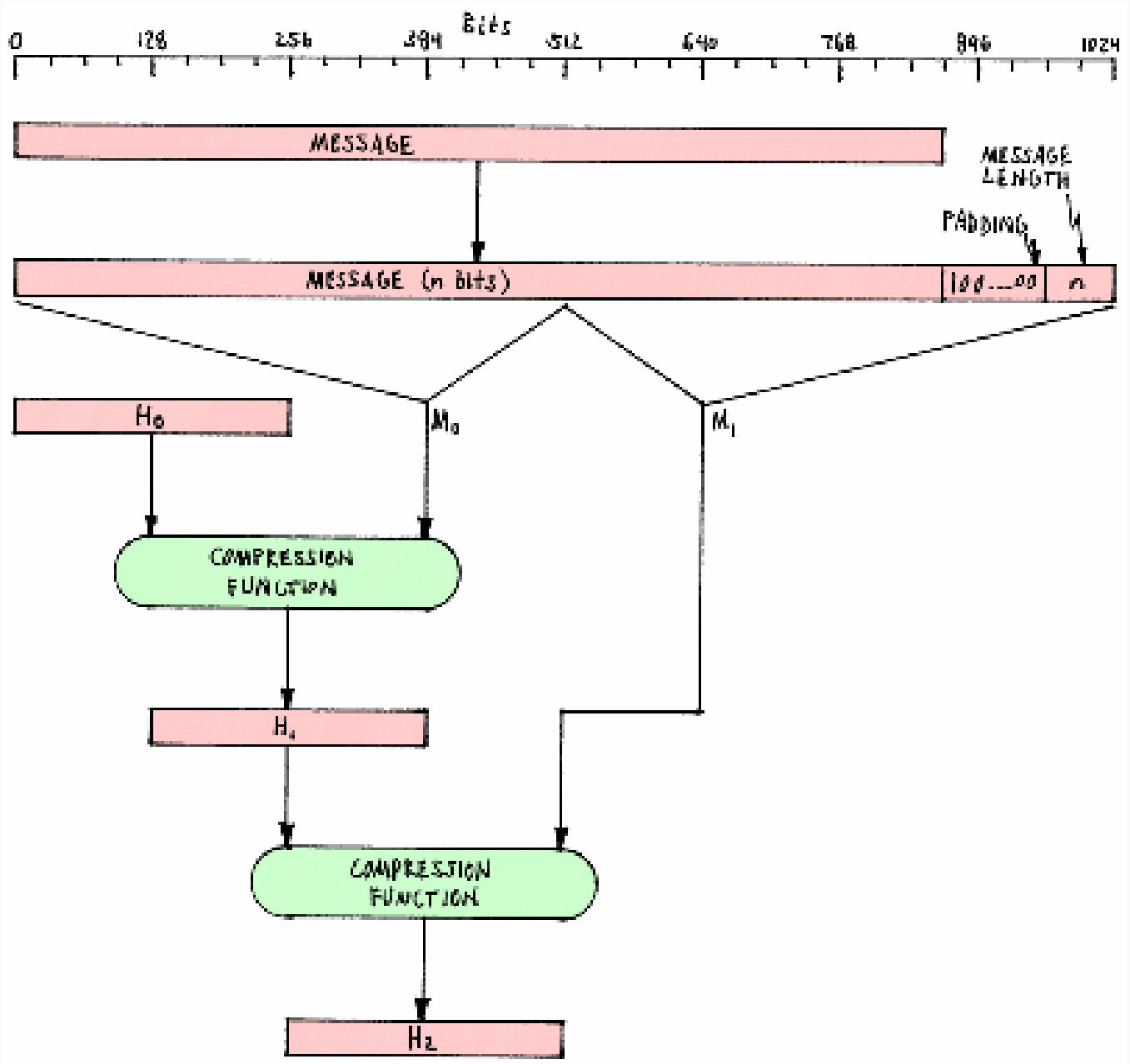


Figura 6: Visão Geral do SHA-256

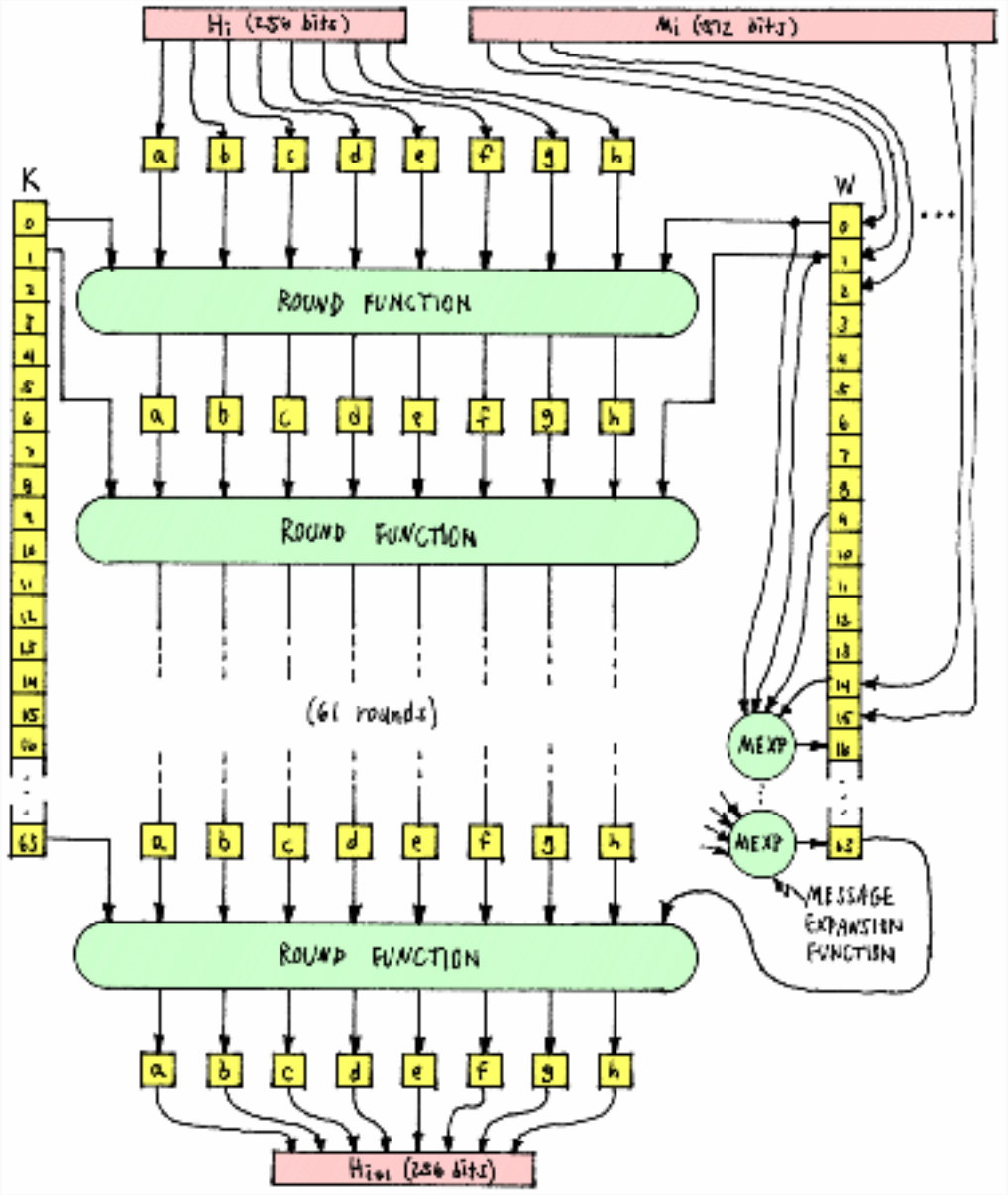


Figura 7: Compressão do SHA-256 em 64 rodadas

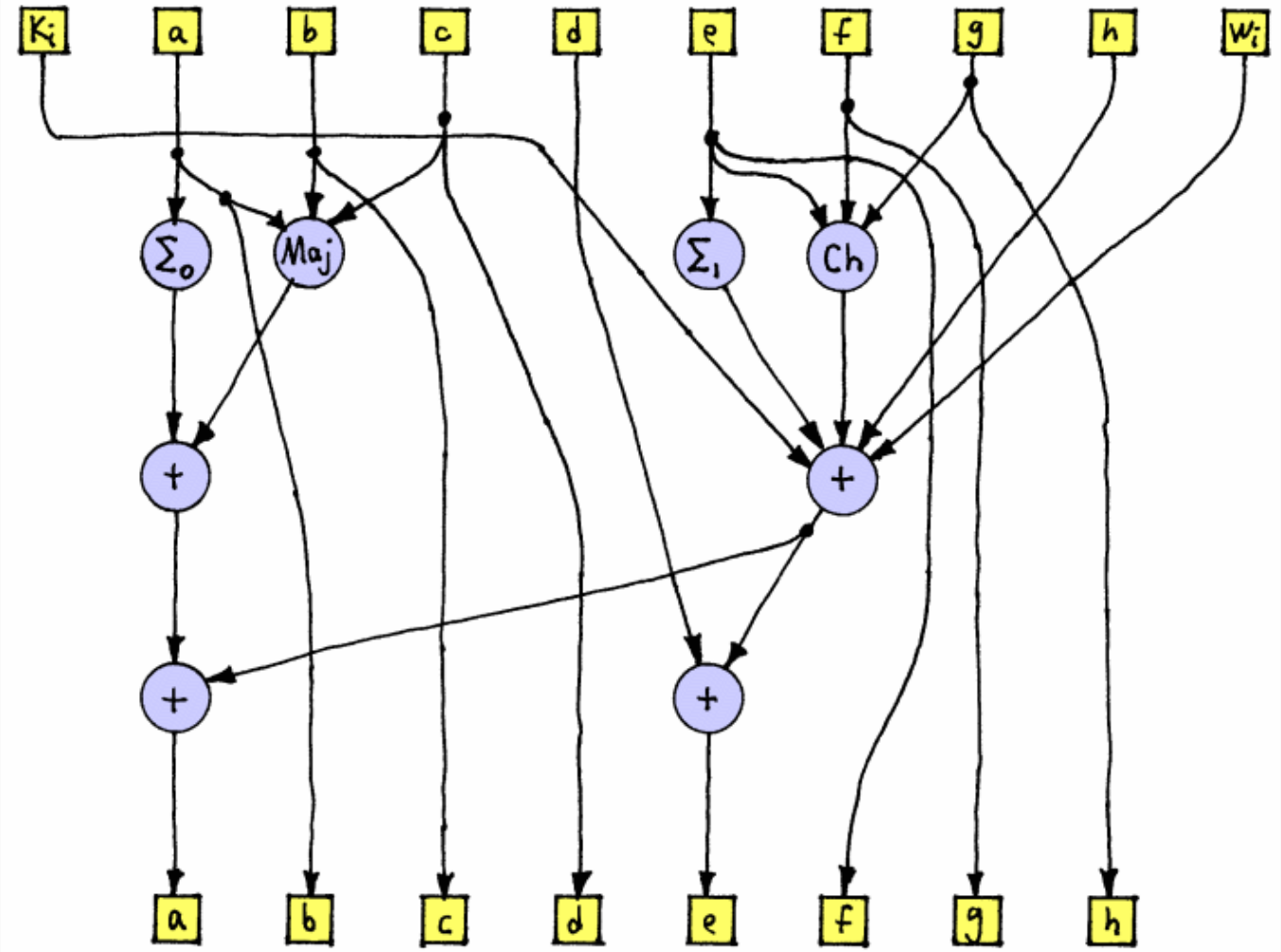


Figura 8: Rodada do SHA-256

Usos de Hashes

- Para consultas de dados rápidas (isto é, em tempo fixo, independente do número de entradas), por exemplo,
  - em uma tabela hash, e
  - em uma árvore de Merkle;
- Para garantir a *integridade* de um arquivo diante modificações *voluntárias* (e involuntárias), isto é, detetar diferenças entre o arquivo e uma versão de referência (tipicamente a antes do transporte do arquivo).
- Para distribuir valores uniformemente (`key stretching`), intuitivamente, torná-los menos previsíveis; em particular, para *gerar e armazenar senhas* (como `PBKDF = Password Based Key Derivation Function`).

Árvore de Merkle para Rápida Verificação Parcial

Uma **árvore de dispersão** ou **de Merkle** (por `Ralph Merkle`) arranja dados em blocos por uma árvore (binária), cujos *vértices* (= nós) são hashes e cujas *folhas* (= nós terminais) contêm os blocos de dados, para poder verificar rapidamente cada bloco de dados pela computação do hash da raiz.

Vantagem = Rápida Verificação Parcial

Em uma *árvore* de dispersão de profundeza n (isto é, com 2^n folhas) o bloco de dados de cada folha é verificável

- pelo **conhecimento**
  - de n hashes obtidos de uma fonte **dubitável**, e
  - do hash do topo obtido de uma fonte **confiável**, e
- pelo **cálculo**
  - do hash do bloco de dados da folha,
  - dos $n - 1$ hashes dos seus antecessores, e
  - a comparação do hash da raiz calculado com o obtido.

Principal Uso = P2P

O principal uso de árvores de Merkle é garantir que blocos de dados recebidos de outros pares em uma `rede ponto-a-ponto` (P2P) sejam inalterados (após o transporte).

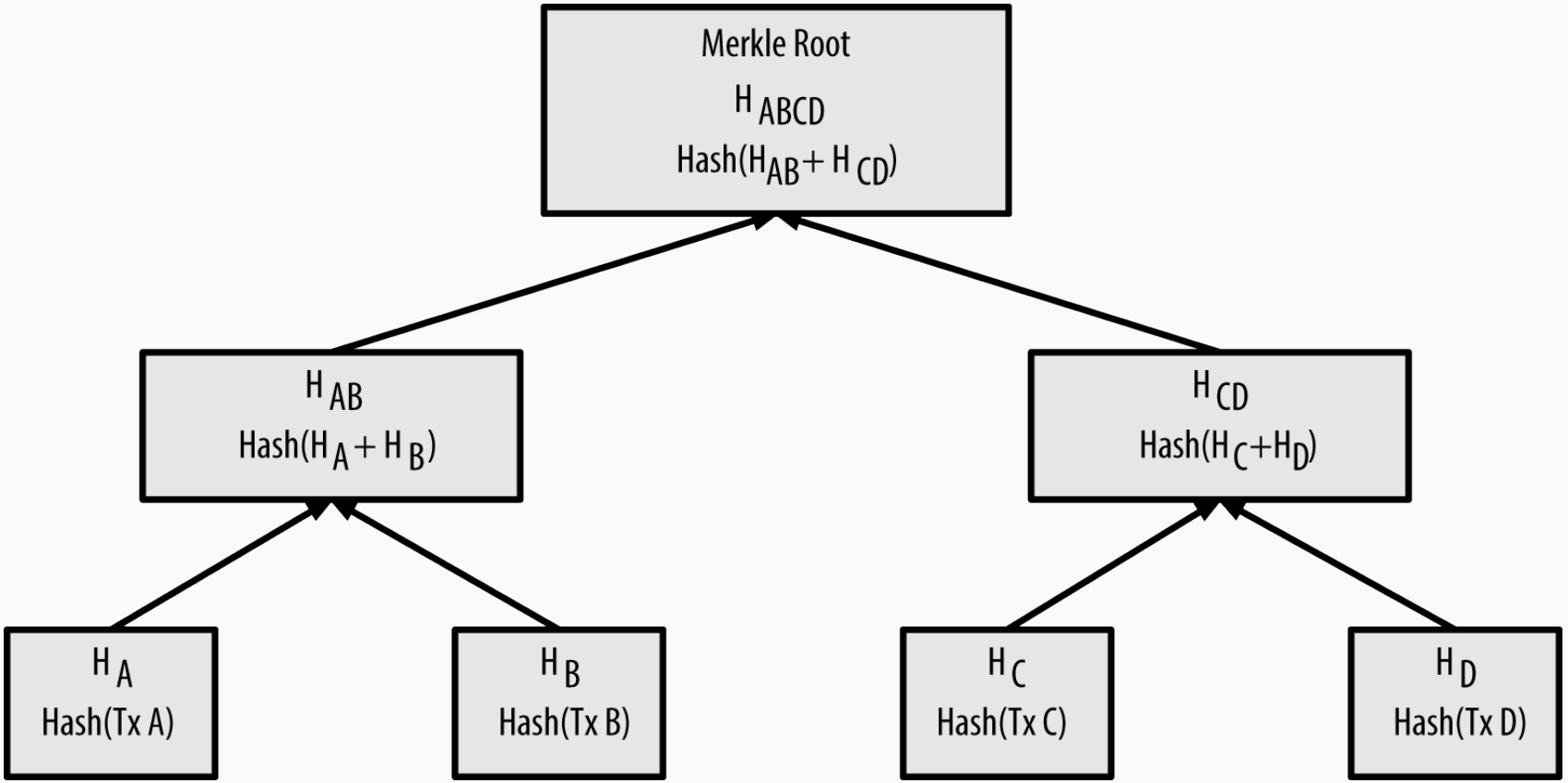


Figura 9: Árvore de Transações

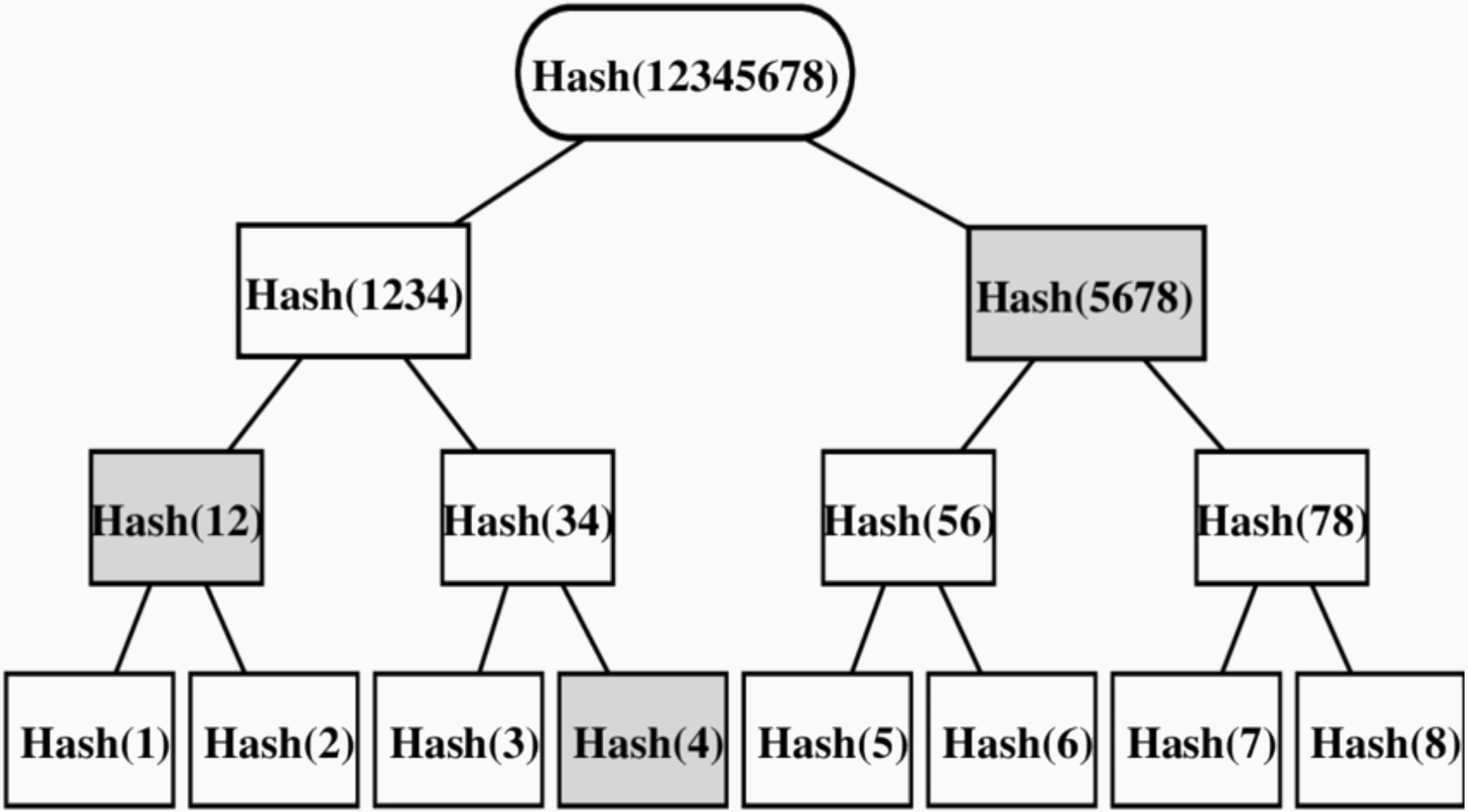


Figura 10: Verificação do terceiro bloco duma árvore de Merkle pela computação do hash radical. O hash de cada nó (mãe) é o hash (da concatenação) das duas filhas. Os hashes que necessários e que foram informados para calcular o do topo são cinzentos.

Verificação da integridade

Para verificar em Figura 10 o terceiro bloco de dados é preciso

1. obter hashes Hash(4) Hash(12), e Hash(5678), e
2. computar hashes Hash(3) Hash(34), Hash(1234), e Hash(12345678)
3. comparar o Hash(12345678), o *hash da raiz* calculado, com o obtido além.

Em uma rede P2P,

- a árvore de Merkle é recebida de qualquer ponto na rede P2P (não particularmente confiável), e
- o hash da raiz é recebida de uma fonte confiável, por exemplo, de um site reconhecido.

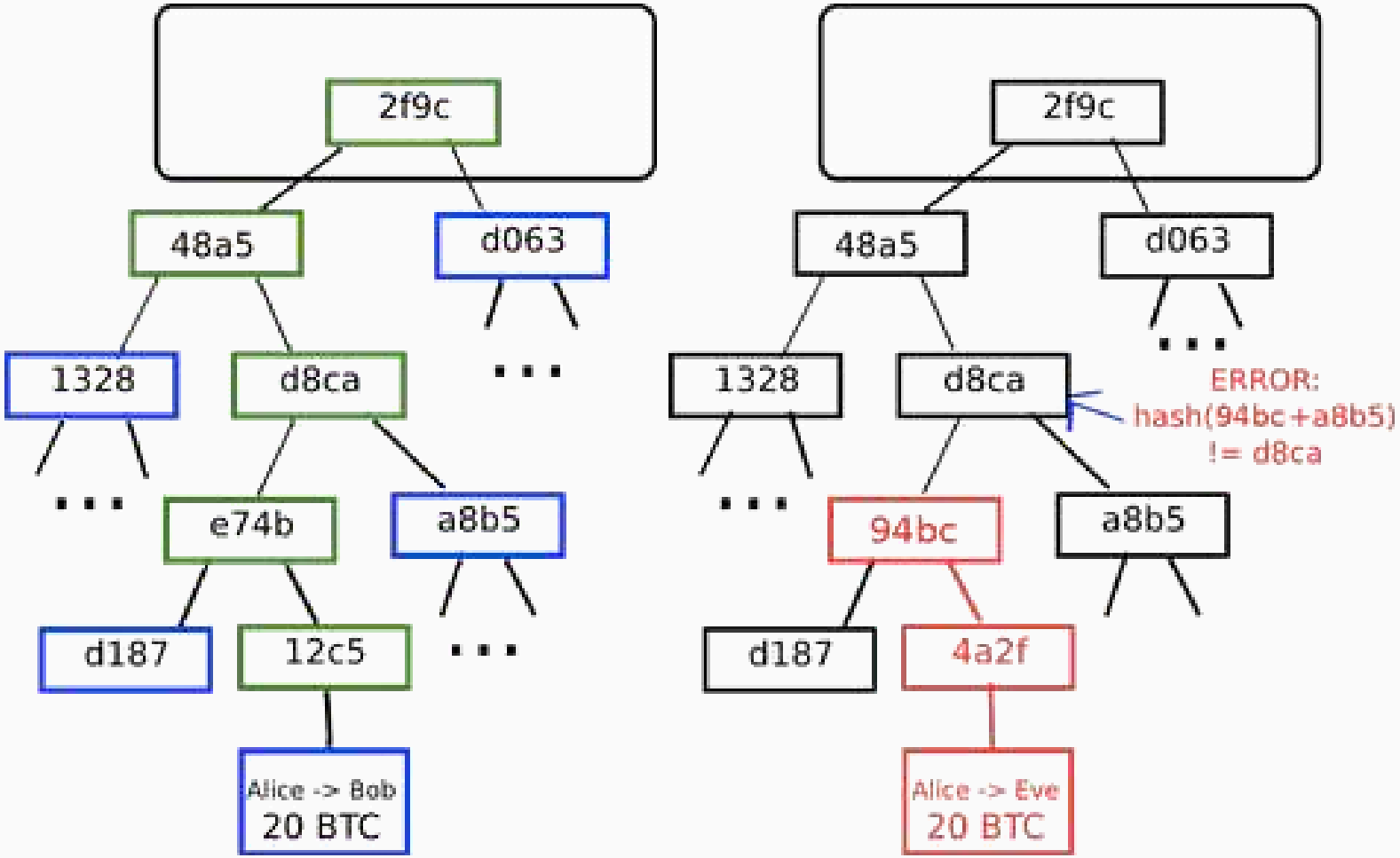


Figura 11: Árvore de Merkle com defeito

1 Cadeia

2 Laço

3 Bloco

4 Mineração

5 Transação

6 Criptografia

Estrutura do Bloco

Campo	Descrição	Tamanho
Magic	= 0xD9B4BEF9	4 bytes
Cabeçalho	Contém 6 items	80 bytes
Tamanho do Bloco		4 bytes
Número de transações		1 – 9 bytes
Transações		< 1 Megabyte

Cada bloco agrupa transações (que têm, em média, 5000 bytes).
No início, é indicado

- o seu tamanho (até um 1 Megabyte) e
- o seu número das transações (em média 2000).

Os Blocos da Cadeia

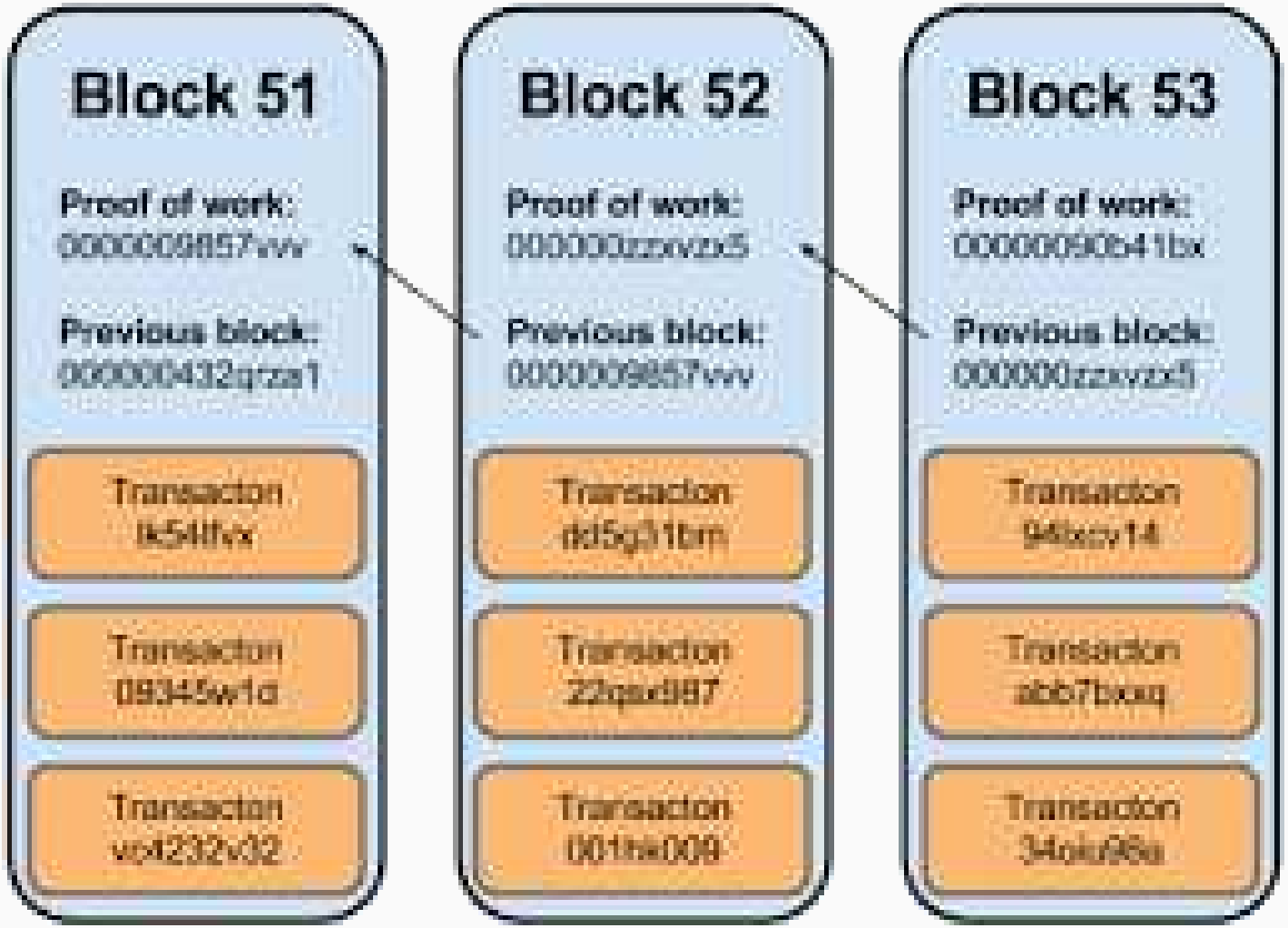


Figura 12: Blockchain

Cabeçalho

O cabeçalho de cada bloco contém:

1. a *versão* do software usado,
2. o hash do **bloco anterior** (para os blocos formarem uma cadeia),
3. o hash da **raiz** da árvore de merkle das transações,
4. o `timestamp`, a data de criação em segundos contados desde 1970-01-01 às 00:00 UTC, e
5. a **dificuldade**, grosso modo o número de zeros com que o hash do bloco precisa de começar para ele poder estender a cadeia, e
6. um `nonce`, um campo sem conteúdo (semântico); serve para alterar o hash do bloco sem alterar o conteúdo (semântico).

Transações

O “tronco” do bloco, o seu conteúdo (em contraste aos metadados do cabeçalho) são as transações:

- A primeira transação, a `coinbase` transação, a recompensa pelo trabalho feito pela sua criação, é alterável pelo criador do bloco. Logo, comumente ele e os seus colaboradores são os destinatários.
- As outras transações são transmitidas à rede pelos remetentes, isto é, pagadores, e todos os criadores de blocos, os *mineradores*, decidem quais transações incluírem no bloco que estão criando.

Para incentivar a inclusão de uma transação, o remetente pode pagar uma taxa ao criador do bloco; logo, o minerador inclui tantas transações quanto possíveis (≈ 1 Megabyte).

Para poder verificar rapidamente as transações, elas são agrupadas em uma *árvore de Merkle*, uma árvore (binária), cujos *nós* são hashes e cujas *folhas* são transações.

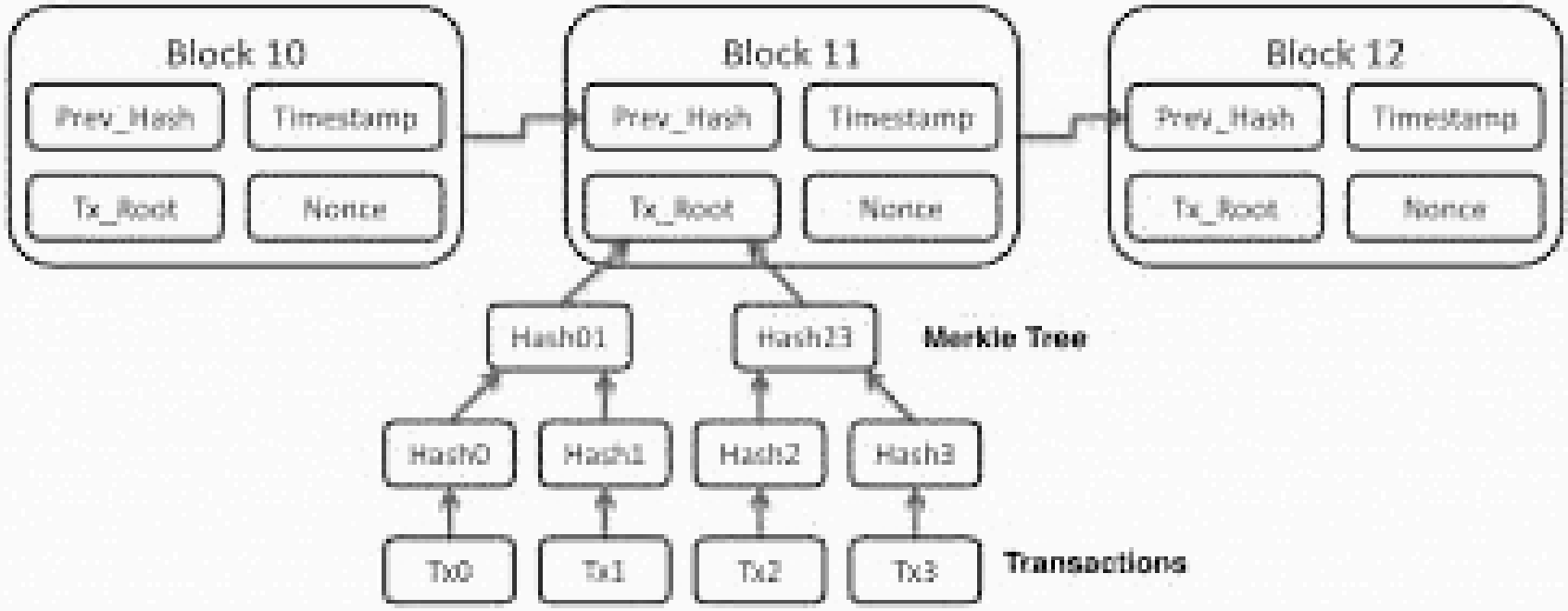


Figura 13: Transações em um bloco

1 Cadeia

2 Laço

3 Bloco

4 Mineração

5 Transação

6 Criptografia

Prova de Trabalho

Uma **prova de trabalho** é a apresentação de uma informação cuja obtenção é computacionalmente custosa.

Frequentemente, o trabalho consiste em realizar um evento pouco provável por muitas repetições; iterar e iterar até lograr! Por exemplo, encontrar uma entrada de uma função hash cuja saída comece com quatro zeros; provar até lograr:

```
"Hello, world!0" => 1312...
"Hello, world!1" => E9AF...
"Hello, world!2" => AE37...
...
"Hello, world!4248" => 6E11...
"Hello, world!4249" => C004...
"Hello, world!4250" => 0000...
```

Exemplo de uma Prova de Trabalho

Concatenemos iterativamente à sequência de caracteres

`"Hello, world!"`

um nonce, um número a uso único (= once em inglês) até que (a expansão hexadecimal d') o seu hash SHA-256 comece com `0000`!

Existem $\#\{0, 1, \ldots, 9, \mathrm{A}, \mathrm{B}, \ldots, \mathrm{F}\}^4 = 16^4 = 4096$ combinações de quatro dígitos hexadecimais; por isso, se os valores da função hash são uniformemente distribuídos,

$\Longrightarrow$ esperamos cerca de 4096 tentativas para encontrá-lo!

...

Com efeito, após 4251 tentativas, que levam um milissegundo num computador moderno, obtemos

```
"Hello, world!0" => 1312AF178C253F84028D480A6ADC1E25...
"Hello, world!1" => E9AFC424B79E4F6AB42D99C81156D3A1...
"Hello, world!2" => AE37343A357A8297591625E7134CBEA2...
...
"Hello, world!4248" => 6E110D98B388E77E9C6F042AC6B49...
"Hello, world!4249" => C004190B822F1669CAC8DC37E761C...
"Hello, world!4250" => 0000C3AF42FC31103F1FDC0151FA7...
```

Porém, no Bitcoin, o objeto hasheado, o cabeçalho do bloco, é mais complexo; em particular, porque contém a raiz da árvore (de Merkle) das transações (= um hash de todas as suas folhas).

Mineração = Buscar um bloco com hash pequeno

No Bitcoin, a prova de trabalho chama-se **mineração** e consiste em buscar um bloco cujo hash com 32 bytes pelo algoritmo SHA-256 seja menor que um certo número alvo:

- inicialmente, em 2009 um número que começa com 4 bytes que são 0,
- atualmente, em 2018, um número que começa com 10 bytes que são 0.
- O número de zeros é continuamente (cada 2016-ésimo bloco) ajustado tal que a busca leve em média (mundialmente!) 10 minutos.

Ligar os Blocos

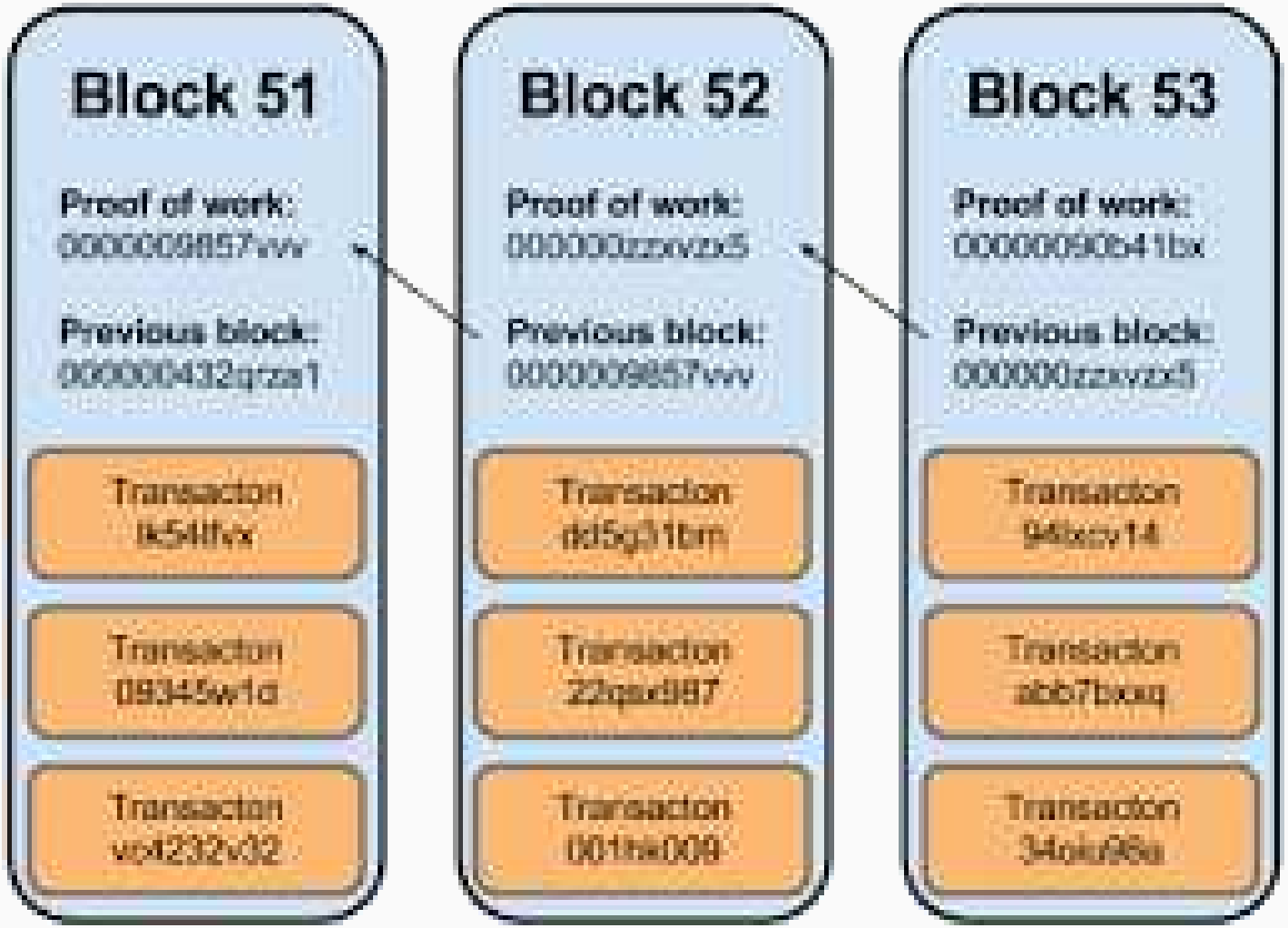


Figura 14: Blockchain

Mineração de Hashes

Para acrescentar um bloco à blockchain, é preciso dar a **prova de trabalho** do cálculo de (um *cabeçalho* de) um bloco tal que o seu hash seja pequeno, isto é, que a sua expansão binária começa com muitos zeros!

Atualmente, em agosto de 2018, com 20 zeros na expansão hexadecimal ou 80 zeros na expansão binária. Por exemplo, a expansão hexadecimal do hash de um bloco minerado no dia 31 de agosto de 2018 começa com

`00000000000000000000E17D5D11694F4602F68A8AB629093...`

Talvez o software mais usado a este fim seja atualmente CGMiner; existem versões para todos os sistemas operacionais, umas para processadores gráfico (GPUs) e outras para processadores especificamente programados para mineração (ASICs).

Irreversibilidade

O alto custo de trabalho,

- enquanto é **inútil** para **erguer** uma cadeia de blocos,
- é **essencial** para a integridade da cadeia, porque impossibilita praticamente a alteração (maléfica) de blocos anteriores; isto é, garante a **irreversibilidade** da blockchain.

Uma vez a transação é em um bloco da blockchain que foi estendido por, digamos pelo menos cinco, outros blocos, é *praticamente impossível* substituir os blocos da blockchain.

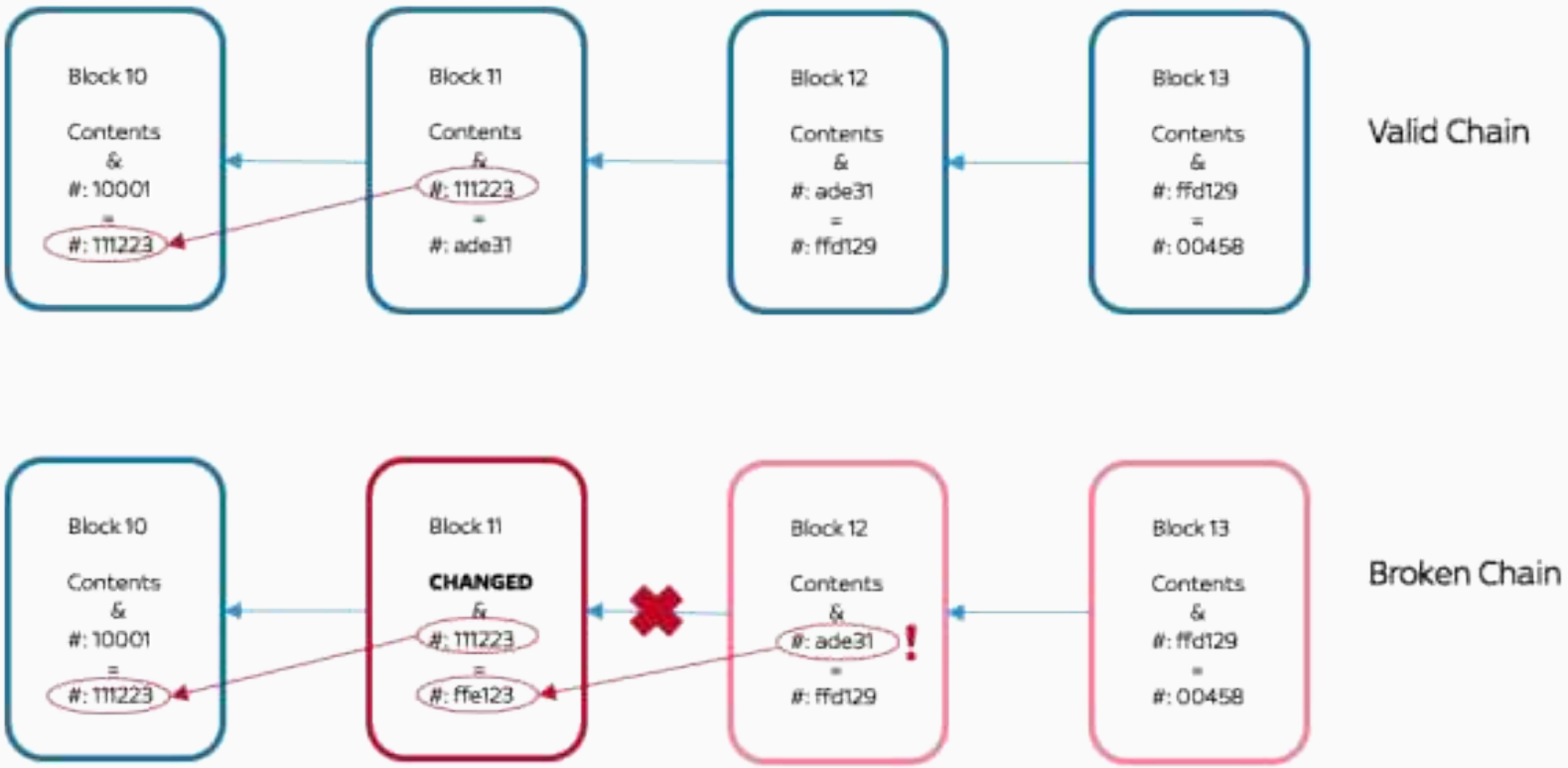


Figura 15: Bloco inválido

Sobrepujança

Como cada bloco contém o hash do bloco anterior, e o hash do bloco depende dele, a **modificação de um bloco necessita a modificação do bloco posterior**; Como cada bloco precisa de ter um hash que comece com muitos 0's para **ser aceito na blockchain**, é muito trabalhoso encontrá-los!

Para alterar um bloco na cadeia, o singelo minerador precisa de

- recalcular todos os blocos posteriores (tal que os seus hashes comecem com um número de 0's suficiente),
- enquanto os outros mineradores concatenam outros blocos!

Este imenso trabalho é uma **corrida contra o tempo e a força computacional mundial**!

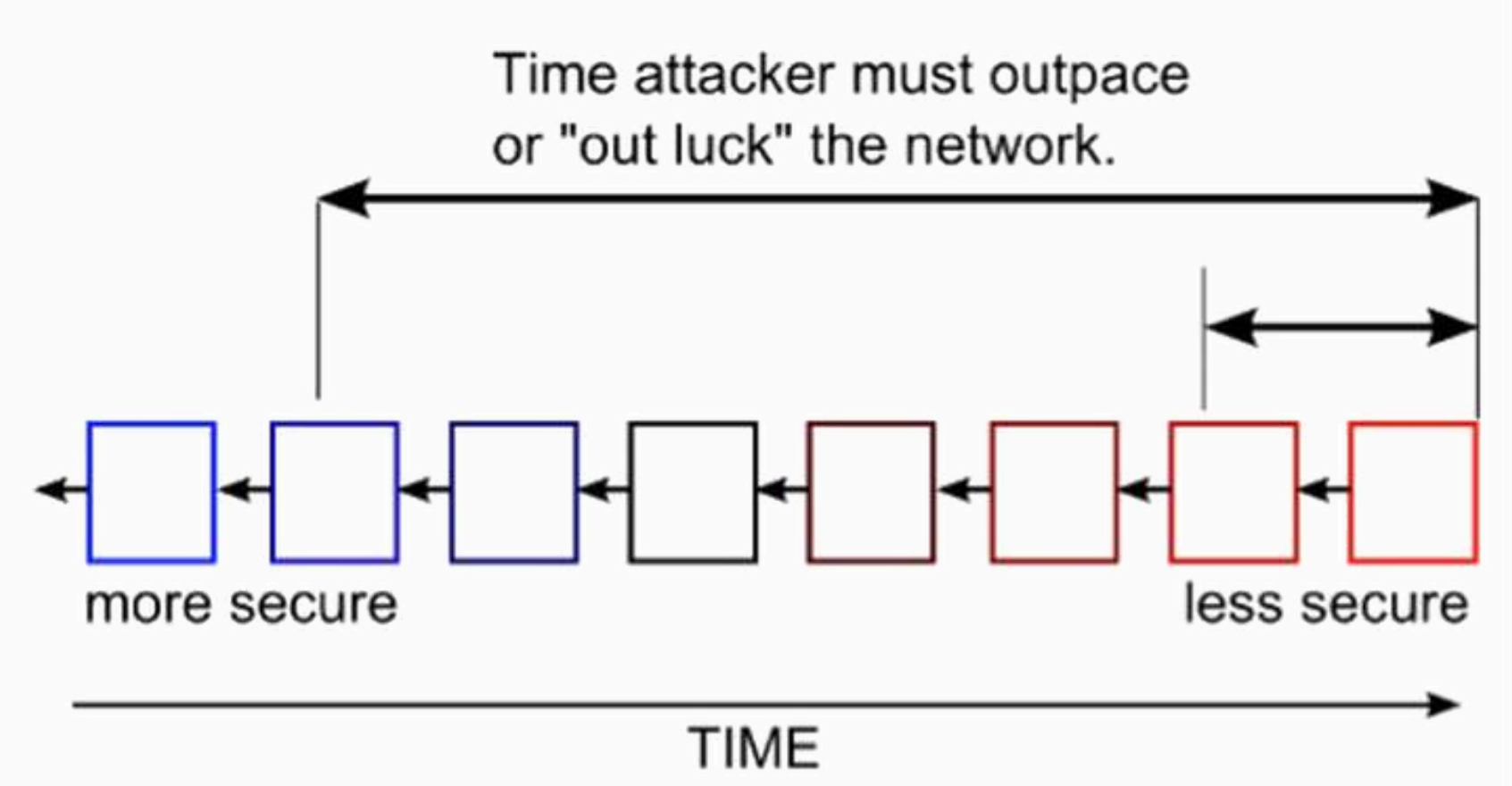


Figura 16: Sobrepujança

Ataque de 51%

Se um minerador tem mais força computacional do que todos os outros mineradores, então poderá enganar o negociante assim:

1. Remete uma transação,
2. espera a sua confirmação, isto é, espera a sua inclusão em um bloco da blockchain,
3. no momento em que é inclusa em um bloco, bifurca a blockchain pela prolongação do bloco precedente por outro bloco que não inclua esta transação,
4. continua a prolongar este ramo pela criação de outros blocos,
5. quando a transação é confirmada, isto é, em média depois 6 blocos, transmite à rede os blocos que criou. Pela sua suposta maior força computacional, este ramo é mais comprido do que a blockchain; logo, os nós aceitam o novo ramo como nova blockchain válida.

Ajusto da Dificuldade:

O campo `dificuldade` no cabeçalho do bloco compara

- o esforço computacional que *está* (isto é, atualmente) em média necessário para encontrar um **novo** bloco da blockchain

com

- o esforço computacional que *era* em média necessário para encontrar o **primeiro** bloco (o bloco 'Génesis') da blockchain:

No início, para o primeiro bloco, o bloco Génesis, ser aceite na cadeia, a expansão binária do hash do seu cabeçalho precisou de começar com 32 zeros (= o tamanho, em bits, do `nonce` no cabeçalho do bloco).

Cada reajuste (após 2016 blocos) calcula a nova dificuldade como proporção entre

- o tempo **alvo** $\mathrm{A} = 2016 \cdot 10$ minutos (cerca de duas semanas), e
- o tempo U em minutos que a criação dos **últimos** 2016 blocos levou;

isto é,

$$\text{nova dificuldade} = \text{última dificuldade} \cdot \mathrm{A}/\mathrm{U}$$

Para abrandar saltos, a nova dificuldade é no máximo 4, isto é, mesmo se $\mathrm{A}/\mathrm{U} > 4$ (quer dizer, a rede levou menos de três dias e meio), então a nova dificuldade é 4.

Tempo de Mineração

1. No início da blockchain, $n = 32$, e eram necessárias em média 2^{32} (≈ 4 bilhões) computações de hashes para encontrar um bloco com um tal hash.

   O `Intel Core i7 2600` consegue $\approx 2,4 \cdot 10^7$ (= 24 milhões) hashes por segundo. Por isso, ele levou em média ≈ 15 minutos .

2. Atualmente, $n \geq 80$ e são necessárias em média mais de $2^{80} \approx 1,2 \cdot 10^{24}$ (um septilhão (= um trilhão vezes um trilhão)) de hashes para o bloco ser aceite na cadeia.

   Por isso, o `Intel Core i7 2600` leva em média $\approx 5 \cdot 10^{16}$ segundos ($> 10^9$ = um bilhão de anos) até ele encontrar um bloco que será aceite na cadeia.

Isto passa uma ideia da juntada força computacional dos mineradores que calculam o hash em média em dez minutos!

Recompensa

Todos os bitcoins são gerados por mineração e transferidos aos mineradores, aos criadores do bloco, pela primeira transação de cada bloco, a `coinbase` que é destinada aos criadores.

Ao descobrir um novo bloco, o minerador transmite-o à rede e os nós verificam, entre outros,

- que o **hash do cabeçalho** comece de fato com o número de 0s exigidos pela dificuldade atual, e
- que todas as **transações sejam válidas**.

Após este bloco ser estendido por pelo menos 100 blocos (e assim a rede ter certeza que ele permaneça na blockchain, a cadeia mais comprida), o descobridor:

- Ganha uma certa quantia de bitcoins como recompensa para o trabalho da descoberta: inicialmente, em 2009 foram 50, atualmente, em 2018, são $12,5$; o valor é dividida pela metade após 210000 blocos (como um bloco é minerado em média de 10 em 10 minutos, isto leva cerca de 4 anos).
- Ganha todas as taxas de transação inclusas no bloco descoberto; um incentivo para incluir a transação no bloco.

Como a recompensa diminui gradualmente (até se esvair após $21000000 = (50 + 25 + 12,5 + \cdots) \cdot 210000$ blocos), as taxas terão um papel mais e mais importante para incentivar os mineradores. Em 2018, já $> 80\%$ dos bitcoins foram minerados.

1 Cadeia

2 Laço

3 Bloco

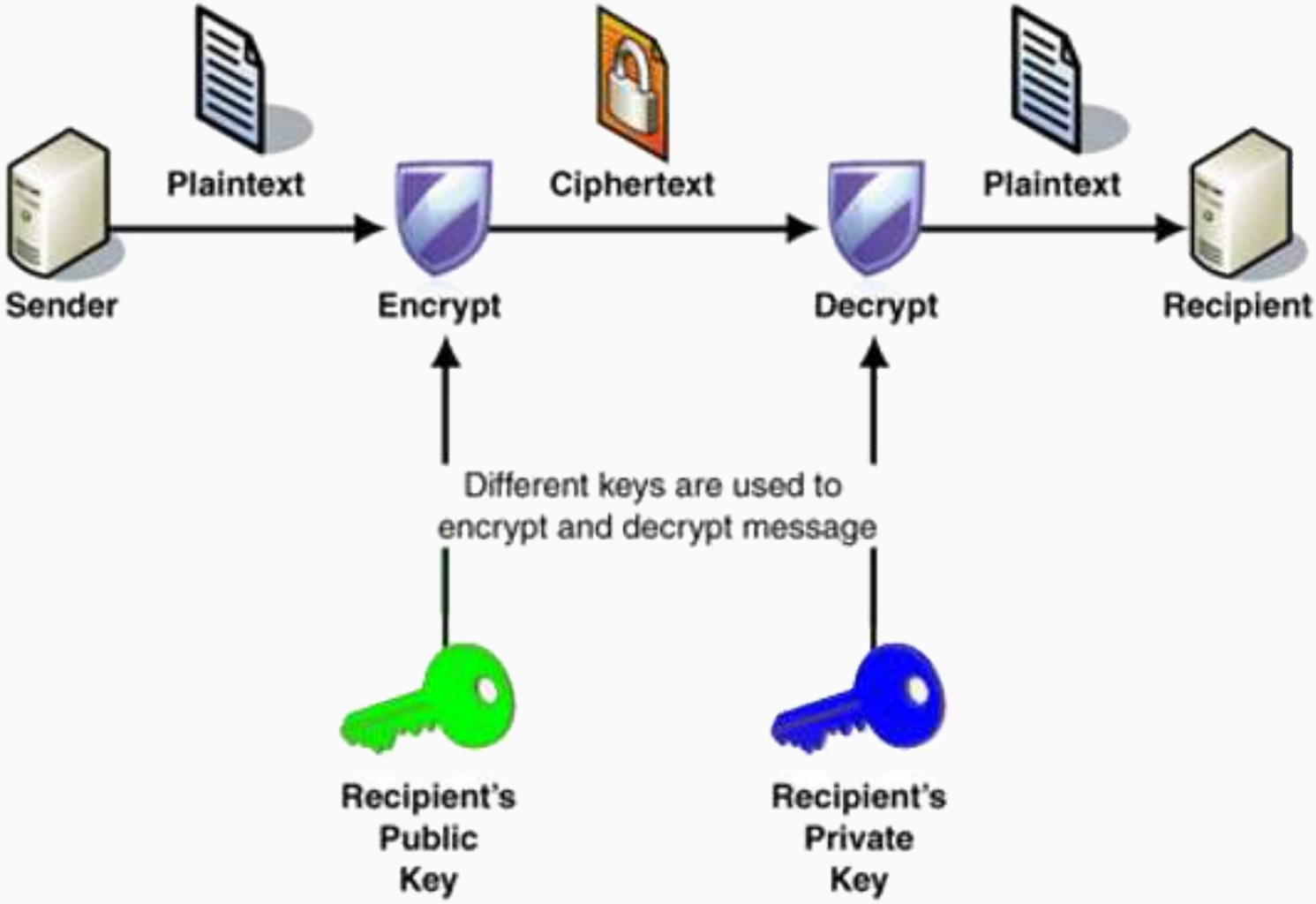
4 Mineração

5 Transação

6 Criptografia

A criptografia estuda o **embaralhamento** de dados (cifração) tal que só uma informação adicional secreta, a **chave**, permite **desfazê**-lo (decifração). Ela é **assimétrica** se

chave para cifrar ≠ chave para decifrar.



Chave pública e privada

Há duas chaves, uma chave **pública** e outra **privada**. Dois usos:

- (**Cifração**) A chave **pública** usa-se para **cifrar**, e a chave *privada* para *decifrar*,

  $\Longrightarrow$ pode ser **transferido** por qualquer cifrador um texto ao decifrador e *a ele só*.

- (**Assinatura Digital**) A chave **privada** usa-se para **cifrar**, e a chave *pública* para *decifrar*,

  $\Longrightarrow$ pode ser **verificado** por qualquer decifrador se um texto origina do cifrador e *dele só*.

Chave Pública codificada pelo alfabeto ASCII

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: SKS 1.1.6
Comment: Hostname: pgp.mit.edu

mQENBFcFAs8BCACrW3TP/ZiMRQJqWP0SEzXqm2cBZ+fyBUrvcu1fGU890
pd43J diW IreHx/sbJdW1wjABeW8xS1bM67nLW9VVHUPLi9QP3VGfmqm
XqbWIB7OxizZ PTDCWm oymm/+TlTTAZWU6Wwvmjk88QlmU941tUvBsQ1
cw1cAxw+2jLCgkz8XvW npMPKKj1f sNZ/FcPVMC6dkwHAFc7Rm4DNibJ
zLvD8woL0vAdUR4HhOQli9+Fpv UO0KVVhOwF0f 14EURddA4qZVyPM8e
3FvxiWF0JWJuxCuiBHh5ghT/Q+OQMMOJW VTwME5nQ87vohfX gjRbYjW
8gyLRqIjt2Gc7dNgIoKE5r/PABEBAAG0I0Vubm8g TmFnZWwgPGVubm8u
bm FnZWxAdC1vbmxpbmUuZGU+iQE5BBMBAgAjBQJXBQLPAh sDBwsJCAc
DAgEGFQgCCQo LBBYCAwECHgECF4AACgkQVJDsz4ujnoR03gf9HIIc pO
...

-----END PGP PUBLIC KEY BLOCK-----
```

Transferência

Todos os bitcoins são gerados por mineração. Uma vez gerado, um(a moeda de) `bitcoin` muda de lugar por uma cadeia de assinaturas digitais: O dono transfere o seu `bitcoin` ao próximo

- pela sua assinatura (com a sua chave privada) de (hashes das) transferências anteriores destinadas à (uma codificação de um hash da) sua chave pública, e
- pela indicação
  - da quantia e
  - de (um hash da) chave pública do próximo destinatário.

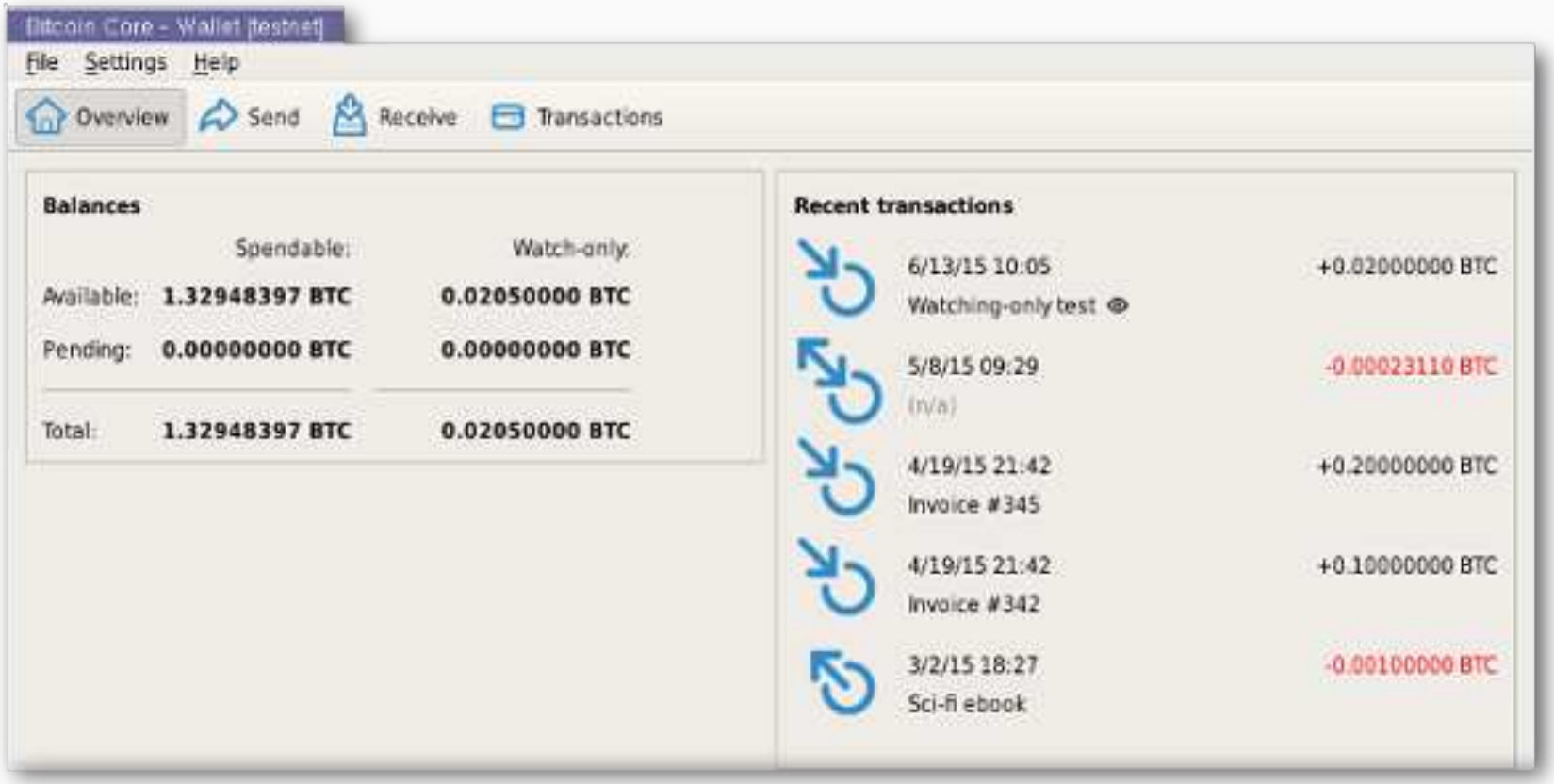


Figura 17: A carteira no `Bitcoin`

A soma constada na carteira da interface gráfica `Bitcoin Core` do bitcoin (desenvolvida inicialmente por Satoshi Nakamoto sob o nome `Bitcoin-Qt` e usado por 90% dos negociantes) é a soma de todas as transações que o dono da carteira recebeu.

Transação de Geração

A *transação de geração* é determinada pelo minerador:

- como única informação no input um campo `coinbase` de 100 bytes cujo conteúdo é arbitrário.
- o `output` distribui a recompensa (de $12,5$ bitcoins em 2018) aos favorecidos pelo minerador.

Transação de negociação

A *transação de negociação* tem

- um `input` que refere a saídas de transações em que o remetente recebeu bitcoins, e
- um `output` que transfere aos destinatários uma quantia **igual** a da soma das saídas das transações da entrada.
  - Frequentemente, a transação inclui um **troco**, isto é, o remetente figura como um dos destinatários.
  - Se a soma é menor, a diferença é paga ao criador do bloco como **taxa** para incentivá-lo a incluir a transação.

Uma vez incluída em um bloco na cadeia, e este bloco ser estendido por um número suficiente (≥ 6) de outros, a transação pode ser considerada irreversível, **confirmada**.

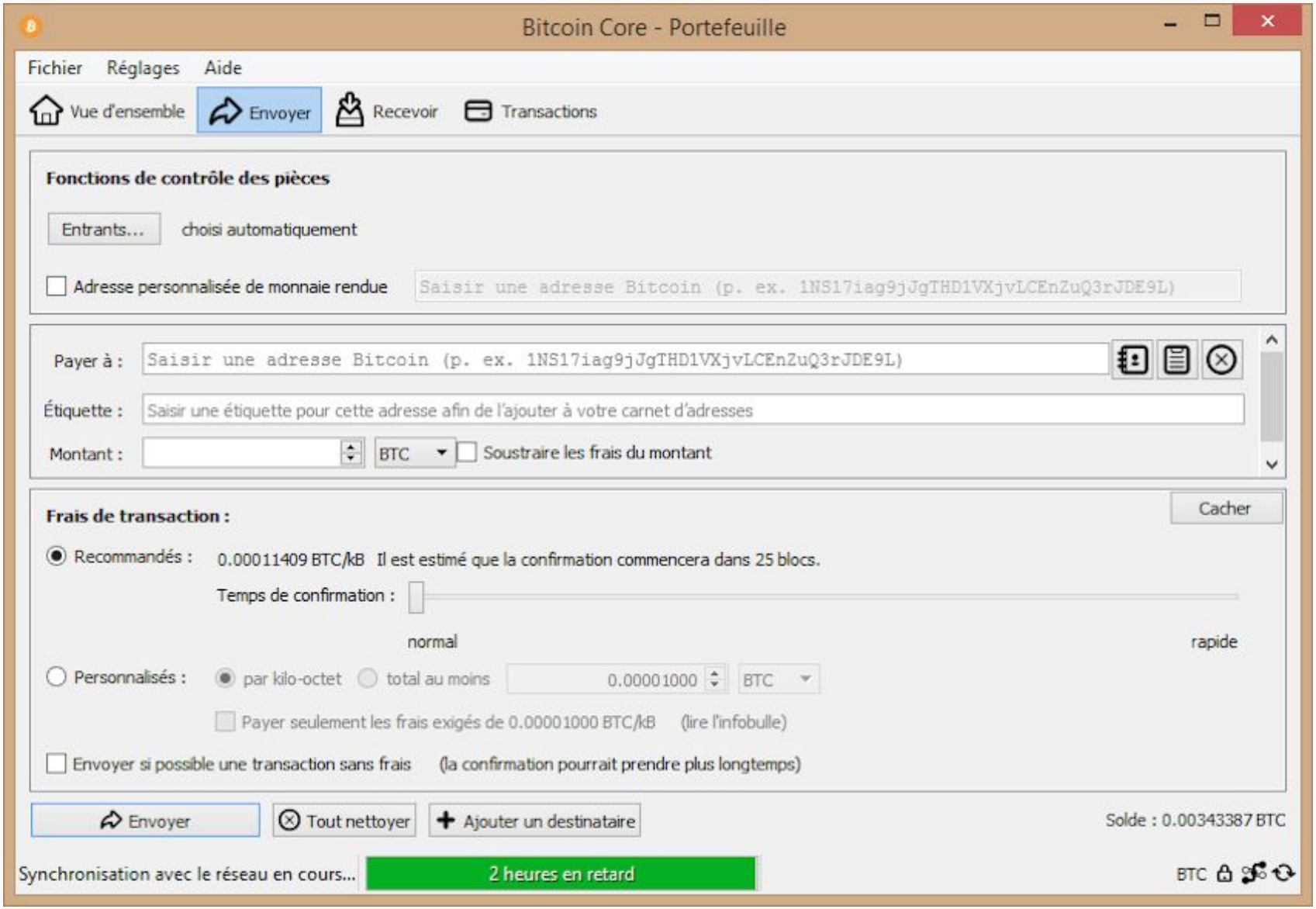


Figura 18: Transação no aplicativo `Bitcoin Core`. O pagamento de uma taxa de transação ajuda a diminuir o tempo de confirmação.

Esgotamento

Todo output pode ser usado **uma única vez só** em uma transação posterior; isto é, para evitar perda, a soma **inteira** do input precisa ser enviada! Por exemplo, se a soma da entrada é 5 BTC, mas o remetente quer somente transferir 1 BTC ao destinatário, então ele cria dois ouputs,

- um de 1 BTC para o destinatário, e
- outro de 4 BTC para ele, o remetente (= o **troco**).

De novo, toda a quantia que não foi gastada será considerada uma *taxa de transação*, isto é, transferida ao criador do bloco (para estimulá-lo a incluir esta transação antes de outras).

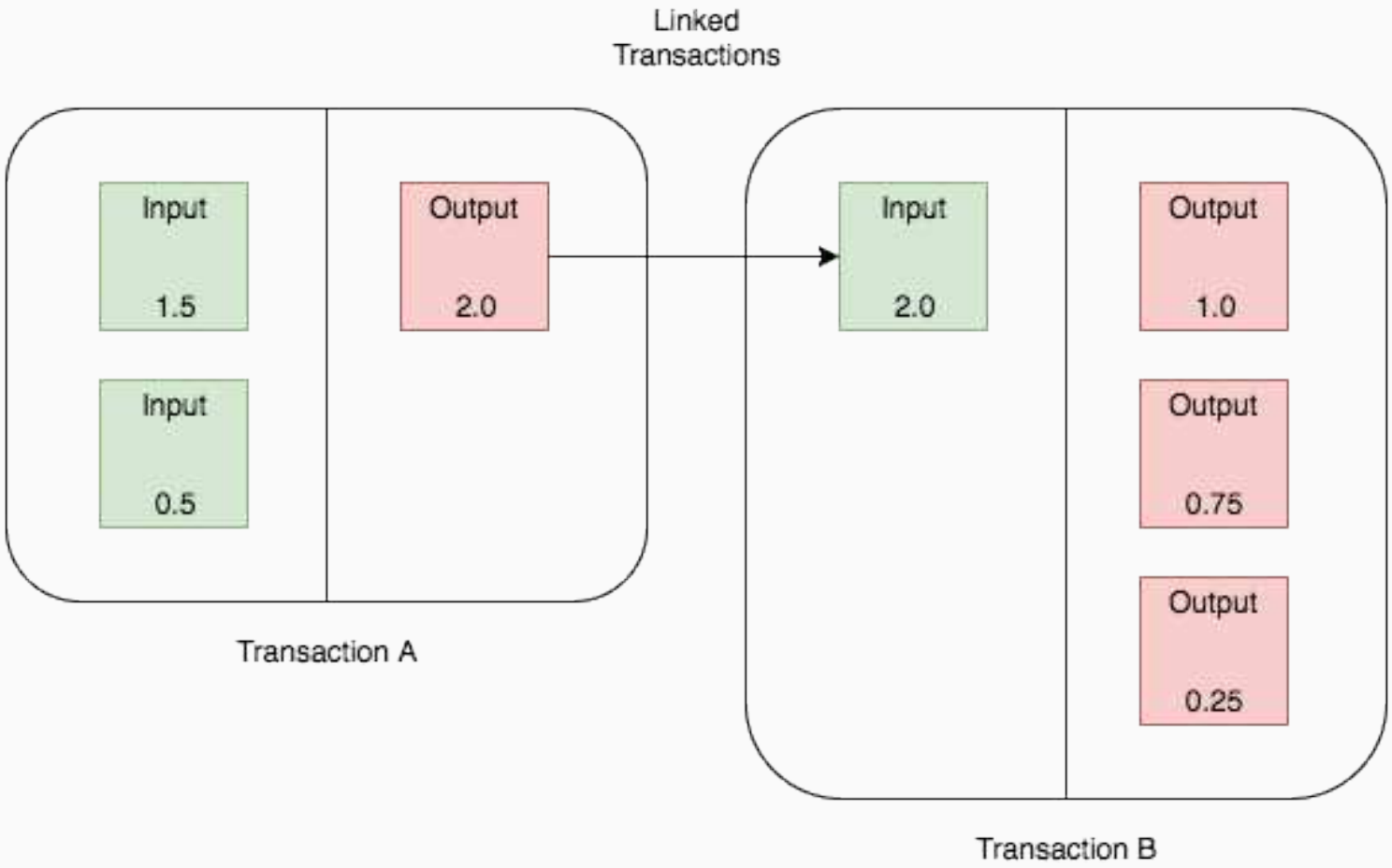


Figura 19: Transações ligadas com várias entradas e saídas

Alice, para mandar 4 Bitcoins ao Bob:

1. No input,
   1.1 escolhe transações que recebeu cuja soma é $\geq$ a quantia que mandará ao Bob, por exemplo, de $2 + 3 \geq 4$, e
   1.2 refere aos seus hashes e, para cada transação, o índice no output que refere a ela como recipiente.
2. No output, indica
   2.1 a quantia 400000000 (em Satoshis) que mandará ao Bob,
   2.2 o endereço do Bob, um hash da sua chave pública.
   2.3 além da quantia do troco e do próprio endereço
3. Finalmente, no input,
   3.1 cria um hash
      - dos hashes e índices das transações do input,
      - dos endereços nos outputs dos quais recebeu do input,
      - do endereço do Bob e da quantia a ser transferida;
      - do próprio endereço e da quantia do troco. (Toda sobra será dada ao minerador.)
   3.2 assina este hash pela sua chave privada.

Fluxo de um Bitcoin

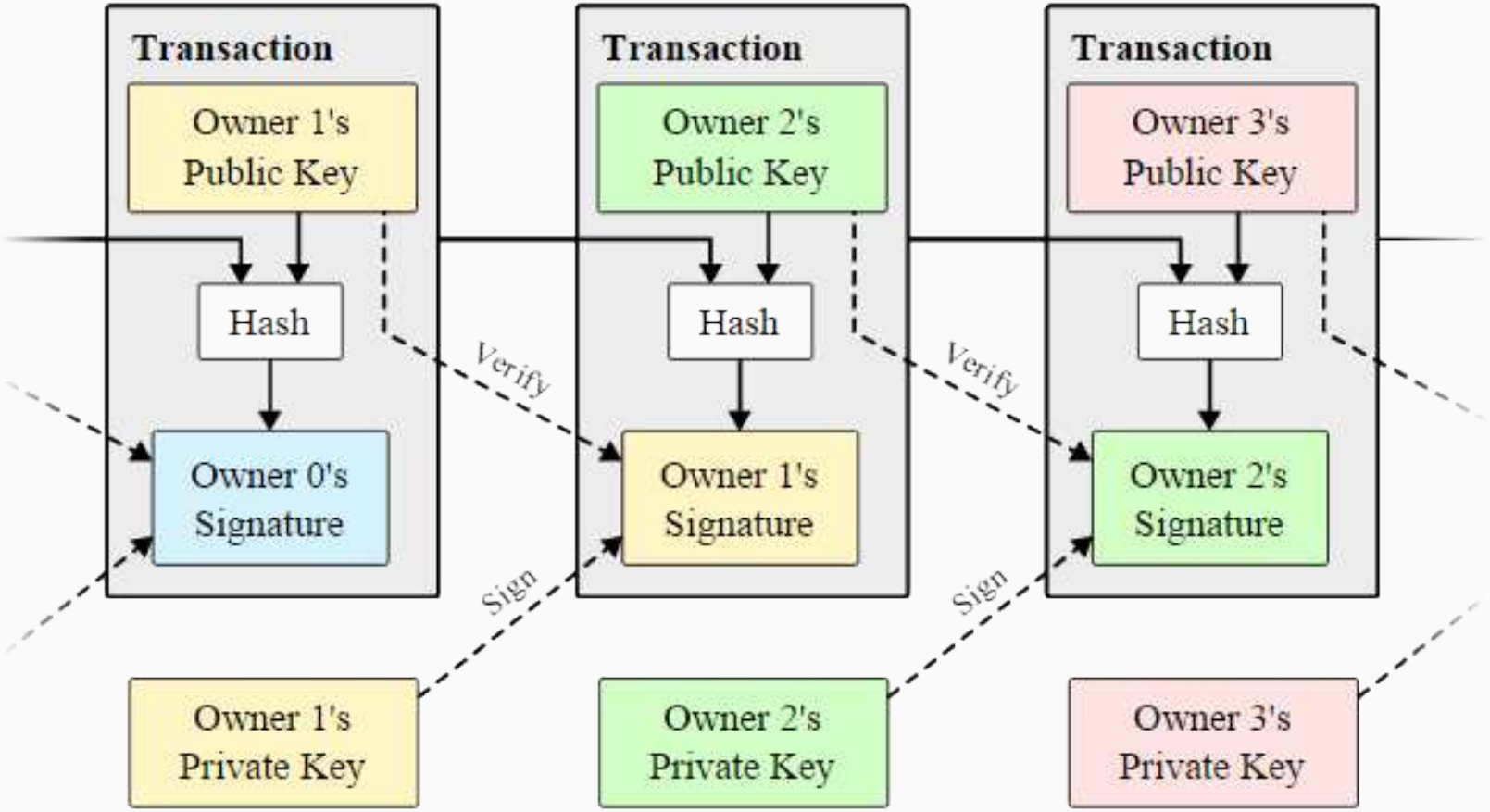


Figura 20: No hash assinado pelo remetente (= o antigo dono) entram o endereço (= chave pública) do destinatário (= o novo dono) e a transação recebida e que será gastada; em particular, nela figura a chave pública do remetente.

Endereço de Bitcoin

A identidade de cada negociante corresponde à sua chave assimétrica (ECDSA), um par de

- uma chave *pública* com 32 bytes, e
- uma chave *privada*.

Para obter o seu *endereço*, várias funções de hash são aplicadas à chave pública; a sequência final de 25 letras é codificada pela Base58 cujos 58 algarismos são

- todos os números,
- todas as letras minúsculas, e
- todas as letras maiúsculas
- exceto o número e a letra `0O` e as letras `Il` pelo risco da sua confusão.

Processamento

1. O usuário envia uma transação pelo seu aplicativo de carteira.
2. A transação é difusa pelos nós e faz parte do ‘acervo de transações não-confirmadas’.
3. Mineradores,
    3.1 escolhem transações deste acervo (de preferência, as que lhe pagam a maior taxa de transação, a diferença entre a entrada e saída),
    3.2 verificam-nas (por exemplo, se o saldo do pagador é suficiente), e
    3.3 acrescentam-nas ao bloco que tentam gerar;

    normalmente até esgotar o tamanho máximo (= 1 Megabyte) do bloco .

4. Cada minerador tenta de modificar o bloco de tal maneira que o seu hash seja suficientemente pequeno (por exemplo, atualmente, que comece com 10 bytes nulos). A probabilidade de encontrar um tal bloco, isto é, a dificuldade do problema, é a mesma para todos os blocos e mineradores. Esta dificuldade é ajustada cada 2016-ésimo bloco (isto é, após cerca duas semanas) dependendo da força computacional total dos mineradores para um tal bloco sempre ser encontrado em média em dez minutos.

   Quando um novo bloco é concatenado à cadeia, todos os mineradores têm de recomeçar com outro bloco, já que, por exemplo,

   - umas transações foram feitas,
   - o indicador (ao hash do bloco anteiror) mudou.

5. Todo minerador que encontra um bloco válido transmite-o aos nós da rede.

6. O nó verifica se o bloco seja válido, isto é,
    - que todas as transações sejam válidas, e
    - que o hash seja suficientemente pequeno.
7. Se o nó acorda sobre a validade do bloco, então o concatenam à cadeia. Uma confirmação de uma transação é cada extensão da cadeia por um bloco após aquele que contém a transação.

    A extensão da cadeia pelo bloco que contém a transação conta como primeira confirmação, a após este bloco como segunda confirmação, e assim por diante.

    A chance da cadeia mudar após seis confirmações (isto é, após, em média, uma hora) é minúscula; por isso, comumente, uma transação é vista como conclua após a sexta confirmação.

Verificar que transação ainda não foi gastada

Para o nó verificar *rapidamente* a validade da entrada de uma transação, isto é, se as transações redimidas não foram gastadas ainda, ele atualiza a cada momento o banco-de-dados *Unspent Transaction Ouput* (UTXO) de todas as transações que ainda não foram gastadas. Hoje, em 2018, o UTXO tem cerca de 5 GB e é guardada na memória para garantir consultas rápidas.

Recordemo-nos de que cada transação é única, e pode ser gastada unicamente uma vez, e inteiramente (para obter o troco, cria se outra transação). Quando uma transação é transmitida,

- as transações anteriores que figuram no input são removidas do UTXO, e
- as novas transações do output são adicionadas ao UTXO.

Confirmações para concluir uma Negociação

Uma *confirmação* de uma transação é cada extensão da cadeia por um bloco após aquele que contém a transação.

1. A extensão da cadeia pelo bloco que contém a transação conta como *primeira* confirmação,
2. a após este bloco como *segunda* confirmação, e
3. assim por diante.

A chance da cadeia mudar após 6 confirmações (que leva, em média, uma hora) é minúscula; por isso, comumente, uma transação é considerada conclusa após a 6.confirmação (na interface gráfica padrão para o bitcoin, o `Bitcoin Core`). Porém, o número 6 é arbitrário.

Em contraste, `bitcoins` *minerados* só podem ser gastados após 100 confirmações.

1 Cadeia

2 Laço

3 Bloco

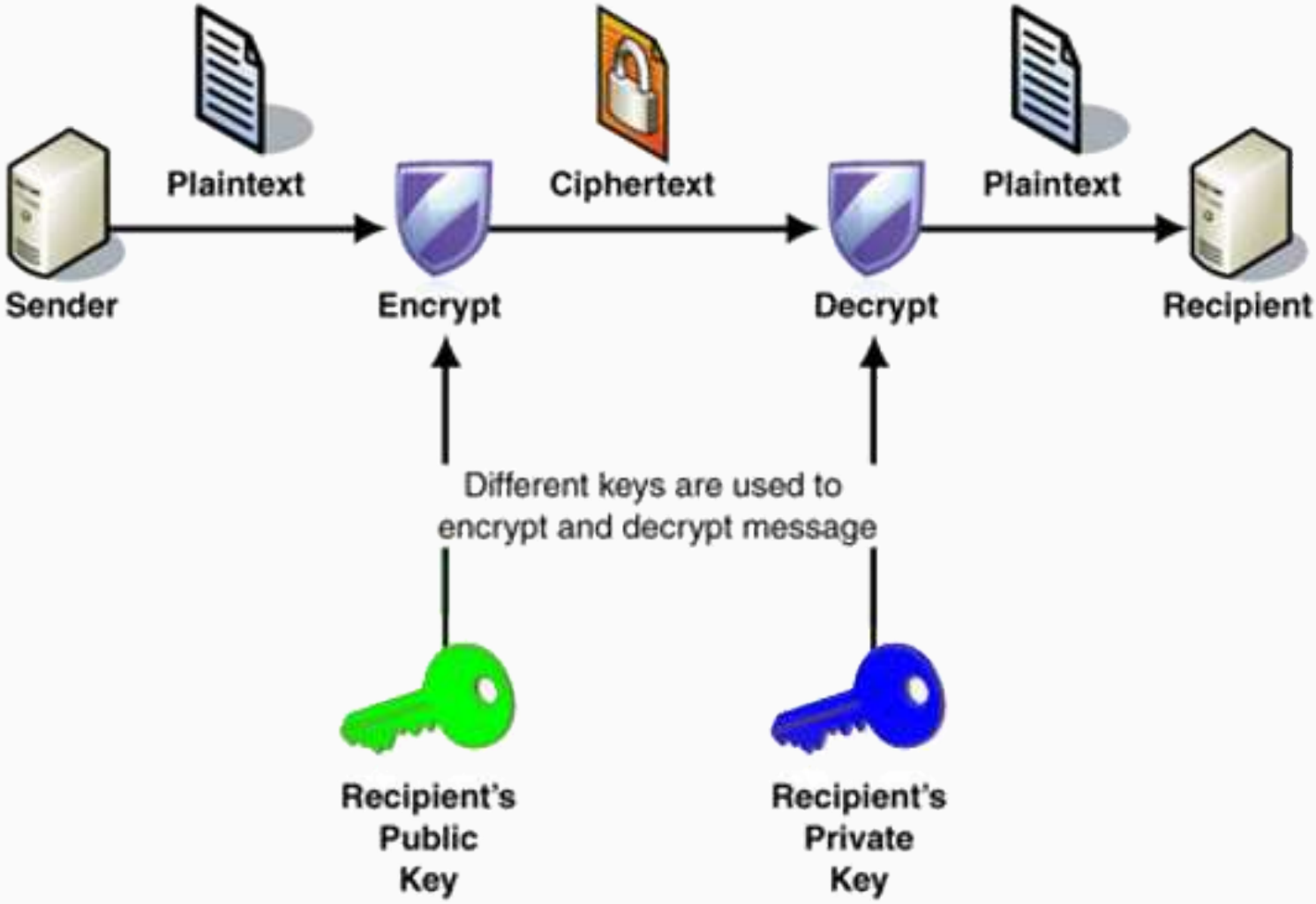
4 Mineração

5 Transação

6 Criptografia

A criptografia estuda o **embaralhamento** de dados (cifração) tal que só uma informação adicional secreta, a **chave**, permite **desfazê**-lo (decifração). Ela é **assimétrica** se

chave para cifrar ≠ chave para decifrar.



Chave pública e privada

Há duas chaves, uma chave **pública** e outra **privada**. Dois usos:

- (**Cifração**) A chave **pública** usa-se para **cifrar**, e a chave *privada* para *decifrar*,

  $\Longrightarrow$ pode ser **transferido** por qualquer cifrador um texto ao decifrador e *a ele só*.

- (**Assinatura Digital**) A chave **privada** usa-se para **cifrar**, e a chave *pública* para *decifrar*,

  $\Longrightarrow$ pode ser **verificado** por qualquer decifrador se um texto origina do cifrador e *dele só*.

Inventores da Ideia da Criptografia Assimétrica

A criptografia assimétrica foi sugerida pela primeira vez, publicamente, em Diffie e Hellman (1976). Antes (uns 20 anos), somente os serviços secretos tiveram consciência desta técnica.

Figura 21: Diffie e Hellman

Funções sobre Conjuntos Discretos

Na criptografia assimétrica,

- a *facilidade* de cifrar (um número), e
- a *dificuldade* de decifrar (um número)

baseiam-se em uma função invertível tal que

- ela é **facilmente** computável, mas
- a sua função inversa é **dificilmente** computável.

Exemplos para tais funções são

- a *exponenciação* $x \mapsto g^x$ (no algoritmo `Diffie-Hellman`), e
- a *potenciação* $x \mapsto x^e$ (no algoritmo `RSA`)

mas **não** definidas sobre $\mathbb{Z}$, porque $\mathbb{Z}$ é incluso em $\mathbb{R}$ e ambas as funções, exponenciação e potenciação, são **contínuas** sobre $\mathbb{R}$.

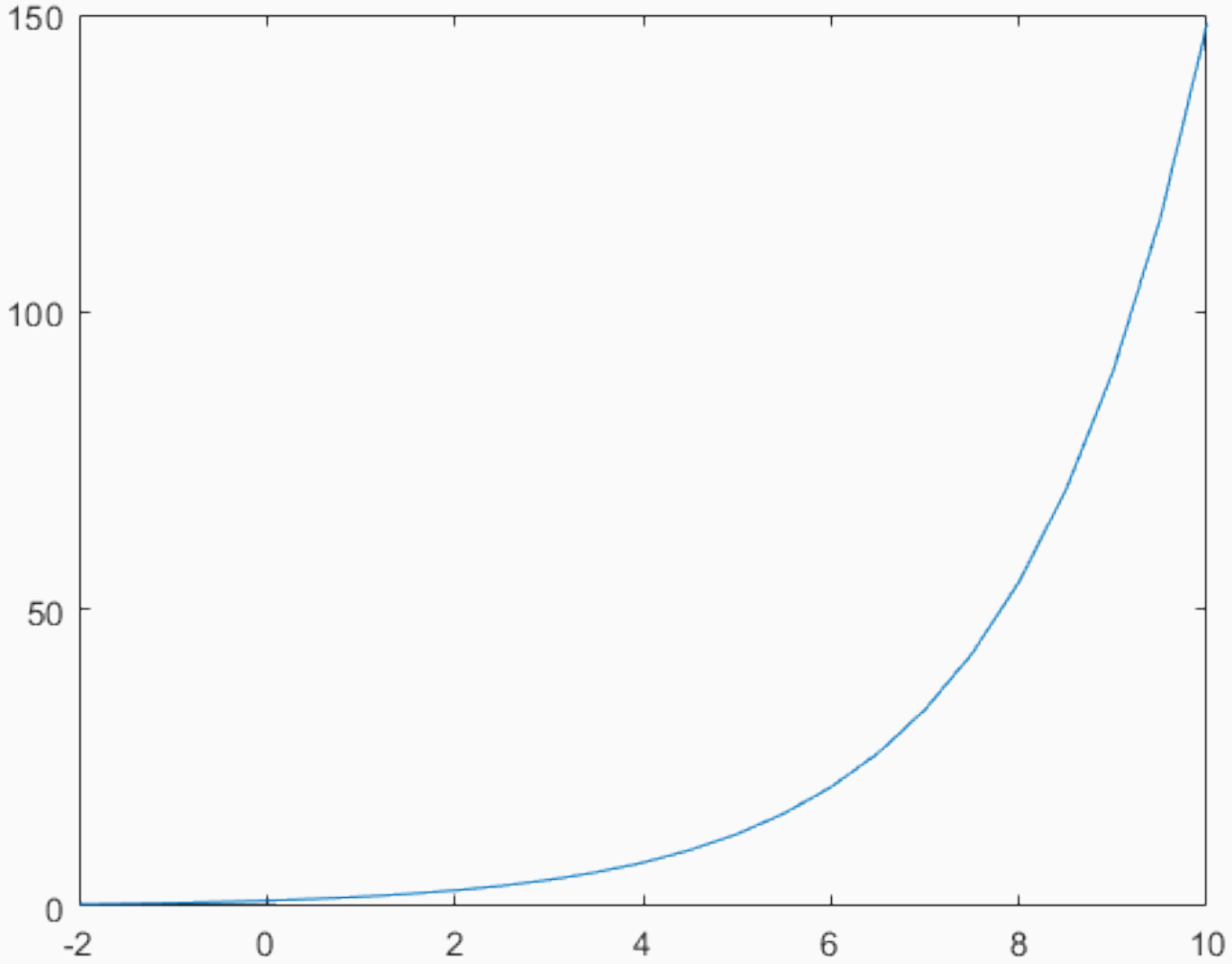


Figura 22: Exponencial $x \mapsto e^x$

Se estas funções fossem definidas sobre o anel $\mathbb{Z}$ e o qual é incluso em $\mathbb{R}$, os seus **inversos** podiam ser **aproximados** sobre $\mathbb{R}$, por exemplo, pelo **Método da Bisseção**: Dado y_0, encontrar um x_0 tal que $f(x) = y_0$ equivale encontrar um **zero** x_0 da função

$$x \mapsto f(x) - y_0.$$

1. (*Inicialização*) Escolha um intervalo $[x', x'']$ tal que

$$f(x') < 0 \quad \text{e} \quad f(x'') > 0.$$

2. (*Recalibração*) Calcule o seu meio $x''' := \frac{x'+x''}{2}$. Temos
   - ou $f(x''') = 0$, então $x''' = x_0$,
   - ou $f(x''') < 0$, então substituta a borda esquera x' por x''',
   - ou $f(x''') > 0$, então substituta a borda direita x'' por x'''.

   e itere com as novas bordas do intervalo.

Pelo **Teorema do Valor Intermediário**, o zero é garantido de estar no intervalo, que a cada passo diminui e converge à interseção.

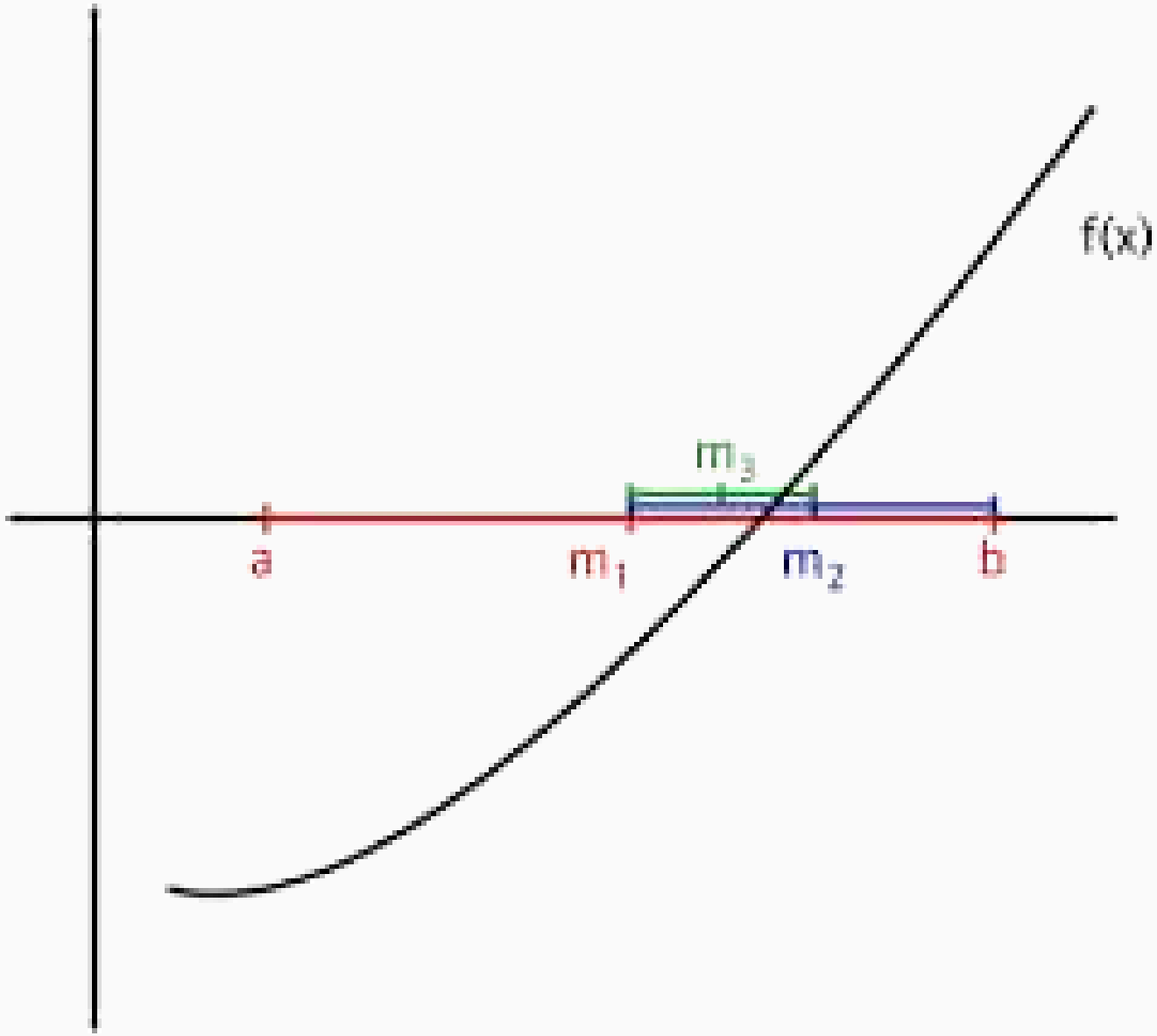


Figura 23: Bisseção de uma função contínua

O Anel dos Números Inteiros

Denotem

$$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\} \quad \text{e} \quad \mathbb{R}$$

os *anéis* dos números inteiros e dos números reais (= a reta).

Anel

Um **anel** (comutativo com 1) é

- um conjunto que contém
  - $\mathbf{0}$ (= o elemento neutro da adição), e
  - $\mathbf{1}$ (= o elemento neutro da multiplicação);
- sobre o qual operam
  - a **adição** $+$ (e o seu *inverso* $-$), e
  - a **multiplicação** $\cdot$,

  que satisfazem a lei *associativa*, *comutativa* e *distributiva*.

Anéis Finitos

Para evitar a iterada aproximação ao zero e assim **dificultar** a computação do inverso (além de facilitar a computação da função), a criptografia assimétrica desenrola-se sobre os **anéis finitos**

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, ..., m-1\}.$$

Neles, vale $m = 1 + \cdots + 1 = 0$ e **toda adição (e logo toda multiplicação e toda potenciação) tem resultado** $< m$ e assim $\mathbb{Z}/m\mathbb{Z}$ é como anel **não** incluso em $\mathbb{R}$. Por exemplo, para $m = 7$, vale

$$2^2 = 2 \cdot 2 = 4 \quad \text{e} \quad 3^2 = 3 \cdot 3 = 7 + 2 = 0 + 2 = 2.$$

Introduzamos estes anéis finitos primeiro pelo exemplo $\mathbb{Z}/12\mathbb{Z}$ (= o anel dos números das horas do relógio) e depois em geral.

Relógio = Anel dos números $1, 2, ..., 11, 12 = 0$

Figura 24: Relógio como Anel dos números $1, 2, ..., 11, 12 = 0$

O Anel $\mathbb{Z}/7\mathbb{Z}$ para $m = 7$

Figura 25: Os comprimidos semanais

O Anel $\mathbb{Z}/m\mathbb{Z}$ para $m = 15$

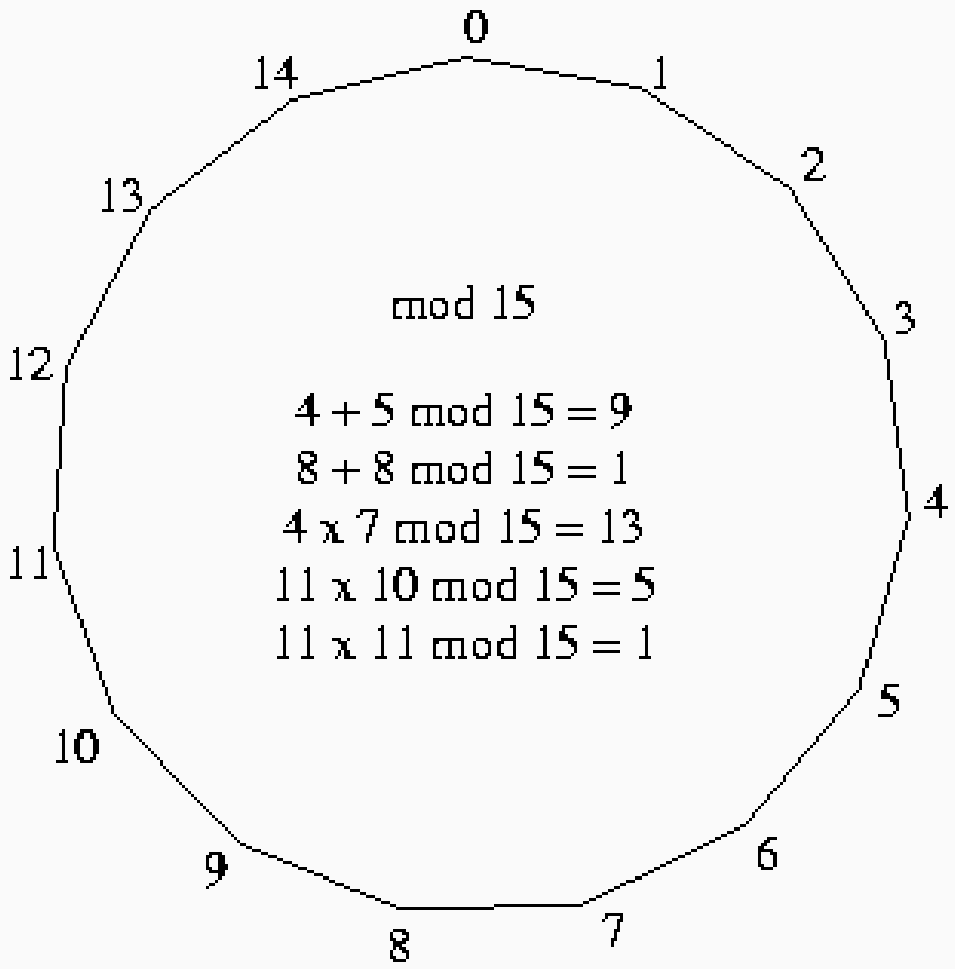


Figura 26: O relógio com 15 horas

Aritmética do Relógio

O exemplo protótipo de aritmética modular é a aritmética do relógio, em que vale a equação

$$12 = 0,$$

e que implica, entre outros, as equações

$$9 + 4 = 1 \quad \text{e} \quad 1 - 2 = 11;$$

Quer dizer,

- 4 horas depois das 9 horas é 1 hora, e
- 2 horas antes da 1 hora são 11 horas.

Formalmente, derivamos estas equações das igualdades

$$9+4 = 13 = 12+1 = 0+1 = 1 \quad \text{e} \quad 1-2 = -1+0 = -1+12 = 11.$$

Podemos ir mais longe: $9 + 24 = 9$, quer dizer se agora são 9 horas, então 24 horas mais tarde também. Formalmente,

$$9 + 24 = 9 + 2 \cdot 12 = 9 + 2 \cdot 0 = 9.$$

Em geral, para quaisquer a e x em $\mathbb{Z}$,

$$a + 12 \cdot x = a$$

ou, equivalentemente, para quaisquer a e b em $\mathbb{Z}$,

$$a = b \quad \text{se } 12 \mid a - b$$

Congruência Modular

*Seja $m \geq 1$ um inteiro. Os números inteiros a e b são **congruentes módulo** m ou, em fórmulas,

$$a \equiv b \mod m.$$

se $m \mid a - b$, isto é se a sua diferença $a - b$ é divisível por m. Ou, em outras palavras, se deixam o **mesmo resto** divididos por m.

Construção

- como **conjunto**

$$\mathbb{Z}/m\mathbb{Z} := \{0, ..., m-1\};$$

e como **aplicação** $\bar{\cdot}\colon \mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$, o **resto** dividido por m,

$$x \mapsto r \quad \text{onde } x = qm + r \quad \text{com } r \text{ em } \{0, ..., m-1\};$$

- como **elementos neutros** da adição e multiplicação 0 e 1,
- como **operações** $+$ e $\cdot$ os **restos** da **soma** e do **produto** dividido por m,

$$x+y := r \quad \text{onde } x+y = qm+r \quad \text{com } r \text{ em } \{0, ..., m-1\}, \text{ e}$$

$$x \cdot y = r \quad \text{onde } x \cdot y = qm + r \quad \text{com } r \text{ em } \{0, ..., m-1\}.$$

O anel $\mathbb{Z}/p\mathbb{Z}$ para p primo é um *corpo*, isto é, nele podemos dividir por todos os números (exceto 0); é denotado $\mathbf{F}_p$.

Tabelas de Adição e Multiplicação para $m = 4$

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

*	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

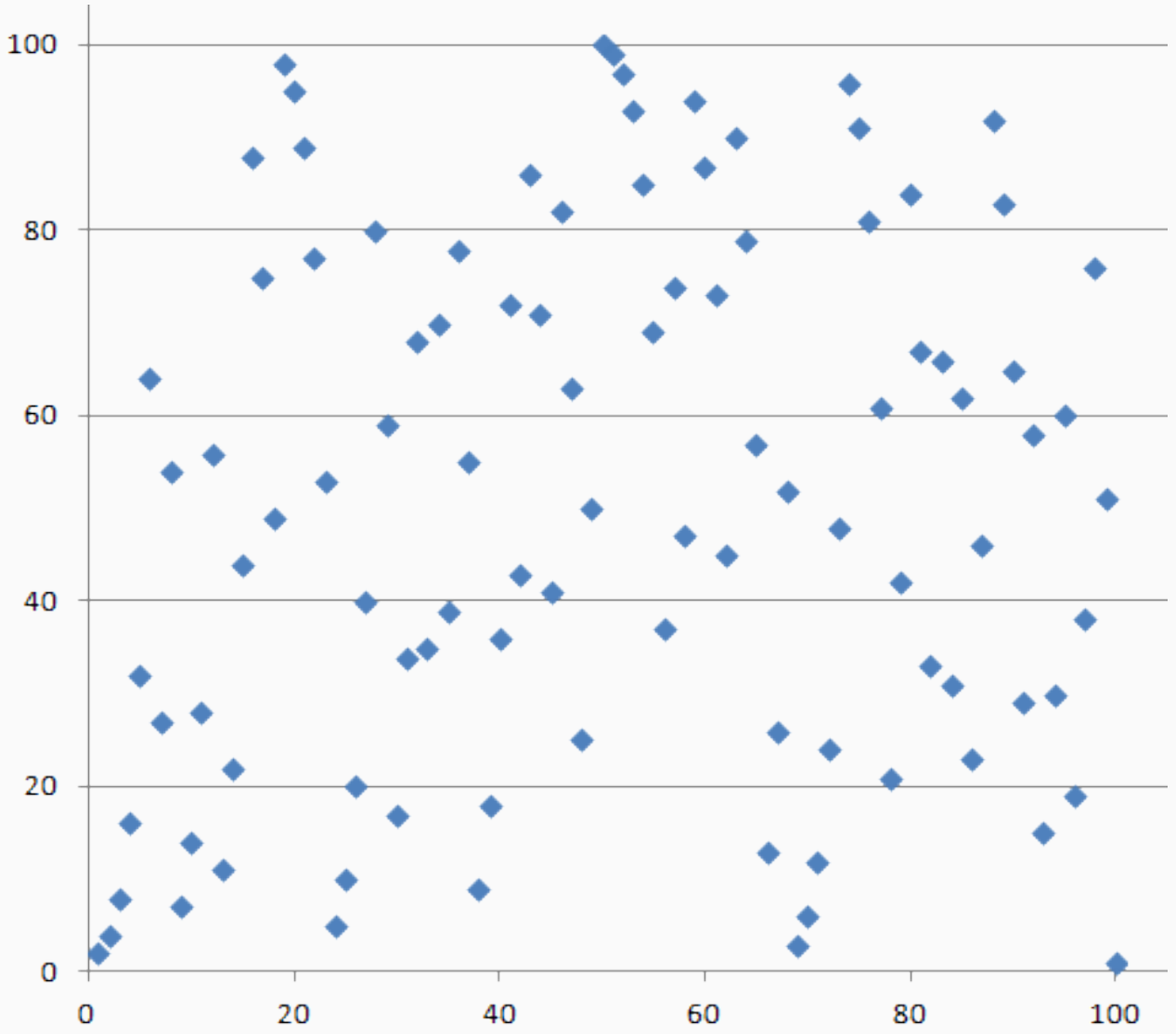


Figura 27: A exponencial com base 2 em $\mathbb{Z}/101\mathbb{Z}$

1 Cadeia

2 Laço

3 Bloco

4 Mineração

5 Transação

6 Criptografia

Curvas Elípticas

Uma curva E sobre um corpo (de característica $\neq 2, 3$) é *elíptica* se dada por uma equação

$$y^2 = x^3 + ax + b$$

para coeficientes a e b tais que a curva não seja *singular*, isto é, que a sua *discriminante* não se esvaía, $4a^3 + 27b^2 \neq 0$.

Após escolha de um domínio (por exemplo, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ ou $\mathbf{F}_p$ para um número primo p) os pontos (x, y) que resolvem esta equação formam uma curva no plano sobre ele.

Para o domínio $\mathbb{R}$, as curvas assumem as seguintes formas no plano real ao a e b variarem:

Figura 28: Curvas elípticas reais

Enquanto sobre os corpos finitos, obtemos um conjunto discreto de pontos (simétrico em volta do eixo horizontal médio).

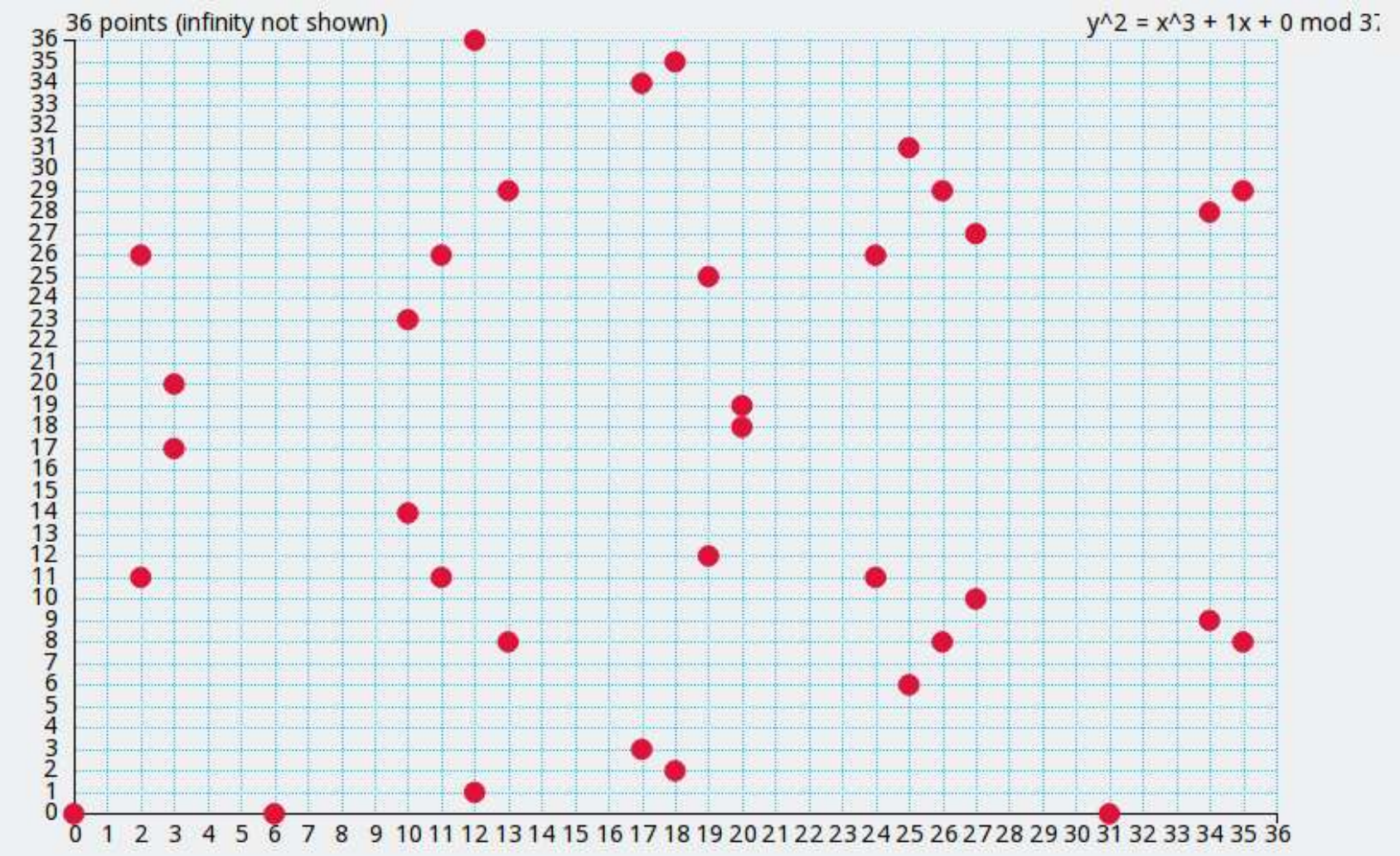


Figura 29: Curva elíptica modular

A equação $y^2 = x^3 + ax + b$ é a *forma de Weierstraß*, mas existem várias outras que se revelaram computacionalmente mais eficientes, como a de *Montgomery*

$$\mathrm{B}y^2 = x^3 + \mathrm{A}x^2 + x \quad \text{onde } \mathrm{B}(\mathrm{A}^2 - 4) \neq 0$$

que pode ser transformada em uma equação de Weierstrass pela substituição

$$(x, y) \mapsto (t, v) = \left(\frac{x}{\mathrm{B}} + \frac{\mathrm{A}}{3\mathrm{B}}, \frac{y}{\mathrm{B}}\right) \quad \text{com } a = \frac{3 - \mathrm{A}^2}{3\mathrm{B}^2} \text{ e } b = \frac{2\mathrm{A}^3 - 9\mathrm{A}}{27\mathrm{B}^3}$$

- Se a característica é 2, isto é, $\mathbb{F}_q$ com $q = 2^n$, a equação tem a forma $y^2 + cxy + dy = x^3 + ax + b$.

Exemplo usado na Criptografia

A Curve25519 com

$$y^2 = x^3 + 486662x^2 + x$$

sobre $\mathbf{F}_p (:= \mathbb{Z}/p\mathbb{Z})$ com $p = 2^{255} - 19$ (de onde o seu nome).

O seu número de pontos é

$$\#\mathrm{E} = 2^{252} + 27742317777372353535851937790883648493.$$

Esta curva tornou-se popular como alternativa imparcial no lugar das curvas recomendadas, e logo desconfiadas, pelo NIST (= National Institute for Standards and Technology).

Adição Geométrica

Entre todas as curvas, a graça das elípticas (dadas por uma equação $y^2 = x^3 + ax + b$) é que permitem **somar pontos**: A soma r de dois pontos p e q é definida por

$$p + q + r = 0 \quad \text{se uma reta passa por } p, q \text{ e } r.$$

O grupo aditivo dado por uma curva elíptica é obtido assim:

- os **elementos** são os seus **pontos**, isto é, os pares (x, y) com entradas no corpo que satisfazem a equação;
- o elemento **neutro** 0 é (geralmente) o ponto **infinito** $(0, 0)$;
- a **adição** é geometricamente dada por $p + q + r = 0$ se os pontos p, q e r são **colineares**, isto é, uma reta passa por todos eles.

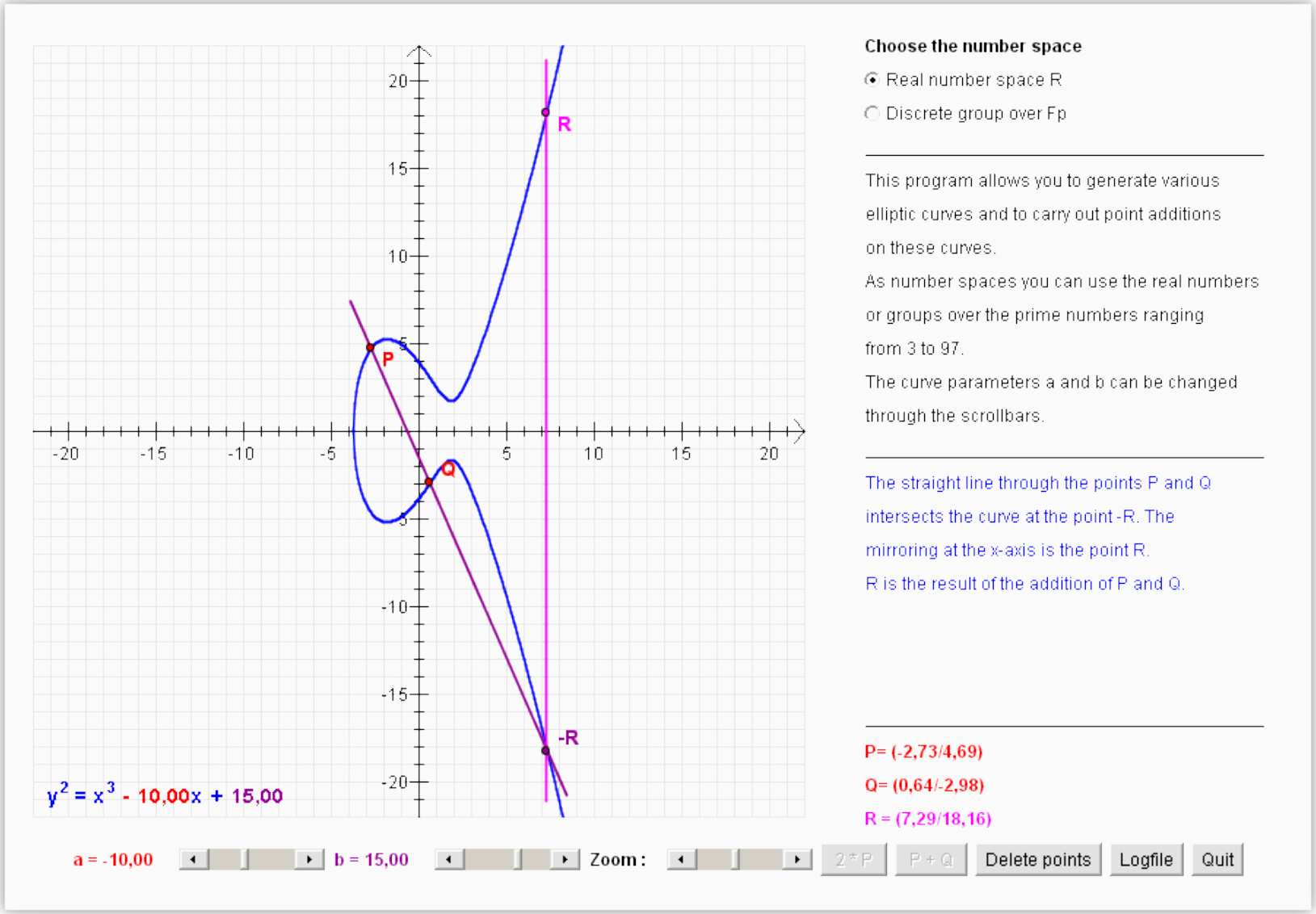


Figura 30: Adição de dois pontos nos números reais no CrypTool 1

Adição Algébrica

Expressa por coordenadas cartesianas a adição de dois pontos de uma curva elíptica é dada por uma fórmula **algébrica**, isto é, envolve unicamente as operações básicas da **adição**, **multiplicação** e **potenciação**: Denote

$$\mathrm{P} + \mathrm{Q} = \mathrm{R} \quad \text{e} \quad (x_p, y_p) + (x_q, y_q) = (x_r, y_r).$$

Se a curve E é dada por $x^3 + ax + b$, então

$$x_r = \lambda^2 - x_p - x_q \quad \text{e} \quad y_r = \lambda(x_p - x_r) - y_p \quad (*)$$

onde λ é a inclinação da reta que passa por P e Q,

$$\lambda = \frac{y_q - y_p}{x_q - x_p} \quad \text{caso } x_q \neq x_p, \quad \text{e} \quad \lambda = \frac{3x_p^2 + a}{2y_p} \quad \text{caso } x_q = x_p.$$

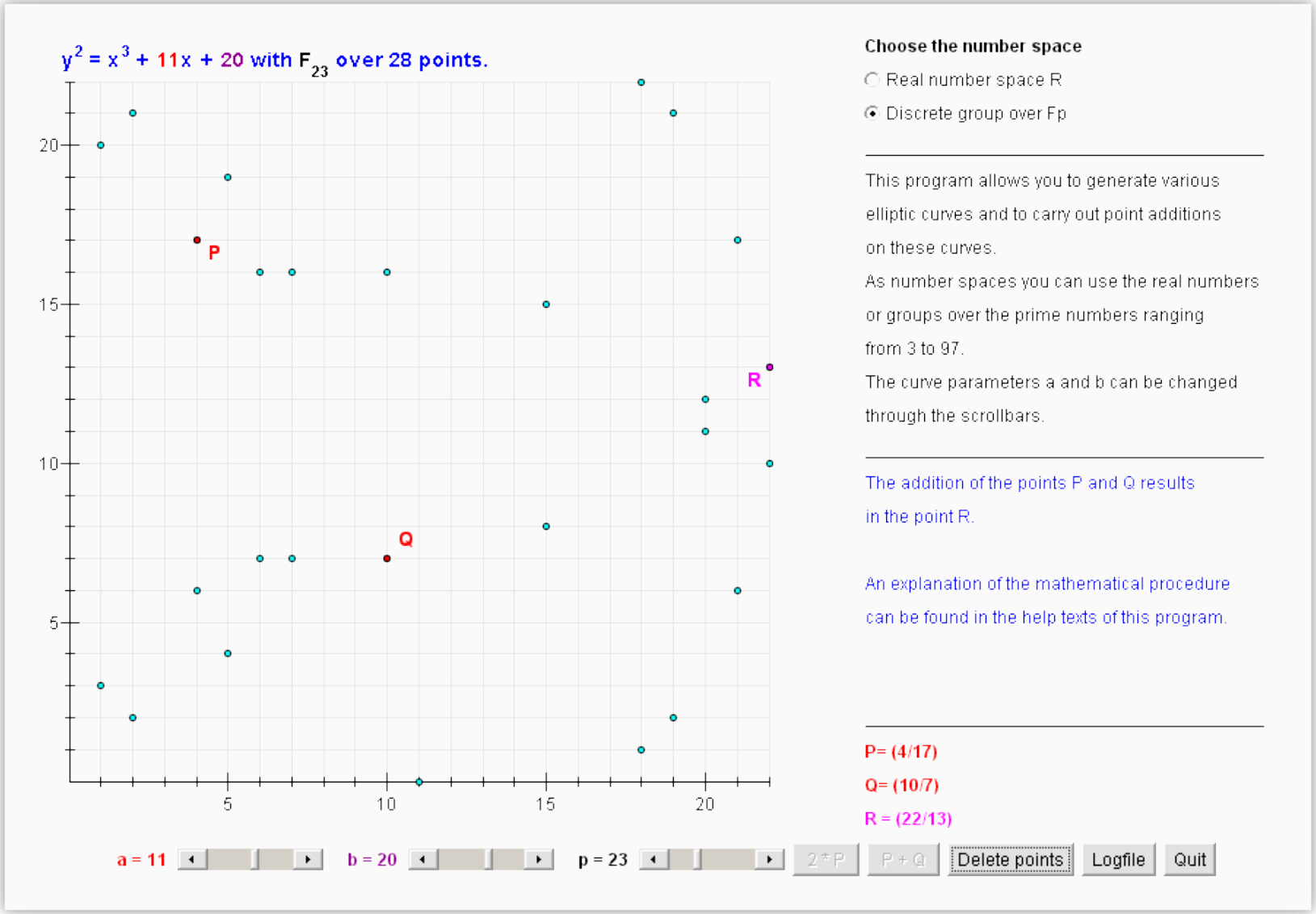


Figura 31: Adição de dois pontos sobre um corpo finito no CrypTool 1

Troca de Chaves (Clássica) de Diffie-Hellman

Para Alice e Bob construírem uma chave secreta através de um canal inseguro, combinam primeiro

- um número primo p *apropriado*, e
- um número natural g *apropriado*.

e depois

1. Alice, para gerar **uma metade** da chave, escolhe a,
   - calcula $\mathrm{A} \equiv g^a \bmod p$, e
   - transmite A ao Bob.
2. Bob, para gerar **outra metade** da chave, escolhe b,
   - calcula $\mathrm{B} \equiv g^b \bmod p$, e
   - transmite B à Alice.
3. A chave secreta **mútua** entre Alice e Bob é

$$c := \mathrm{A}^b = (g^a)^b = g^{ab} = g^{ba} = (g^b)^a = \mathrm{B}^a \bmod p.$$

Diffie-Hellman com Curvas Elípticas Finitas

Ao restringirmos às soluções (x, y) em $\mathbf{F}_p \times \mathbf{F}_p$ para um grande primo p e fixarmos um tal ponto P,

- enquanto, dado um número n, é **fácil** computar

$$\mathrm{Q} = n\mathrm{P} = \mathrm{P} + \ldots + \mathrm{P},$$

- em contraste, dado $\mathrm{Q} = \mathrm{P} + \ldots + \mathrm{P}$, é **dificílimo** computar quantas vezes P foi adicionado, computar n com $\mathrm{Q} = n\mathrm{P}$.

Na criptografia por curvas elípticas ECC, em vez de:

- **multiplicarmos** repetidamente (n vezes) a **base** g

$$g^n = g \cdots g,$$

- **adicionamos** repetidamente (n vezes) um **ponto** G, isto é,

$$n \cdot \mathrm{G} = \mathrm{G} + \cdots + \mathrm{G}.$$

Troca de Chaves

No protocolo ECDH (Elliptic Curve Diffie-Hellman), para Alice e Bob construírem uma chave secreta, combinam

- um número primo p *apropriado*,
- uma curva elíptica E *apropriada* sobre $\mathbf{F}_p$, e
- um ponto G *apropriado* em E.

1. Alice, pra gerar **uma metade** da chave, escolhe a inteiro,
   - calcula $\mathrm{A} \equiv a\mathrm{G}$, e
   - transmite A ao Bob.
2. Bob, pra gerar **outra metade** da chave, escolhe um b inteiro,
   - calcula $\mathrm{B} \equiv b\mathrm{G}$, e
   - transmite B à Alice.
3. A chave secreta **mútua** c entre Alice e Bob é

$$c := b\mathrm{A} = ba\mathrm{G} = ab\mathrm{G} = a\mathrm{B}$$

Tamanho de Chaves em bits

Esta Tabela compara os tamanhos de chaves em bits a um nível de segurança comparável entre um algoritmo simétrico como o `AES`, o `Diffie-Hellman` comum e o elíptico, em particular:

- uma chave elíptica pode ser compartilhada por soletração (são 40 letras na notação hexadecimal),
- uma chave comum tem que ser compartilhada por um arquivo (e usa-se uma impressão digital para referi-la.)

Chave Simétrica	Chave Asimétrica	Chave Elíptica
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Os algoritmos usando ECC

A criptografia por curvas elípticas ECC (Elliptic Curve Criptography) usa a troca de Chaves de `Diffie-Hellman` para

1. **estabelecer** uma chave secreta, para
2. **transformá-la** por um *hash* criptográfico, para
3. usá-la para cifrar a comunicação por um algoritmo criptográfico **simétrica**.

É padronizado pelo ECIES (Elliptic Curve Integrated Encryption Scheme), um procedimento *híbrido* (**mistura** criptografia **assimétrica** com criptografia **simétrica**).

Isto é, uma vez a chave secreta mútua c estabelecida, Alice e Bob usam-na para cifras **simétricas** como AES.

Passos no ECDSA

Para a Samanta assinar um documento d e o Vitor verificar a assinatura, combinam primeiro

1. uma (potência de um) número **primo** p *apropriado*,
2. uma **curva** elíptica E sobre $\mathbb{F}_p$.
3. um **ponto** G (da curva E) de (alta) ordem n.

1. Samanta, para **criar** uma rubrica (ou *firma*),
   - 1.1 escolhe um número f em $\{1, ..., n-1\}$,
   - 1.2 calcula $\mathrm{F} = f\mathrm{G}$.
   - 1.3 transmite F ao Vitor.
2. Samanta, para **assinar** o (hash do) documento d,
   - 2.1 escolhe um número efêmero e em $\{1, ..., n-1\}$,
   - 2.2 calcula a coordenada $\mathrm{E}_x \bmod n$ de $\mathrm{E} = e\mathrm{G}$; Si $\mathrm{E}_x = 0$, então escolhe outro e;
   - 2.3 resolve $d\mathrm{G} + \mathrm{E}_x\mathrm{F} = \mathrm{S}\mathrm{E} \bmod p$ para S, isto é, calcula $\mathrm{S} \equiv [d + \mathrm{E}_x f]/e \bmod n$; se $d + \mathrm{E}_x f \equiv 0 \bmod n$, então escolhe outro e;
   - 2.4 transmite E_x e S ao Vitor.
3. Vitor, para **verificar** a assinatura,
   - 3.1 calcula $\mathrm{S}' = \mathrm{S}^{-1} \bmod n$,
   - 3.2 calcula $d' = d \cdot \mathrm{S}' \bmod n$,
   - 3.3 calcula $\mathrm{E}' = \mathrm{E}_x \cdot \mathrm{S}' \bmod n$,
   - 3.4 calcula $v = d'\mathrm{G} + \mathrm{E}'\mathrm{F}$, e
   - 3.5 verifica que $v_x = \mathrm{E}_x$.

Anotações

Estão **disponíveis**

- os **eslaides** desta palestra e
- um **manuscrito** sobre criptografia que aprofunda o que vimos

online em `konfekt.bitbucket.io/talks/blockchain`

Referências Bibliográficas

Diffie, Whitfield, e Martin Hellman. 1976. «New directions in cryptography». *IEEE transactions on Information Theory* 22 (6): 644–54.

1. Cadeia
2. Laço
3. Bloco
4. Mineração
5. Transação
6. Criptografia