

## Capítulo

# 3

## Segurança de Aplicações Blockchain Além das Criptomoedas

Alexandre Melo Braga, Fundação CPqD ([ambraga@cpqd.com.br](mailto:ambraga@cpqd.com.br))

Fernando C. Herédia Marino, Fundação CPqD ([fmarino@cpqd.com.br](mailto:fmarino@cpqd.com.br))

Robson Romano dos Santos, Fundação CPqD ([robsons@cpqd.com.br](mailto:robsons@cpqd.com.br))

### *Abstract*

*There is much more to blockchain technology than cryptocurrencies. This chapter gives a broad view to the security of blockchain technology in general, not only limited to cryptocurrencies such as Bitcoin. In fact, cryptocurrencies are treated within this text as examples of specific blockchain technologies, even though quite important ones. In addition, software development of blockchain systems explicitly covers modern platforms such as Hyperledger and Ethereum. This text is introductory and aims to show to software programmers and security experts all those aspects of blockchain technology that are necessary to properly develop secure applications, facilitating further studies. In particular, the chapter covers the fundamentals of blockchain technology, blockchain-based software development, security of blockchain systems, and secure coding of blockchain applications.*

### *Resumo*

*Existe muito mais na tecnologia blockchain que as moedas criptográficas. Este capítulo aborda a segurança de aplicações da tecnologia blockchain de modo amplo e além das criptomoedas como o Bitcoin. As criptomoedas, apesar de bastante importantes, são tratadas no texto como exemplos de tecnologias blockchain específicas. Além disso, os aspectos de construção de sistemas cobrem plataformas blockchain modernas como Hyperledger e Ethereum. O texto é introdutório e tem o objetivo de mostrar aos programadores de software e especialistas em segurança da informação os aspectos da tecnologia blockchain necessários ao desenvolvimento de aplicações seguras, facilitando o aprofundamento em estudos futuros. Em particular, este capítulo cobre os seguintes assuntos: os conceitos fundamentais da tecnologia blockchain, o desenvolvimento de software baseado em blockchain, questões de segurança de sistemas blockchain e codificação segura de aplicações blockchain.*

### 3.1. Introdução

Muito tem sido falado sobre blockchain e criptomoedas. Porém, existe muito mais na tecnologia blockchain que as moedas criptográficas. Este capítulo aborda a segurança de aplicações desenvolvidas com a tecnologia blockchain de modo amplo e além das criptomoedas, como o Bitcoin.

Este texto aborda dois assuntos do universo blockchain que possuem diversos desafios e oportunidades: desenvolvimento de aplicações e segurança de sistemas. O escopo do texto é a utilização correta e segura de plataformas blockchains prontas, por isto a implementação de protocolos de consenso é apenas introduzida brevemente e tratada superficialmente. Ainda, o texto não faz um tratamento exaustivo do tema segurança em blockchains, mas dá detalhes suficientes para fomentar o interesse pelo assunto.

Este texto é introdutório e tem o objetivo de mostrar aos programadores de software e especialistas em segurança da informação os aspectos da tecnologia blockchain necessários ao desenvolvimento de aplicações seguras, facilitando o aprofundamento em estudos futuros. O tratamento dado ao tema é, no geral, prático e fortemente fundamentado em análises de casos reais encontrados na literatura especializada, incluindo análises de programas vulneráveis.

Outros objetivos do texto são os seguintes: apresentar conceitos básicos de blockchain para programadores iniciantes em segurança da informação; mostrar como os conceitos são utilizados em plataformas blockchain modernas, ilustrar vulnerabilidades de software comumente encontradas no desenvolvimento de aplicações blockchain, e finalmente, incentivar a formação de mão-de-obra especializada no desenvolvimento de aplicações blockchain seguras. Este texto busca atender à demanda crescente pelo desenvolvimento seguro de aplicações blockchain tanto de uso público (como as criptomoedas), quanto de uso privado (como os blockchains corporativos).

A tecnologia blockchain tem sido considerada [1] um acelerador de inovação na indústria, sendo baseada nas capacidades emergentes da 3ª plataforma tecnológica, que é caracterizada pela computação ubíqua (em qualquer lugar e hora) e consumida por comunidades colaborativas. A 1ª plataforma foi caracterizada pelos mainframes e redes de dados, já a 2ª foi constituída pela Internet, os PCs e as redes locais [1].

Em geral, privacidade, escalabilidade e interoperabilidade são desafios da tecnologia blockchain comuns a várias aplicações [1]. Outros desafios são a transferência de dados em grandes volumes, a integração aos sistemas existentes e a segurança, que depende, em grande parte, de como a aplicação blockchain é construída [1]. Este texto consolida informação de várias fontes e estudos recentes, apresentando-os de um ponto de vista diferenciado e voltado para o desenvolvimento de aplicações blockchain seguras.

Este texto está organizado da seguinte forma. A Seção 3.2 explica os conceitos fundamentais da tecnologia, enquanto a Seção 3.3 trata o desenvolvimento de software baseado em blockchain. A Seção 3.4 aborda as questões de segurança de sistemas blockchain e a Seção 3.5 detalha a codificação segura de aplicações blockchain. Finalmente, a Seção 3.6 faz considerações finais. Ainda, o leitor interessando pode consultar as referências bibliográficas na Seção 3.7.

### 3.2. Conceitos básicos

Esta seção explica os conceitos fundamentais da tecnologia blockchain. Os seguintes assuntos são tratados: o que é o blockchain, blockchains públicos e privados, consenso em sistemas distribuídos, redes *peer-to-peer*, passos de uma transação blockchain, criptografia para blockchain (funções de hash e assinaturas digitais), a estrutura de uma transação, a estrutura da cadeia de blocos, bifurcação da cadeia de blocos, árvores de Merkle, e propriedades técnicas do blockchain, tais como as seguintes: imutabilidade, atualidade, irrefutabilidade, prevenção à duplicação de transações, transparência, visibilidade pública, descentralização, disponibilidade e desintermediação.

#### 3.2.1. O que é blockchain

Em linhas gerais, um blockchain é uma base (de dados) **de transações** distribuída e compartilhada pelos nós de um sistema distribuído organizado como uma rede *peer-to-peer* (P2P), conforme ilustrado na Figura 3.1. Qualquer nó desta rede, com os direitos de acesso adequados, pode consultar e modificar a base de dados distribuída. Os registros desta base de dados são chamados blocos. A base de dados somente aceita a inclusão de blocos novos e nunca a remoção ou modificação de blocos existentes. Por isto, a coleção de blocos é crescente e guarda a história desde a sua criação até o momento da atualização mais recente.

Um blockchain é um ambiente seguro para registro de transações, uma vez que não há adulteração e nem modificação dos registros já feitos. O blockchain é mantido simultaneamente por todos os nós da rede P2P, não existindo local principal ou preferencial para armazenamento de uma base de dados original. Todo nó tem a sua réplica da base de dados, e todas elas são mantidas integras, consistentes e sincronizadas pelos protocolos de consenso. Este texto adota o termo *ledger* para a base de dados

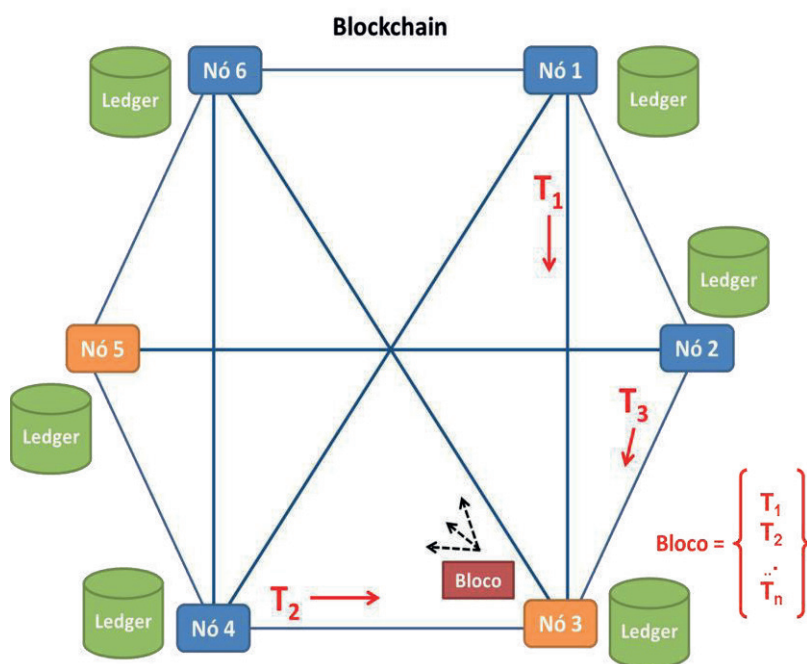


Figura 3.1. Blockchain como uma base de dados distribuída em uma rede P2P.

(coleção crescente de registros de transações) distribuída e blockchain para o sistema distribuído formado pela *ledger* distribuída e os nós da rede P2P.

Uma curiosidade é que, de fato, não existe qualquer informação na *ledger* que se pareça com uma moeda eletrônica (no sentido de uma sequência de bits unicamente identificável, distinguível das demais e transferível), apesar de o termo criptomoeda ser comumente associado à tecnologia blockchain e ao Bitcoin.

### 3.2.1.1. Blockchains públicos e privados

Atualmente, as redes blockchain são divididas em dois grandes grupos. As redes públicas ou de acesso aberto (sem permissão ou *permissionless*) são aquelas em que o acesso pode ser anônimo, as aplicações têm característica aberta e a própria rede segue suas regras (desde que não violem a legislação vigente). Nestas redes, os nós são competidores na criação de blocos e, por isto, não confiam plenamente uns nos outros. Neste caso, a confiança advém da boa execução das regras de consenso e não dos pares.

As redes privadas ou de acesso autorizado (permissionadas ou *permissioned*) geralmente oferecem acesso a usuários identificados, autenticados e autorizados. Nestas redes, os usuários não são anônimos, mas sim grupos selecionados de usuários conhecidos. Por isto, os blockchains permissionados são mais adequados aos ambientes corporativos fechados, estando em conformidade às regras destes ambientes. Ainda, nas redes permissionadas, alguns dos nós da rede P2P podem funcionar como validadores do consenso. A Figura 3.2 mostra algumas características de tais redes. Uma terceira opção são as redes híbridas que combinam características dos blockchains públicos e privados.



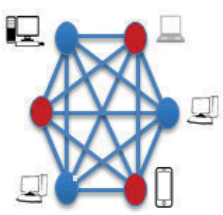

 Nó simples somente inicia ou recebe uma transação  Nó validador valida, inicia ou recebe uma transação	 <p>Privado Permissionado</p>	 <p>Público Não Permissionado</p>
	Acesso à rede	Necessita de autorização
Legislação e regulação	Conforme legislação e regulações	Pode ter regras próprias
Quem são os validadores	Grupo pré-selecionado	Anônimos
Potenciais aplicações	Ambientes corporativos fechados	Aplicações de acesso aberto

Figura 3.2. Rede pública e rede privada.

### 3.2.1.2. A transação

A estrutura de dados de uma transação reflete a semântica da aplicação. No caso das criptomoedas, esta estrutura se parece com um balancete contábil de débito e crédito e é composta dos seguintes elementos (Figura 3.3): um timestamp, o identificador (*hash*) da transação anterior de onde vem o valor de entrada (pode haver mais de um), o valor de entrada, o valor de saída, o endereço de destino (que vai receber o crédito), e uma assinatura digital feita com a chave privada do criador da transação (o debitado) [2].

O fluxo completo de ações para a realização de uma transação blockchain fim-a-fim, detalhado a seguir e ilustrado na Figura 3.4, pode ser dividido em quatro partes [3]: preparação dos clientes, registro da transação, validação do bloco por consenso e consulta ou confirmação. Na Figura 3.4, tendo em vista dois usuários/clientes de um sistema blockchain, Alice e Bob, a preparação dos clientes consiste de dois passos:

1. Alice (Cliente A) cria sua conta (gera/escolhe um endereço);
2. Alice divulga sua conta (endereço) para Bob (Cliente B).

O registro da transação ocorre em outros dois passos:

3. Bob forma uma transação e a assina digitalmente para o endereço de Alice;
4. Bob propaga a transação entre os nós da rede P2P (via seu nó preferido).

O consenso é transparente para os clientes e ocorre em mais dois passos:

5. A transação é incluída em um bloco e os nós da rede trabalham para obter o consenso sobre a criação do bloco, de acordo com as regras de consenso;
6. Os nós da rede P2P propagam seu resultado para outros nós, a transação é aceita

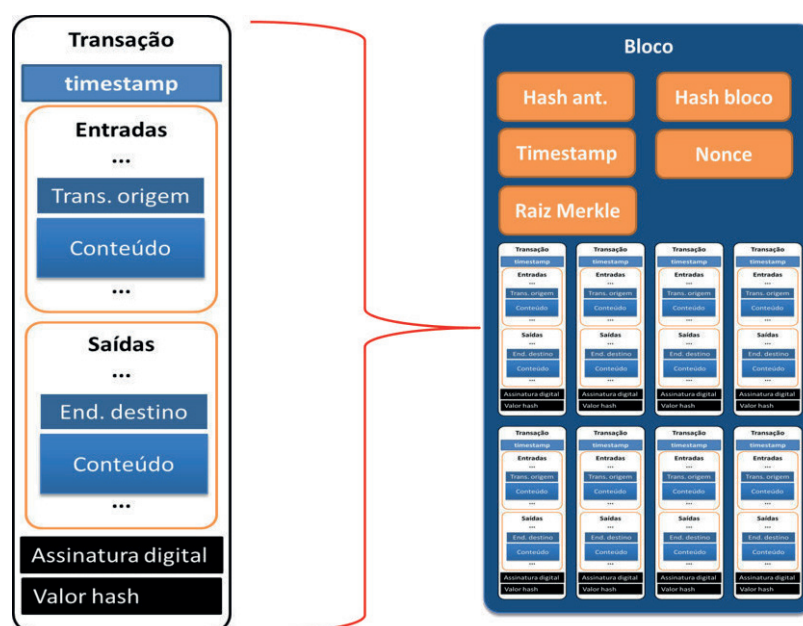


Figura 3.3. A transação e o bloco.

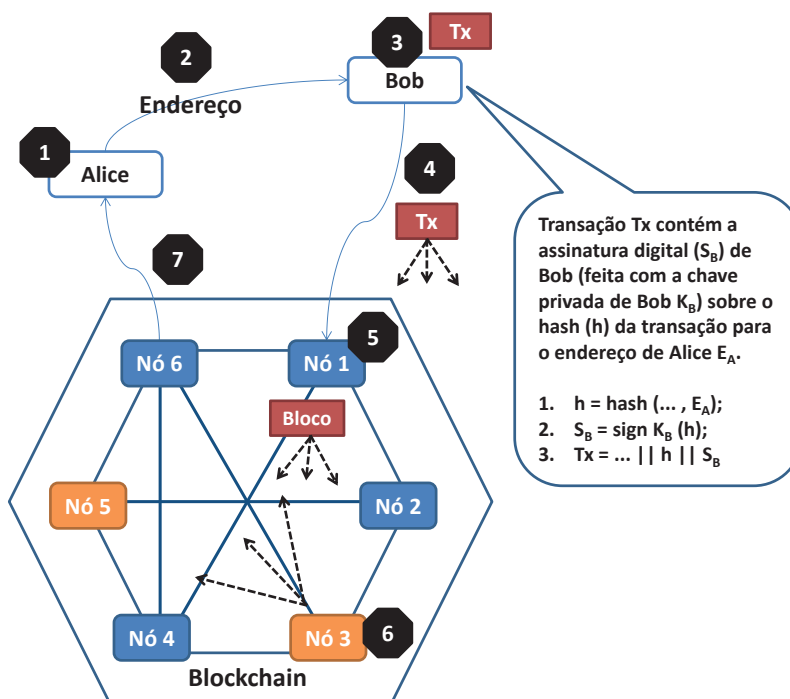


Figura 3.4. Os passos de uma transação.

de acordo com o consenso e passa a fazer parte do blockchain.

Finalmente, a consulta ou confirmação conclui o fluxo em um último passo:

7. Alice consulta a *ledger* e entende que sua transação foi aceita.

Muitas vezes, devido à natureza assíncrona da comunicação e ao tempo relativamente longo (para máquinas e não para pessoas) necessário para a realização do consenso, o último passo (confirmação) não é realizado. Conforme será discutido adiante no texto, muitas fraudes e outros problemas de segurança em transações poderiam ser evitados simplesmente aguardando o tempo necessário e verificando a realização bem sucedida da transação.

O modelo de computação distribuída utilizado pelo blockchain pode ser mais bem entendido a partir de um exemplo de criptomoeda como o Bitcoin. Geralmente, uma criptomoeda está associada a uma rede pública (*permissionless*), conforme suas próprias regras. Os nós da rede são encarregados de validar as transações monetárias envolvendo a criptomoeda. No caso do Bitcoin, qualquer pessoa com um computador pode se tornar um nó processador de transações e validador de blocos da rede P2P.

Um nó validador escolhe um número definido de transações não processadas (pendentes) para montar o bloco. Neste exemplo, O bloco é um conjunto de transações monetárias utilizando a criptomoeda em questão. O processo de validação ocorre quando um nó da rede, seguindo um conjunto de regras bem definidas, consegue montar um bloco reconhecido como válido pelos outros nós da rede.



### 3.2.1.3. A cadeia de blocos

As transações são incluídas em blocos (Figura 3.3), que estão ordenados em uma cadeia, formando uma estrutura de dados conhecida em computação como lista ligada (Figura 3.5). O bloco mais recente é a cabeça (*head*) da cadeia. Cada bloco contém um conjunto de transações e um cabeçalho composto dos seguintes itens: o *hash* do bloco anterior (ponteiro para o item anterior da lista), um número pseudoaleatório único (*nonce*), e o *hash* da raiz da árvore de transações no bloco.

Vários nós estão fazendo a mesma coisa simultaneamente, processando blocos, mas não necessariamente sobre as mesmas transações. A montagem do bloco pelo nó depende das transações ainda não processadas (pendentes) visíveis ao nó. Há uma competição entre os nós para construir blocos e validar transações antes dos concorrentes. Quando uma validação ocorre, todos os nós da rede são informados e o bloco, já autenticado pelos demais nós, é inserido na cadeia com as transações devidamente validadas.

Esta dinâmica da cadeia de blocos é devida à competição entre os nós da rede e pode resultar em pequenas falhas ou bifurcações na cadeia, aceitas como parte do processo, e corrigidas com o tempo. Na Figura 3.6, a competição entre os nós resulta em dois blocos 101, mas só um deles é aceito pela maioria dos nós. Depois, dois blocos 104 são construídos, por um tempo não há consenso. Finalmente, os nós da rede adotam o ramo mais longo da cadeia.

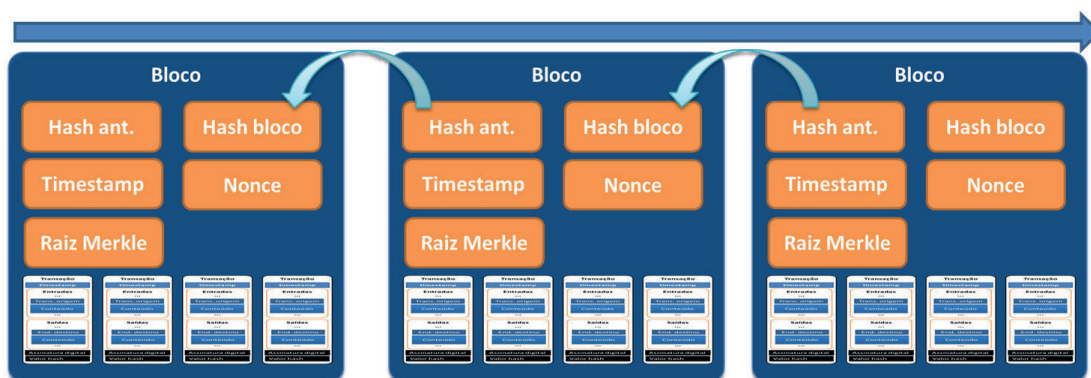


Figura 3.5. A cadeia de blocos.

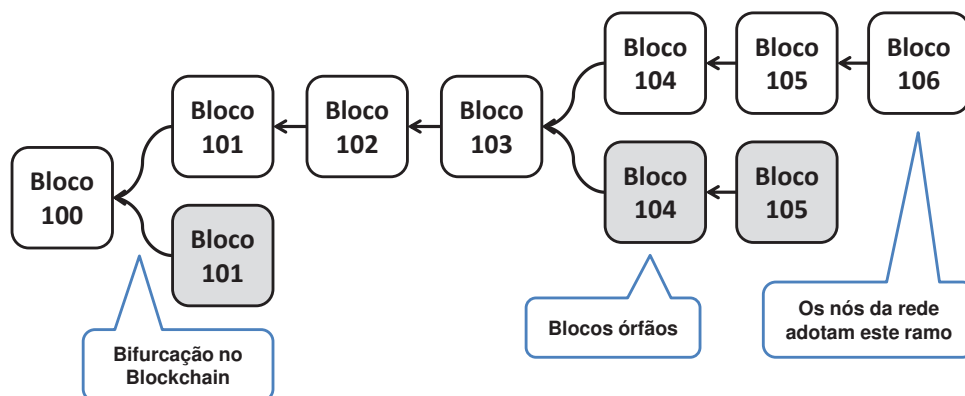


Figura 3.6. A dinâmica da cadeia de blocos.

### 3.2.1.4. Árvores Merkle

As transações dentro de um bloco estão ordenadas entre si de acordo com uma estrutura em árvore binária baseada em *hashes*, que é conhecida como *Merkle Tree*. Nesta estrutura, as folhas da árvore são os *hashes* das transações e os *hashes* dos pais são calculados com os *hashes* dos filhos. Por exemplo, os *hashes* dos ramos imediatos são calculados com os *hashes* das folhas, os *hashes* dos ramos intermediários são calculados com os *hashes* dos ramos imediatos, sucessivamente, até o cálculo do *hash* da raiz da árvore, que é incluído no bloco.

Esta estrutura em árvore acelera a operação de verificação do bloco (se a transação pertence ao bloco) e pode ser feita em  $\log(n)$  computações de *hash*, onde  $n$  é o tamanho da árvore. A verificação do *hash* de uma transação só usa o ramo da árvore (*Merkle branch*) em que a transação está e que é necessário para verificar o *hash* da transação.

A Figura 3.7 ilustra a estrutura da árvore Merkle e a verificação de uma transação. À esquerda da figura, observa-se que a raiz da árvore Merkle é obtida pelo cálculo dos *hashes* de dois ramos da árvore  $H(H12+H34)$ , onde  $H12$  é o *hash* dos *hashes* das transações  $T1$  e  $T2$ ,  $H12 = H(H1 + H2)$ , com  $H1 = H(T1)$  e  $H2 = H(T2)$ . De modo análogo  $H34$  é o *hash* dos *hashes* das transações  $T3$  e  $T4$ ,  $H34 = H(H3 + H4)$ , com  $H3 = H(T3)$  e  $H4 = H(T4)$ .

À direita da figura, observa-se que a verificação da transação  $T4$  requer os *hashes* da própria  $T4$ ,  $H(T4)$ , e também o *hash* da transação  $T3$ , no mesmo ramo, para o cálculo do *hash* intermediário  $H(H(T3)+H(T4))$ . Além disso, é necessário o valor *hash*  $H12$  para a verificação da raiz Merkle. Assim, é possível verificar rapidamente a integridade de um bloco e das transações incluídas nele.

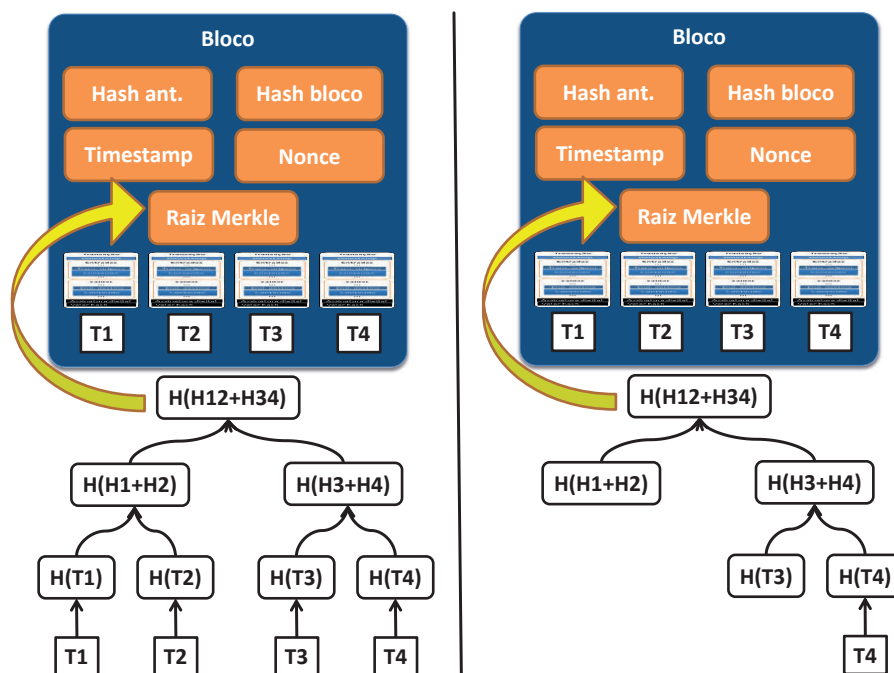


Figura 3.7. Árvore Merkle do bloco (esquerda) e verificação da transação  $T4$  (direita).



### 3.2.1.1. Consenso em sistemas distribuídos

No blockchain, os nós da rede P2P decidem consensualmente sobre a ordem em que as transações são registradas na *ledger*. Consenso distribuído é um termo da ciência da computação usado na disciplina de algoritmos distribuídos [4] e é um aspecto crítico da tecnologia blockchain e das criptomoedas.

Em um sistema distribuído, dois ou mais nós de uma rede trabalham de modo coordenado por um objetivo comum. Idealmente, usuários do sistema distribuído o percebem como uma plataforma única, isto é, a distribuição é transparente. Um nó é um participante individual do sistema distribuído. Os nós podem trocar mensagens entre si. Os nós podem ser honestos, defeituosos ou maliciosos. Um nó de comportamento indeterminado ou inesperado (honesto ou malicioso) é chamado de nó bizantino.

Os principais desafios de projeto de sistemas distribuídos com alta disponibilidade são a tolerância a falhas (mais precisamente, a partições) e a coordenação entre nós. Em geral, sistemas distribuídos apresentam um compromisso entre três propriedades [5]:

- **Consistência:** todos os nós do sistema distribuído têm os dados mais recentes.
- **Disponibilidade:** o sistema está operante, acessível para uso, aceitando requisições e respondendo sem falhas e com dados corretos sempre que é requisitado.
- **Tolerância à partição:** o sistema distribuído continua operando corretamente, mesmo que um grupo de nós falhe. Isto é, ele continua funcionando corretamente com alguns nós honestos, mesmo na presença de um grupo de nós maliciosos.

Em teoria, não é possível para um sistema distribuído atingir plenamente todas as três propriedades [5]. Por isto, na prática, sistemas distribuídos reais adotam decisões de compromisso que privilegiam uma ou duas propriedades, sacrificando pelo menos uma delas. A replicação contribui para a tolerância a falhas. A consistência é obtida com o uso de algoritmos de consenso que vão garantir que todos os nós têm os mesmos dados.

No Bitcoin, a consistência é sacrificada em nome da disponibilidade e da tolerância a partições. Isto significa que a consistência não é obtida simultaneamente com as outras duas propriedades, mas sim gradativamente, com o tempo, à medida que os blocos são validados pelos nós da rede. Vale lembrar a Figura 3.6, em que a competição entre nós da rede resulta em dois blocos 101, mas só um deles é aceito pela maioria dos nós. Depois, dois blocos 104 são construídos, por um tempo não há consenso. Finalmente, os nós da rede adotam o ramo mais longo da cadeia.

Consenso é um processo de acordo entre nós mutuamente suspeitos, que é realizado sobre dados comuns a todos os nós para alcançar uma interpretação em comum da realidade (uma versão da verdade). Consenso significa que quase todos (os envolvidos) concordam. Consenso é diferente de unanimidade, uma vez que nem todos tem que concordar, basta que a maioria concorde. Um mecanismo de consenso possui os seguintes requisitos [5]:

- **Acordo:** todos os nós honestos decidem sobre o mesmo valor.
- **Término:** todos os nós honestos terminam o consenso e chegam a uma decisão.

- **Validade:** o valor acordado pelos nós honestos deve ser um dos valores propostos inicialmente por algum dos nós honestos.
- **Tolerância a partições:** o algoritmo de consenso deve ser capaz de funcionar corretamente mesmo na presença de nós defeituosos ou maliciosos (nós bizantinos).
- **Integridade:** nenhum nó decide (ou vota) mais de uma vez em uma mesma rodada do mecanismo de consenso.

No blockchain, o consenso ocorre entre os nós da rede P2P por meio de métodos compostos por protocolos específicos e regras bem definidas. Todos os nós da rede P2P são envolvidos de algum modo na tomada de decisão por consenso. Trata-se, portanto, de um grupo (ou comunidade) decidindo em conjunto e de modo confiável. Opcionalmente, um nó centralizador (validador) pode coletar e propagar o consenso na rede P2P. O resultado de uma realização do protocolo de consenso deve ser confiável (determinístico) para toda execução.

Existem dois tipos de mecanismos de consenso usados em blockchains [5]:

- Consenso baseado em prova ou liderança, em que um nó líder é eleito para decidir pelos outros ou um nó apresenta provas de que a sua decisão tem mais peso no consenso que a decisão dos outros nós. Exemplos deste tipo de consenso são a prova de participação utilizada pelo Ethereum e a mineração ou prova de trabalho (*proof of work*), utilizada no Bitcoin, em que o nó finaliza a montagem do bloco quando resolve uma expressão matemática computacionalmente custosa.
- Consenso tolerante a falhas (ou consenso bizantino) em que há rodadas de votações até a decisão ser obtida. O consenso bizantino é caracterizado pela necessidade de  $3*n+1$  nós na rede P2P para tolerar  $n$  divergências no consenso. Vale observar que no consenso bizantino (tolerante a partições), três vezes mais nós são necessários para atingir o consenso que no consenso por maioria simples (com duas vezes mais nós, isto é  $2*n + 1$  para detectar  $n$  falhas). Isto ocorre por que no consenso bizantino, o nó malicioso pode responder certo, errado ou maliciosamente (certo ou errado).

A segurança dos protocolos de consenso em blockchain é baseada na suposição de que a maioria dos operadores dos nós da rede P2P (os mineradores) está mais interessada em se beneficiar dos mecanismos de incentivo do protocolo e menos propensa a quebrar as regras. As taxas pagas aos mineradores pelo esforço computacional de construir blocos válidos é um dos mecanismos de incentivo utilizados por muitas criptomoedas.

O mecanismo de consenso é uma das peças mais importantes do blockchain, por que viabiliza o controle descentralizado, sem o qual o blockchain perde o significado e o diferencial tecnológico. A escolha do mecanismo de consenso depende da finalidade do blockchain. Por exemplo, a prova de trabalho é adequada para blockchains públicos e de acesso aberto, onde todo nó pode participar do consenso. Já o consenso bizantino é mais adequado para as comunidades fechadas.

Do ponto de vista de quem desenvolve aplicações sobre plataformas blockchain, os métodos de consenso são uma funcionalidade, serviço ou configuração a ser habilitada e parametrizada. Sendo muitas vezes transparente (em termos programáticos) para o desenvolvedor de aplicações.

### 3.2.2. Criptografia para blockchain

Em geral, blockchain usa criptografia de dois modos. Primeiro, as funções de resumo criptográfico (funções de *hash*) são usadas na geração dos endereços, que consistem de valores *hash* calculados a partir das chaves públicas. Segundo, as assinaturas digitais usadas na garantia de autenticidade e de irrefutabilidade das transações.

A criptografia assimétrica (de chave pública) para assinatura digital é usada para obter integridade, autenticidade e irrefutabilidade. A assinatura digital é o resultado uma operação criptográfica com a chave privada sobre o texto claro. O dono da chave privada pode gerar mensagens assinadas, que podem ser verificadas por qualquer um que conheça a chave pública correspondente. O assinante não pode negar a autoria, pois há uma assinatura digital feita com sua chave privada. Por isto, a assinatura é irrefutável. A assinatura pode ser verificada por qualquer um com a chave pública. Exemplos de algoritmos de assinaturas digitais utilizadas atualmente são o ECDSA com a curva elíptica secp256k1. Mais sobre criptografia pode ser encontrado na referência [6].

A Figura 3.8, adaptada de [6], ilustra um sistema criptográfico assimétrico (de chave pública) para autenticidade, conhecido como assinatura digital. Na figura, Ana assina o texto claro com sua chave privada, produzindo a assinatura digital, que é enviada para Bob, junto com o texto claro. Bob verifica a assinatura com a chave pública de Ana. Blockchains têm adotado a criptografia de curvas elípticas [7] para assinaturas digitais.

Funções de *hash* (ilustradas na Figura 3.9) geram uma sequência de bits, o valor do *hash*, que é único para o documento de entrada da função. O *hash* é muito menor que o documento original e geralmente tem um tamanho fixo de dezenas (algumas centenas) de bits. A função de *hash* é unidirecional porque não é reversível, isto é, não é possível recuperar o documento original a partir da sequência binária do *hash*. Além disso, idealmente, não existem dois documentos que geram o mesmo valor de *hash*. Exemplos de funções de *hash* seguras utilizadas atualmente são o SHA-2 e o SHA-3.

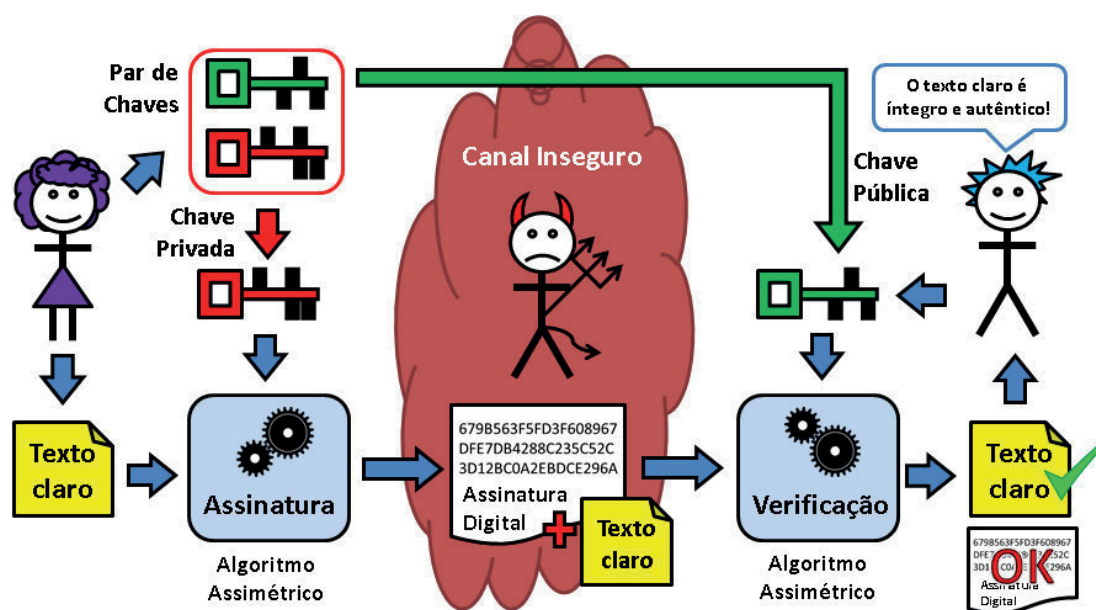


Figura 3.8. Assinatura digital (adaptada de [6]).

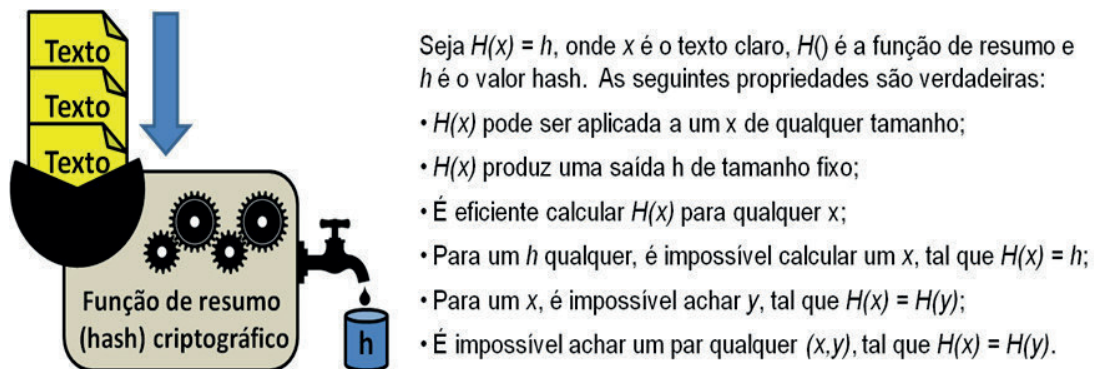


Figura 3.9. Funções de resumo (*hash*) criptográfico (adaptada de [6]).

### 3.2.3. Propriedades técnicas do blockchain

O blockchain tem uma série de propriedades técnicas que são geralmente identificadas como benefícios para as aplicações e os negócios baseados nesta tecnologia. A propriedade mais conhecida em geral é a **imutabilidade**, em que blocos e transações já incluídos na *ledger* são imutáveis. Já a *ledger* somente é alterada de modo incremental e por consenso das partes envolvidas.

A **atualidade** se refere à atualização periódica da *ledger* que sempre ocorre de modo autêntico e legítimo em intervalos curtos de atualização, geralmente próximos do tempo real. Já a **irrefutabilidade** garante que uma transação realizada, e replicada em todos os nós da rede, não pode mais ser negada pelo seu autor.

A **prevenção contra a duplicação** de transações garante que não há registro duplo de transações. Esta propriedade é importante no Bitcoin e outras criptomoedas, pois evita gastar o mesmo valor duas vezes, sendo uma proteção contra o *double spending*.

Duas propriedades relacionadas são a **transparência** e a **visibilidade pública**. Na primeira, todos os nós da rede P2P, assim como os softwares clientes com acesso só para leitura, veem as transações registradas. Na segunda, todos os nós da rede têm acesso a *ledger* e podem verificar a sua legitimidade.

**Descentralização** se refere ao fato de não existir proprietário único da *ledger*, uma vez que todo nó da rede P2P é coproprietário, mantém a sua réplica da *ledger* e contribui para atualizar as outras réplicas. A **disponibilidade** do blockchain é geralmente alta porque alguns nós fora do ar (*off-line*) não impedem o funcionamento dos outros nós, preservando a capacidade de chegar ao consenso. Vale observar que cada mecanismo de consenso requer uma quantidade mínima de nós disponíveis (operantes e conectados) para que o consenso seja viável.

Finalmente, a **desintermediação** é a propriedade emergente da atuação do blockchain como um conector de sistemas complexos (sistemas de sistemas), geralmente eliminando intermediários artificiais nas integrações entre sistemas e abrindo espaço para a simplificação de processos.



### 3.2.4. Todos os conceitos juntos

Um blockchain é um sistema complexo e dinâmico em que, muitas vezes, é difícil visualizar o processo contínuo de validação de transações e inclusão de blocos. A Figura 3.10 tenta capturar em uma única imagem o relacionamento entre os seguintes conceitos: transação, bloco, cadeia, consenso e rede de nós P2P.

Primeiro, uma transação é criada por um usuário em um software de *eWallet*. Em seguida, a transação é propagada para os nós da rede P2P via um nó em particular, onde a transação é validada quanto à integridade e à autenticidade. Se estiver bem construída, a transação é propagada para outros nós da rede para que seja incluída em algum bloco.

A transação é considerada pendente até que seja escolhida por algum dos nós da rede P2P para ser incluída em um bloco. Blocos são criados a todo o momento e incorporam as transações pendentes. Vários nós estão processando blocos simultaneamente, mas não necessariamente sobre as mesmas transações. Por isso, eventualmente, a transação é escolhida por um nó para fazer parte do próximo bloco construído.

O bloco é divulgado para outros nós da rede e validado pelo processo de consenso. Se for válido, o bloco é incorporado à *ledger* e a transação é considerada parte permanente e imutável do blockchain, podendo ser consultada quanto à legitimidade e à integridade. A competição dos nós pelo processamento dos blocos é um processo contínuo.

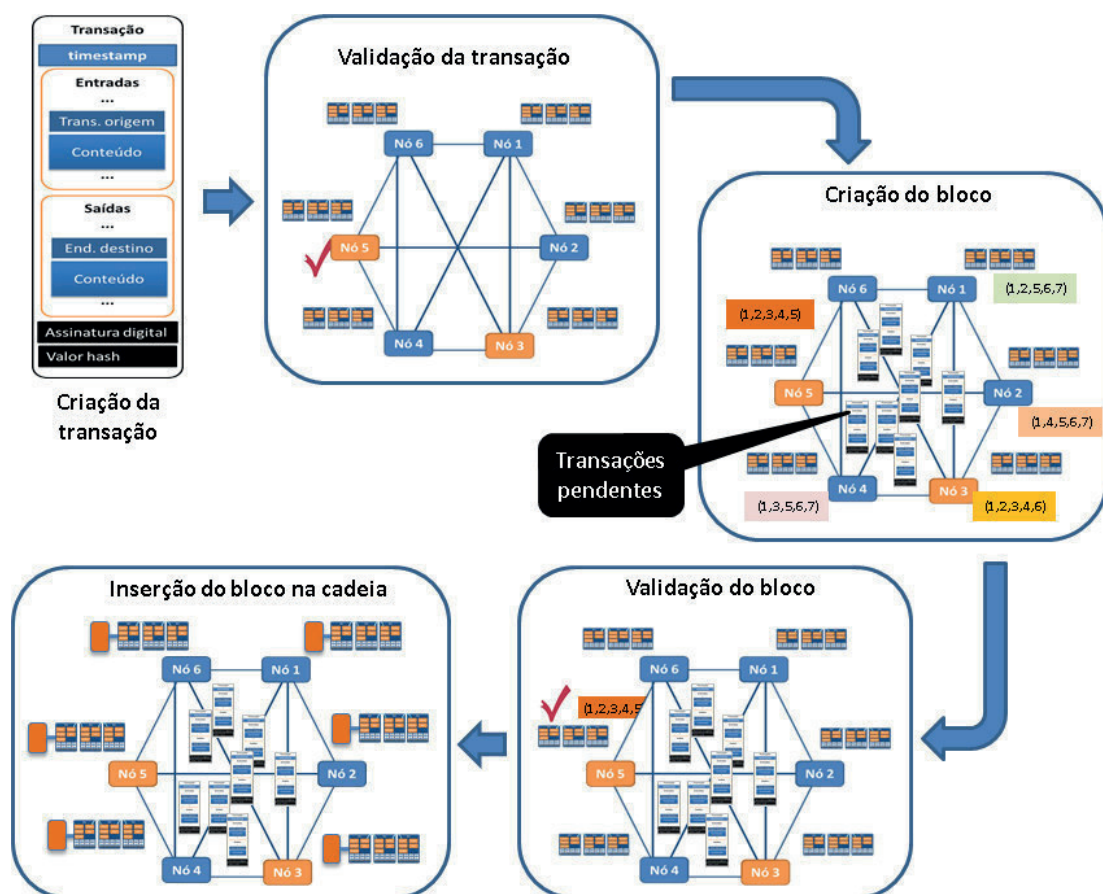


Figura 3.10. Transação, bloco, cadeia, consenso e rede de nós P2P.

### 3.3. Desenvolvimento de aplicações blockchain

Esta seção explica os seguintes aspectos relacionados ao desenvolvimento de aplicações blockchain: camadas de software de uma aplicação blockchain, arquitetura de uma aplicação típica, contratos inteligentes (*chaincodes*), blockchain como um conector de sistemas, o software cliente blockchain (carteira eletrônica - *eWallet*) e as decisões de projeto para aplicações blockchain, incluindo as decisões particulares ao projeto de um blockchain e aquelas próprias da aplicação.

O desenvolvimento de aplicações em software da tecnologia blockchain é dividido em três partes: a aplicação propriamente, o software cliente de usuário final (carteira eletrônica - *eWallet*) e as decisões de projeto de sistemas blockchain.

#### 3.3.1. A aplicação blockchain

A aplicação blockchain é abordada por três aspectos relacionados: as camadas tecnológicas, a arquitetura em módulos da aplicação e os contratos inteligentes.

##### 3.3.1.1. Camadas de software da tecnologia blockchain

Em geral, uma aplicação blockchain é composta por três camadas [8] ilustradas na Figura 3.11. A primeira camada é a do sistema distribuído que consiste na infraestrutura fundamental, responsável pela implementação do conceito de “*ledger* distribuída” e as funcionalidades necessárias para que ele possa ser utilizado, tais como métodos de consenso, armazenamento da *ledger* e protocolos de comunicação ponto a ponto. Esta camada é o que normalmente se chama de blockchain.

A segunda camada contém os serviços de apoio e infraestrutura, que viabilizam o desenvolvimento de aplicações robustas e seguras, relacionados à gestão de chaves criptográficas, integridade e confiabilidade de dados, disponibilidade de nós da rede P2P, rastreabilidade de transações, gestão de identidade, sigilo, privacidade, reputação, entre outros aspectos de segurança (de acordo com o nicho de aplicações preferencial da aplicação e do blockchain), sendo geralmente associada à camada de plataforma. Fazem parte desta camada os softwares que conduzem as transações e as plataformas de

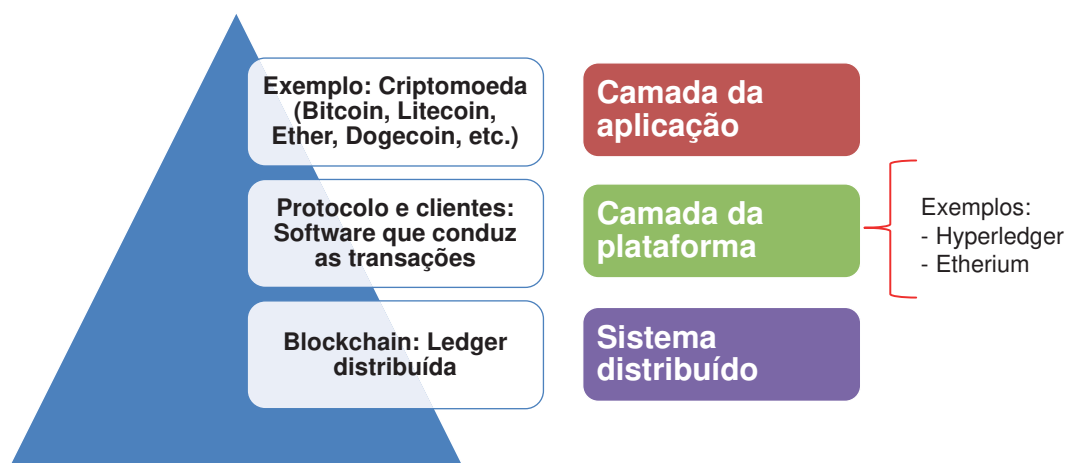


Figura 3.11. Camadas tecnológicas do blockchain.



desenvolvimento de aplicações, como Hyperledger [9] e Ethereum [10].

A terceira camada é a de aplicação, sendo composta não apenas pela lógica de negócios da aplicação, mais também pelos contratos inteligentes, como programas de computador que viabilizam a implementação, dentro de cada um dos nós da rede P2P, de parte das regras de negócio da aplicação. Os contratos inteligentes são discutidos adiante no texto. As criptomoedas (tais como o Bitcoin) são aplicações sobre plataformas blockchain e fazem parte desta camada. Esta visão simplificada em três camadas é geralmente elaborada em maior detalhe durante a implementação de sistemas sofisticados.

### 3.3.1.2. Estrutura de uma aplicação blockchain

Em geral, as aplicações blockchain possuem uma arquitetura de software com cinco módulos bem definidos [11], conforme ilustrado na Figura 3.12. O primeiro módulo é o cliente ou *front-end* de usuário final, geralmente associado aos aplicativos móveis e às interfaces web que implementam as funções de uma carteira eletrônica (*eWallet*).

O segundo módulo consiste da aplicação servidora, que possui regras de negócio e dados armazenados fora do blockchain, por meio de plataformas de software tradicionais e bases de dados comuns. Esta aplicação pode estar hospedada em computadores que não fazem parte da rede P2P, ou em nós da rede P2P, ou pode ainda ser dividida entre os nós da rede blockchain e uma plataforma de aplicações em nuvem, por exemplo.

O terceiro módulo é uma camada de integração entre a aplicação servidora (ou outros sistemas legados) e a aplicação blockchain. Esta camada pode ser disponibilizada por

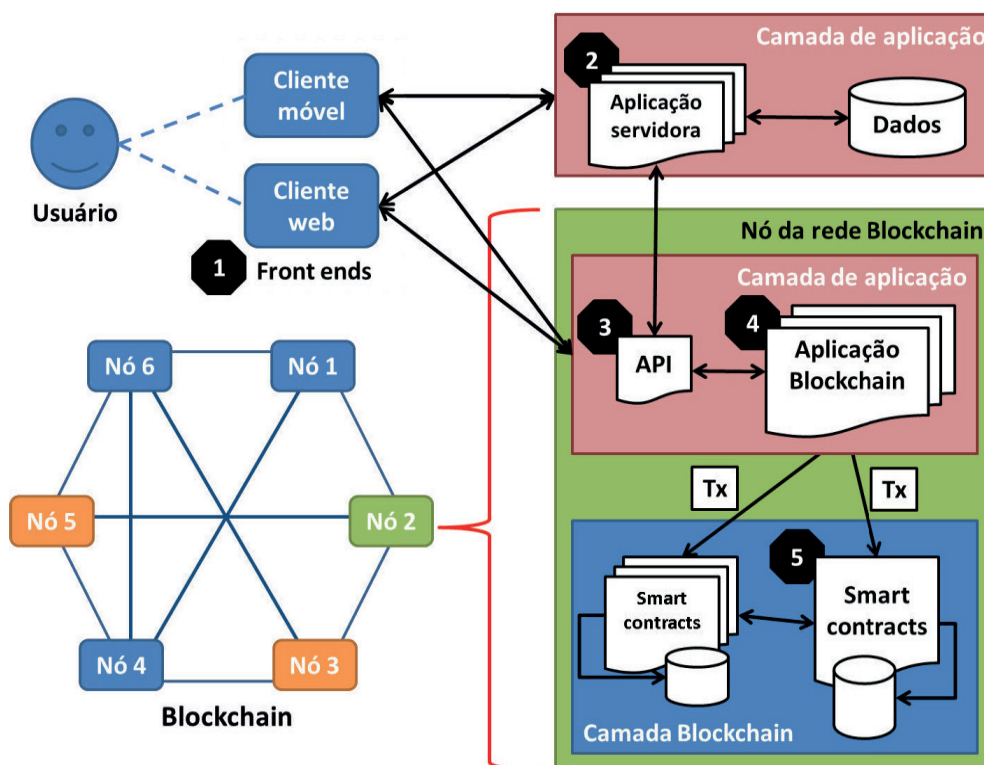


Figura 3.12. Estrutura de uma aplicação blockchain.

meio de uma interface de programação de aplicações (*Application programming Interface* - API), utilizando, por exemplo, uma arquitetura de microserviços.

O quarto módulo é a aplicação blockchain que manipula (por meio de uma API) a *ledger* distribuída. Este módulo pode estar localizado fisicamente em um nó da rede P2P e, em casos que necessitem de personalização extrema, pode até fazer parte do software que implementa as funções de nó da rede P2P, que processa transações e constrói blocos, estendendo o funcionamento deste componente geral do blockchain.

Finalmente, o quinto módulo é formado pelos contratos inteligentes como programas de computador implantados e executados em cada um dos nós da rede blockchain. Os contratos inteligentes o meio pelo qual as transações de semântica complexa (mais sofisticadas que, por exemplo, as transações de débito e crédito das criptomoedas) podem ser implementadas nas plataformas blockchain.

Finalmente, em arquiteturas de software complexas, o blockchain pode atuar como um conector de aplicações [11]. Uma aplicação insere dados, via transações, por um nó da rede P2P. Enquanto outra aplicação coleta, consulta ou recebe dados por outro nó da rede, conforme ilustrado pela Figura 3.13. Além disso, sistemas externos podem interagir com contratos inteligentes ou com nós da rede, simplificando a integração entre sistemas complexos, que são formados por sistemas de sistemas.

A arquitetura de software mostrada nesta seção reforça a ideia de que o blockchain pode ser entendido como um conector sofisticado de sistemas distribuídos grandes e complexos. Tais como, por exemplo, aqueles compostos por cadeias de valor longas ou redes de aglomerados de objetos inteligentes na internet das coisas.

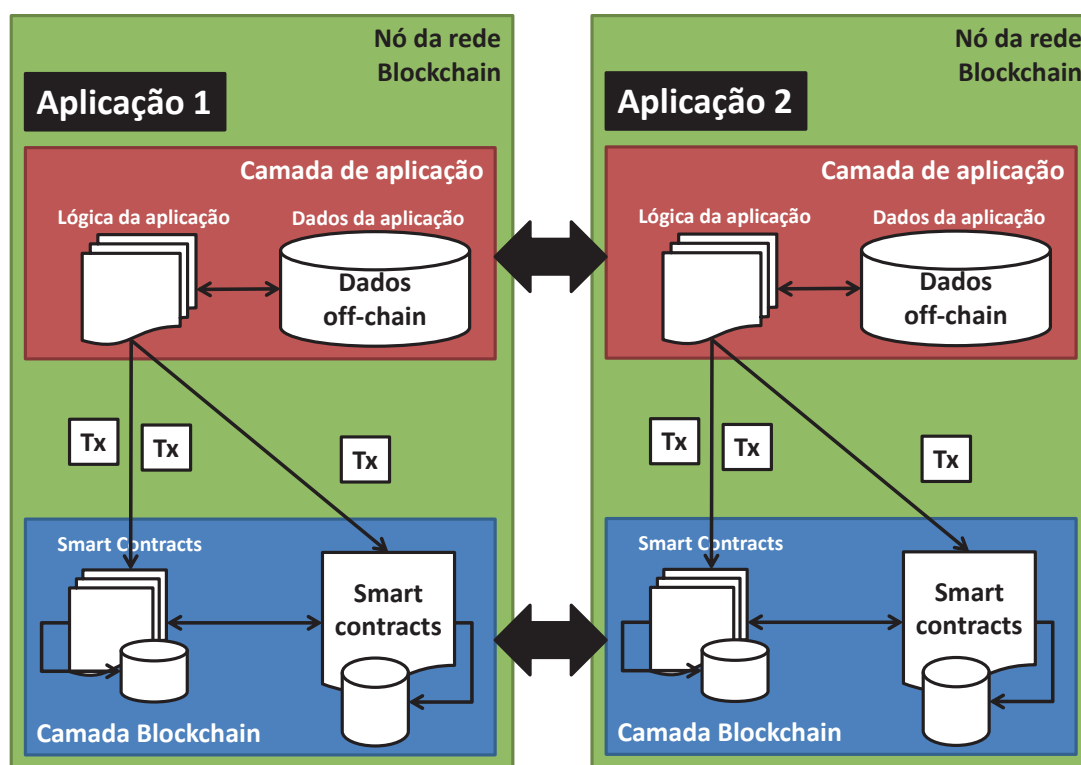


Figura 3.13. Blockchain como um conector de sistemas complexos.

### 3.3.1.3. *Smart contracts (chaincodes)*

A tecnologia blockchain passou por uma grande evolução com a possibilidade de uso dos contratos inteligentes em conjunto ao blockchain. Aplicações baseadas em contratos inteligentes são chamadas de *Decentralized Applications* ou DApps. Uma definição de contrato inteligente é a seguinte [5]: programas de computador seguros e imparáveis (ou ininterrompíveis) que representam acordos executáveis e exigíveis automaticamente.

Os contratos inteligentes resolvem questões que necessitam de acordos com mínima confiança entre as partes participantes de um sistema distribuído ([12], [13]). Conforme ilustrado na Figura 3.14, os contratos inteligentes são programas de computador (escritos em linguagens de programação gerais ou específicas) que podem ser corretamente executados por uma rede de nós mutuamente suspeitos de uma rede P2P, sem que seja necessária uma entidade externa confiável para mediação do acordo ([12], [13]). Diz-se que os nós da rede são mutuamente suspeitos por que eles não precisam confiar incondicionalmente uns nos outros, uma vez que podem ser competidores ou até mesmo adversários. Os programas distribuídos são ditos inteligentes por que, em princípio, podem funcionar autonomamente em benefício dos participantes da rede.

Idealmente, contratos inteligentes seguem o princípio de que “código é lei” (“*code is law*”), uma vez que não existiria necessidade de intermediários ou de árbitros para controlar ou influenciar a execução dos contratos inteligentes, pois eles seriam auto exigíveis em vez de legalmente exigíveis. Na prática, debate-se sobre a validade jurídica dos contratos inteligentes, expressos unicamente em códigos executáveis [14].

O programa executável (binário) do contrato inteligente é implantado e executado por todos os nós da rede P2P de um blockchain e sua execução correta é imposta (garantida) pelo mecanismos de consenso. O *chaincode* da plataforma Hyperledger, escrito na linguagem Go, e o *Contract* da *Ethereum Virtual Machine* (EVM), escrito na linguagem Solidity, são dois exemplos de contratos inteligentes disponíveis atualmente.

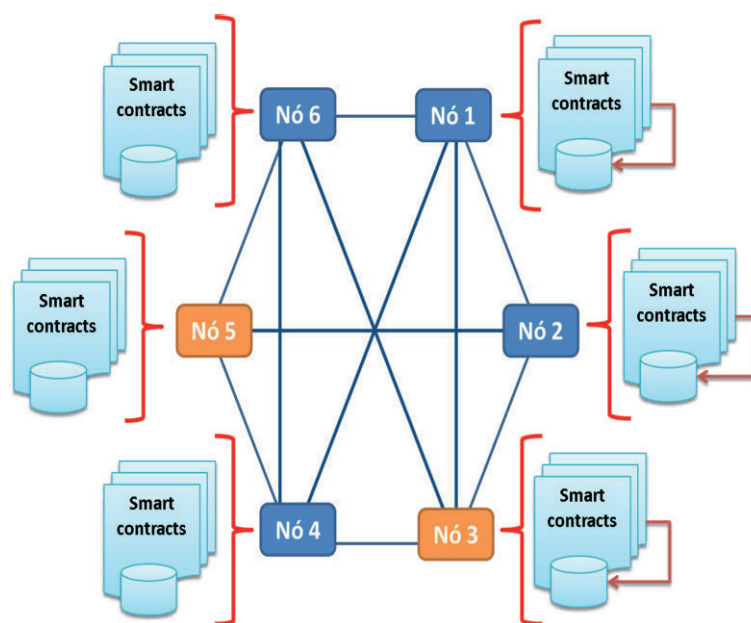


Figura 3.14. Contratos inteligentes (*smart contracts, chaincodes*).

### 3.3.2. O Software cliente blockchain

Também chamado de *eWallet* (carteira eletrônica) no jargão das criptomoedas (e do Bitcoin), o software cliente blockchain é a aplicação, para usuário final, mais comum com os blockchains atuais, uma vez que existem em quantidade muito maior que os nós da rede P2P [2], [8], [15], [16].

Uma *eWallet* típica implementa mecanismos para armazenamento seguro de chaves criptográficas, assinatura digital de transações, encriptação de transações e transmissão segura de dados [17], [18]. A função principal de uma *eWallet* é a gestão de uma coleção de chaves privadas para a manipulação correta dos ativos de valor da aplicação [17], [18]. Ativos associados a cada uma das chaves.

A *eWallet* é personalizada para lógica da aplicação e adota metáforas específicas do negócio, que não estão necessariamente relacionadas às criptomoedas. Comumente, a *eWallet* não precisa participar do consenso, uma vez que não é um nó da rede P2P, mas é crítica para a proteção dos dados da conta do usuário final. A Figura 3.15 mostra uma *eWallet* personalizada para aplicação blockchain específica, que foi desenvolvida pelo CPqD na plataforma Hyperledger [9]. Nesta aplicação, uma criptomoeda é associada à curtida em uma rede social, se tornando um recurso escasso e, por isso, valioso.

O blockchain oferece uma grande oportunidade para a popularização da criptografia assimétrica (de chave pública). Em geral, a complexidade de utilização destes sistemas criptográficos de chave pública pelos usuários finais tem sido a principal barreira para adoção em larga escala das criptomoedas e outros sistemas blockchain [17], [18].

Há inovações em usabilidade da gestão de chaves, boa parte delas está em buscar metáforas adequadas e abstrações claras para os ativos transacionados pela aplicação. As metáforas e abstrações devem favorecer o uso transparente da criptografia [17], [18]. Assim, cabe aos desenvolvedores de aplicações blockchain aplicar metáforas adequadas.

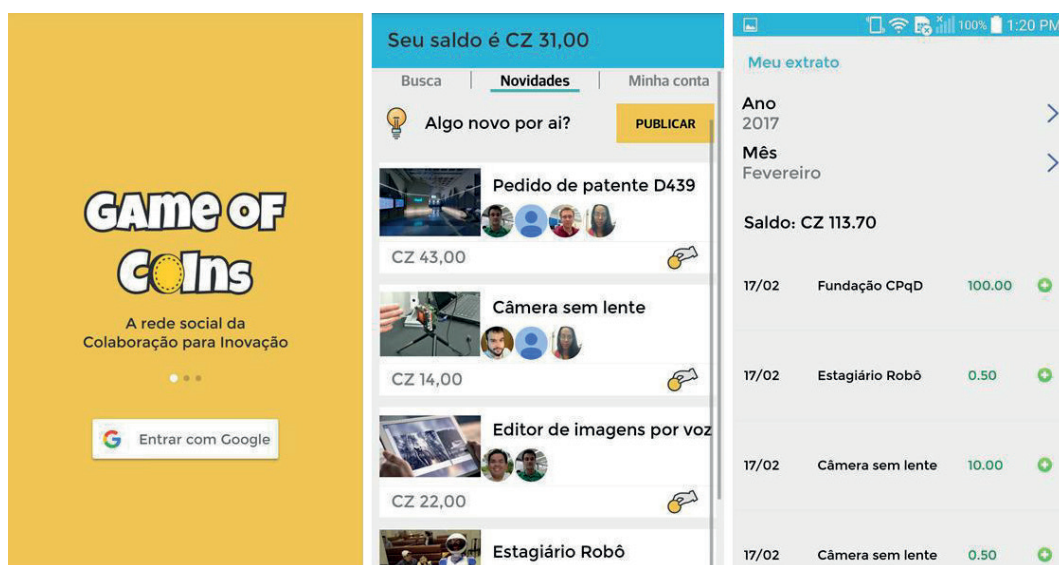


Figura 3.15. Exemplo de um software cliente blockchain (*eWallet*).

### 3.3.3. Decisões de projeto para aplicações blockchain

Os arquitetos, projetistas e desenvolvedores de aplicações baseadas na tecnologia blockchain devem tomar uma série de decisões de projeto relacionadas não apenas a arquitetura do blockchain, mas também a arquitetura da aplicação e como ela usa um blockchain [11]. Algumas decisões de projeto do blockchain afetam a velocidade de processamento de transações, tais como as seguintes:

- Tamanho do bloco: quantas transações fazem parte do bloco e quais os valores de máximo e de mínimo para esta quantidade;
- Transações fora do blockchain: transações sobre dados armazenados fora da *ledger*, mas que devem ser vinculadas às transações na *ledger*;
- Tamanho das transações: qual a estrutura de dados da transação e sobre quais dados as transações são aplicadas;
- Quantas assinaturas por transação: uma única assinatura do autor (origem) da transação, duas assinaturas de origem e de destino, ou até várias assinaturas de origem, de destino e de autorização;
- Protocolo P2P escalável, por exemplo, protocolos com mensagens leves e de baixa frequência, ou protocolos para atualização rápida de réplicas em redes P2P grandes.

Outra decisão de projeto está relacionada ao mecanismo de consenso. Existem várias opções disponíveis para mecanismo de consenso, algumas das mais conhecidas são as seguintes: prova de trabalho (o método adotado pelo Bitcoin), participação, consenso bizantino, e até nenhum (neste caso qualquer transação que entra na rede P2P é incluída na cadeia de blocos, o que pode resultar em inconsistências severas). Geralmente, os requisitos da aplicação exigem que algum método de consenso seja escolhido. Não utilizar métodos de consenso põe em dúvida a necessidade do blockchain.

Do lado da aplicação que usará o blockchain, as decisões de projeto mais importantes estão relacionadas aos seguintes assuntos:

- Escopo de armazenamento dos dados, que podem estar armazenados no blockchain (*in-chain*) ou fora do blockchain (*off-chain*), em bases de dados convencionais.
- Acesso da aplicação aos nós de rede P2P, que pode ser pública (com acesso irrestrito da aplicação aos nós), privada (os nós não podem ser acessados diretamente e nem livremente) ou híbrida. O caso híbrido ocorre, por exemplo, quando *eWallets* só de consulta têm acesso irrestrito aos nós, mas as aplicações que registram transações somente o fazem por nós específicos.
- Quantidade de *ledgers*, que pode ocorrer com *ledger* única ou com muitas *ledgers*;
- Validação de transações: só com dados da aplicação ou com dados internos e externos. Em qualquer caso, a validação da transação deve ser determinística.
- Permissionamento (de acesso) à *ledger*, que pode ser controlado ou livre. No caso controlado, o acesso também é identificado, autenticado e autorizado. Por isto, não há o anonimato que caracteriza o acesso livre.



### 3.3.4. Plataformas de desenvolvimento de aplicações blockchain

Atualmente, é possível desenvolver aplicações blockchain utilizando plataformas de desenvolvimento livres, com código aberto, ou proprietárias, porém predomina a tendência de desenvolvimento sobre plataformas de código aberto. Tais plataformas podem ser classificadas, quanto à utilização de uma *ledger* pública, em dois tipos:

- Os blockchains públicos com rede de acesso aberto, tais como, por exemplo, o Bitcoin [19] e a plataforma Ethereum [10];
- Os blockchains privados com redes de acesso restrito e associados a aplicações empresariais, tais como, por exemplo, as plataformas Ripple [20] e Hyperledger [9].

Esta seção descreve brevemente três plataformas blockchain: Bitcoin [19], Hyperledger [9] e Ethereum [10]. As logomarcas destas plataformas são mostradas na Figura 3.16.

#### 3.3.4.1. A plataforma Bitcoin

Os fundamentos do Bitcoin foram apresentados em 2008 [21] e posteriormente seu código foi disponibilizado, em 2009 [19]. Atualmente, vários desenvolvedores de aplicação, tais como fornecedores de carteira eletrônica, processamento de pagamentos, mineradores, empresas de seguros, dentre outros, utilizam esta plataforma.

O protocolo e o software Bitcoin são publicados abertamente. Atualmente, o código fonte da implementação de referência do protocolo Bitcoin é mantido como *Bitcoin Core* por uma comunidade de programadores [19]. Em geral, existem três formas de utilização da plataforma Bitcoin para desenvolvimento e integração de aplicações:

- Utilizando um provedor de serviço de pagamento, com o qual a aplicação é capaz de aceitar o Bitcoin como forma de pagamento. Muitas empresas disponibilizam APIs e ferramentas para integração de transações Bitcoin às aplicações;
- Utilizando uma API de integração com algum nó da rede blockchain do Bitcoin. Este API torna o acesso à rede mais simples para programadores comuns e é geralmente oferecida por um provedor de serviço;
- Escrevendo a própria API de integração e acessando o blockchain do Bitcoin por meio de um nó da rede P2P.



Figura 3.16. Da esquerda para a direita: Bitcoin, Ethereum e Hyperledger.



### 3.3.4.2. A plataforma Hyperledger

Hyperledger [9] é uma plataforma de código aberto, porém sem uma ledger pública, lançada em 2015 e voltada para ambientes empresariais e diferentes tipos de aplicação. Atualmente ela está na sua versão 1.0. Hyperledger também é um projeto colaborativo com a participação de várias empresas de grande porte e atualmente sua gestão é feita pela Linux Foundation, que faz a incubação de vários projetos associados à plataforma.

O desenvolvimento de aplicações com a plataforma Hyperledger é menos voltado para criptomoedas (embora possa ser utilizada para tal) e promove aplicações baseadas em sua tecnologia de contratos inteligentes, denominados de *chaincodes*, que podem ser escritos em linguagens de programação de uso geral, como Go e Java.

A plataforma Hyperledger possui uma arquitetura de referência baseada em dois grupos de componentes. O primeiro grupo contém os serviços de identidade, políticas, blockchain (consenso, ledger, protocolo P2P) e contratos inteligentes. O segundo grupo contém as interfaces de programação (APIs), kits de desenvolvimento (SDKs) e interfaces de linha de comando (CLI). Os componentes da arquitetura são interligados por chamadas de procedimento remoto.

Hyperledger Fabric foi a contribuição da IBM para o projeto Hyperledger, e atualmente é o núcleo do blockchain Hyperledger, com o conjunto de componentes fundamental, que incluem os serviços de gestão de identidade, autoridade certificadora, mecanismos de consenso, armazenamento da ledger, o protocolo P2P, gestão de *chaincodes*, etc.

### 3.3.4.3. A plataforma Ethereum

A plataforma Ethereum [10] foi proposta no final de 2013 como uma evolução da plataforma Bitcoin, e em 2015 foi tornada pública e de código aberto, já com o conceito de contratos inteligentes integrados à plataforma. A plataforma Ethereum oferece uma infraestrutura computacional com máquinas virtuais descentralizadas denominadas de *Ethereum Virtual Machines* (EVM), que executam contratos inteligentes usando uma criptomoeda própria denominada *ether*. A criptomoeda *ether* é a segunda colocada no mercado de criptomoedas, atrás apenas do Bitcoin. Ethereum é uma das plataformas mais utilizadas em projetos pilotos atualmente e também uma das tecnologias blockchain mais estudadas. A *Ethereum Foundation* é uma fundação sem fins lucrativos que promove a pesquisa, o desenvolvimento e a capacitação em Ethereum.

Na plataforma Ethereum, o software nó da rede P2P é chamado de cliente e o software para usuário final é chamado de *Wallet*. Clientes comuns são o *Geth* (implementação em Go de um cliente Ethereum) e o *Eth* (implementado em C++), o *Pyethapp* (em Python), e o *Parity*, desenvolvido pela empresa *EthCore*.

Os contratos inteligentes permitem o desenvolvimento de diferentes tipos de aplicação, além das criptomoedas. Há uma variedade grande de ferramentas de desenvolvimento de software para Ethereum. Por exemplo, os contratos inteligentes são escritos em linguagens de programação específicas, como Solidity e Serpent (derivação do Python). Existem ambientes de desenvolvimento (IDEs), como o Browser Solidity e o Ethereum Studio. Integração à aplicações externas pode ser feita com a ferramenta Web3.

### 3.3.5. Breve introdução à linguagem de programação Solidity

Ethereum também é um ambiente distribuído de execução de programas chamado de *Ethereum Virtual Machine* (EVM), que executa programas chamados contratos em benefício de usuários. Usuários enviam transações para a rede Ethereum a fim de criar novos contratos, invocar funções em contratos existentes, ou ainda para transferir criptomoedas para contratos ou outros usuários. Os contratos podem ser escritos em uma linguagem de programação chamada Solidity, descrita brevemente nesta seção.

Solidity é uma linguagem de programação completa, muito parecida com JavaScript, mas tipada estaticamente, e que contém as estruturas de controle de fluxo de execução essenciais, tais como laços, decisões e funções. Em Solidity, um contrato é uma construção parecida com uma classe das linguagens orientadas a objeto, tal como Java, possuindo até herança e polimorfismo de métodos. Em tendo propósito específico, Solidity tenta abstrair do programador particularidades do blockchain, porém ainda de modo bastante incompleto e sujeito a erros do programador, tais como os aspectos de armazenamento de dados na *ledger*, assim como as incertezas quanto à ordem de execução de contratos e distribuição de programas.

Os programas escritos em Solidity são traduzidos para um código intermediário (bytecode) que é de fato entendido pela EVM. Neste formato, os contratos são listas de funções associadas a um endereço e a uma transação de implantação. Uma característica interessante dos contratos Ethereum é que, além de receber valores na criptomoeda *ether*, eles também podem transferir valores para usuários ou até outros contratos.

Um usuário invoca funções em contratos por meio do envio de transações para os nós de rede Ethereum. Esta transação deve conter o valor da taxa de execução (remuneração para o operador do nó) proporcional ao esforço computacional para executar a função chamada, e pode conter um valor transferido do usuário (que chama a função) para o contrato. Se a função chamada não termina corretamente, uma exceção é disparada, a execução da função é interrompida, a taxa é paga mesmo assim e a transferência de valores, se houver, pode ser revertida, conforme as condições da chamada. Se a função chamada não é reconhecida, uma outra função anônima (sem nome) e paga (a função de *fallback*) é ativada por padrão. Se a transação não tiver fundos suficientes para a função de *fallback*, uma exceção é disparada, mas a taxa de execução é paga mesmo assim. Quando não é mais necessário, um contrato deve ser explicitamente desligado.

Os contratos em Solidity têm acesso a estruturas de dados implícitas, tais como a transação associada à execução corrente do contrato (*msg*), o bloco em que a transação está incluída (*block*), e o próprio contrato (*this*), que dá acesso ao saldo do contrato (*this.balance*). A partir da transação (*msg*), é possível saber o endereço do usuário origem a transação (*msg.sender*) e o valor transferido para o contrato (*msg.value*). Em geral, os campos de um contrato são armazenados implicitamente, sem necessidade de uma ação direta do programador. Outras informações específicas podem ser registrados na *ledger* por meio de eventos enviados pelo contrato à EVM. Por falta de espaço no texto, somente os elementos específicos da linguagem Solidity são introduzidos por meio do exemplo listado no Programa 3.1.

O exemplo do Programa 3.1 mostra um contrato, composto por campos e funções, que faz operações aritméticas simples (como adição, subtração e multiplicação) sobre dois

operandos e de forma gratuita. Já as operações menos simples, como a divisão e a resolução de uma equação linear, são pagas (a palavra reservada *payable* é usada) e o valor arrecadado vai para o saldo do contrato, de modo automático e transparente. A função de divisão aceita qualquer valor (maior que zero), enquanto a equação linear requer um preço mínimo (a construção *require* é usada para limitar o preço mínimo em wei). Quando o contrato é encerrado, o saldo dele é transferido para o endereço de criador, também de modo automático (com a construção *selfdestruct*).

Diversos elementos importantes podem ser observados no Programa 3.1. Na linha 06, observa-se o endereço do criador do contrato, que foi codificado a partir da chave pública (o tipo *address* é utilizado). A linha 10 mostra o endereço de origem da mensagem (transação) que implanta o contrato. A linha 19 faz o tratamento de exceção por reversão (*revert*). As linhas 18 e 24 mostram funções de acesso pago (*payable*), além do valor para execução do código. A linha 20 mostra como acessar o saldo do contrato (*this.balance*). As linhas 25 e 32 mostram como fixar pré-requisitos para execução de uma função (*require*). A linha 29 mostra a função de *fallback* executada se não houver outra (função) na chamada. Finalmente, a linha 33 mostra o mecanismo de autodestruição (encerramento) do contrato (*selfdestruct*).

**Programa 3.1. Um contrato Ethereum escrito na linguagem Solidity.**

```

01 pragma solidity ^0.4.13;
02
03 contract Equation {
04
05     uint256 public price = 1000000 wei;
06     address public creator;
07
08     event balance(uint256 b);
09
10     function Equation() payable { creator = msg.sender; }
11
12     function add(int256 x, int256 y) returns (int256) { return x + y; }
13
14     function sub(int256 x, int256 y) returns (int256) { return x - y; }
15
16     function mul(int256 x, int256 y) returns (int256) { return x * y; }
17
18     function div(int256 x, int256 y) payable returns (int256) {
19         if (y==0) revert();
20         balance(this.balance);
21         return x/y;
22     }
23
24     function linear(int x, int a, int b) payable returns (int) {
25         require(msg.value >= price);
26         return add(mul(a,x),b); // y = a*x + b
27     }
28
29     function() payable { price = price * 2; } // fallback
30
31     function kill(){
32         require(msg.sender == creator);
33         selfdestruct(msg.sender);
34     }
35 }

```

### 3.4. Segurança de sistemas blockchain

Em se tratando de sistemas complexos, os sistemas blockchain devem ter seus aspectos de segurança tratados de modo abrangente e sistêmico, prevenindo ataques de maneira estruturada e diferente da abordagem reativa, que responde apressadamente aos ataques midiáticos e às vulnerabilidades da moda. Esta seção trata a segurança de sistemas blockchain com complexidade crescente, iniciando com uma visão geral que é detalhada progressivamente, até a discussão de vulnerabilidades específicas em programas.

Esta seção mostra como as boas práticas de segurança de software já consagradas (tais como a gestão de identidades, o controle de acesso, a autenticação forte do usuário, o desenvolvimento seguro de sistemas, etc.) são aplicadas ao contexto de blockchain. Esta seção analisa três aspectos de segurança da tecnologia blockchain: segurança em camadas, privacidade e anonimato e gestão de chaves criptográficas.

#### 3.4.1. Camadas de segurança de um blockchain

A segurança em camadas, também chamada de defesa em profundidade, estabelece que as proteções devem ser aplicadas em camadas de defesa, de modo que a confiança não seja depositada em apenas um mecanismo de proteção. Muitas vezes, os mecanismos são diferentes, mas podem também ser redundantes (que fazem a mesma coisa, tem a mesma função). Proteções redundantes aumentam a capacidade de tolerância a falhas. As proteções redundantes não precisam ser idênticas. De fato, o uso de sistemas redundantes, mas heterogêneos, aumenta a diversidade das proteções.

O blockchain e as aplicações construídas com ele devem adotar a segurança em camadas. Há seis camadas de segurança a serem consideradas em uma aplicação

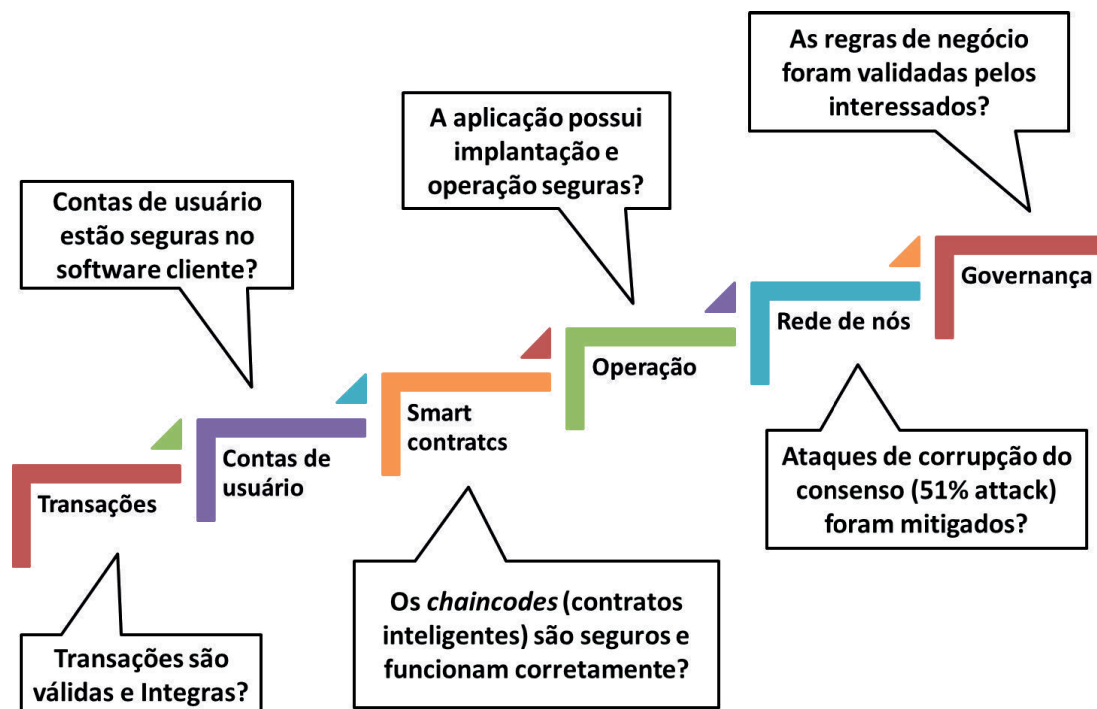


Figura 3.17. Camadas de segurança da tecnologia blockchain e suas aplicações.

blockchain: segurança da transação, segurança da conta do usuário, segurança do software cliente blockchain (*eWallet*), segurança da aplicação e seus *chaincodes*, segurança operacional e na implantação, segurança da rede de nós P2P e governança do blockchain. Estas camadas são o resultado da compilação de boas práticas presentes em livros texto da área [2], [8], [15], [16] e estão ilustradas na Figura 3.17 e a seguir:

- **Segurança da transação.** Requisito mínimo sem o qual o blockchain não faz sentido. O blockchain deve validar as transações com confiança e previsibilidade ao final do consenso. As proteções da transação são sintáticas e de estrutura e protegem tanto as transações quanto os blocos que as contém. Porém, estas proteções não impedem fraudes semânticas associadas à lógica da aplicação.
- **Segurança da conta de usuário.** A conta do usuário é geralmente gerenciada pelo próprio usuário em aplicativos de uso pessoal (*eWallets*). Muitas vezes, a proteção da conta do usuário é confundida com a do software cliente.
- **Segurança da aplicação e dos contratos (*chaincodes*).** Fazem parte desta camada as boas práticas de desenvolvimento seguro de software, incluindo a codificação segura de *smart contracts* e a definição de requisitos de segurança, avaliação de arquitetura e testes de segurança da aplicação.
- **Segurança de implantação e de operação da aplicação.** Fazem parte desta camada os testes de aceitação e homologação da aplicação e dos *chaincodes* antes da implantação em produção. Uma vez no ambiente de produção, a aplicação deve ser monitorada para detecção de anomalias de funcionamento e comportamento. Monitoramentos avançados podem até detectar fraudes.
- **Segurança da rede P2P de seus nós.** Nesta camada, os mecanismos de proteção tradicionais das redes de computadores (tais como sistemas de firewall, IDS, IPS, etc.) podem ser aplicados para proteção dos nós da rede P2P do blockchain. Além disso, proteções específicas devem ser aplicadas para a segurança do protocolo de comunicação e de consenso.
- **Governança da aplicação e do blockchain.** Esta camada abriga aquelas decisões sobre a arquitetura e o projeto do blockchain, discutidas na seção anterior, que afetam o funcionamento com segurança do blockchain, incluindo ainda os controles antifraude, auditoria, privacidade e até conformidade a normas e padrões.

Finalmente, vale observar que as questões de segurança descritas nas próximas seções têm soluções adequadas aos requisitos de aplicações específicas. Por exemplo, defeitos comuns em aplicações podem ser evitados pela adoção de boas práticas de desenvolvimento de sistemas. Vulnerabilidades em plataformas blockchain podem ser mitigadas pela adoção de boas práticas de desenvolvimento seguro de sistemas, tais como requisitos de segurança, padrões de codificação segura, verificações e testes de segurança. Privacidade e anonimato podem ser obtidas pela adoção de mecanismos de sigilo criptográfico das informações sensíveis, complementando políticas de controle de acesso aos dados.



### 3.4.1.1. Segurança da transação

A camada de proteção fundamental de qualquer blockchain é a segurança da transação, sendo um requisito mínimo sem o qual o blockchain não faz sentido. O blockchain deve validar as transações com confiança e previsibilidade ao final do protocolo de consenso, que confirma a finalidade e a imutabilidade de transação. Vide ilustração na Figura 3.18.

A segurança da transação oferece proteções sintática e estrutural para as transações, contribuindo para a segurança dos blocos que as contém. Porém, estas proteções não impedem fraudes semânticas associadas à lógica da aplicação. Todas as transações no blockchain são protegidas criptograficamente quanto à integridade e irrefutabilidade.

Funções de hash criptográfico são utilizadas na garantia de integridade das transações (cada transação tem um *hash*) e dos blocos (cada bloco tem um *hash*). Além disso, a ordem relativa das transações (um aspecto importante da integridade do bloco) é garantida pelas árvores Merkle. Ainda, a própria cadeia de blocos tem seus blocos encadeados pelos valores *hash* dos blocos (cada bloco contém o *hash* do bloco anterior). Estes relacionamentos reforçam a imutabilidade dos blocos já contidos no blockchain.

Assinaturas digitais e criptografia de curvas elípticas são utilizadas nas garantias de imutabilidade e irrefutabilidade da transação. Toda transação é assinada digitalmente pela chave privada de seu autor e, uma vez assinada, não pode mais ser negada. Se a transação contém uma outra transação de origem (para transferência de valores), esta dependência reforça a imutabilidade das transações já contidas no blockchain.

As transações são protegidas pelo consenso contra duplicação, uma vez que ele não permite registro duplo de mesma transação. Esta propriedade é importante para as criptomoedas, pois evita gastar o mesmo dinheiro duas vezes. Evitar o *double spending* significa que, na hipótese de haver duas transações duplicadas (com o mesmo valor de origem) pendentes, a primeira a entrar no blockchain é a válida e a outra é descartada.

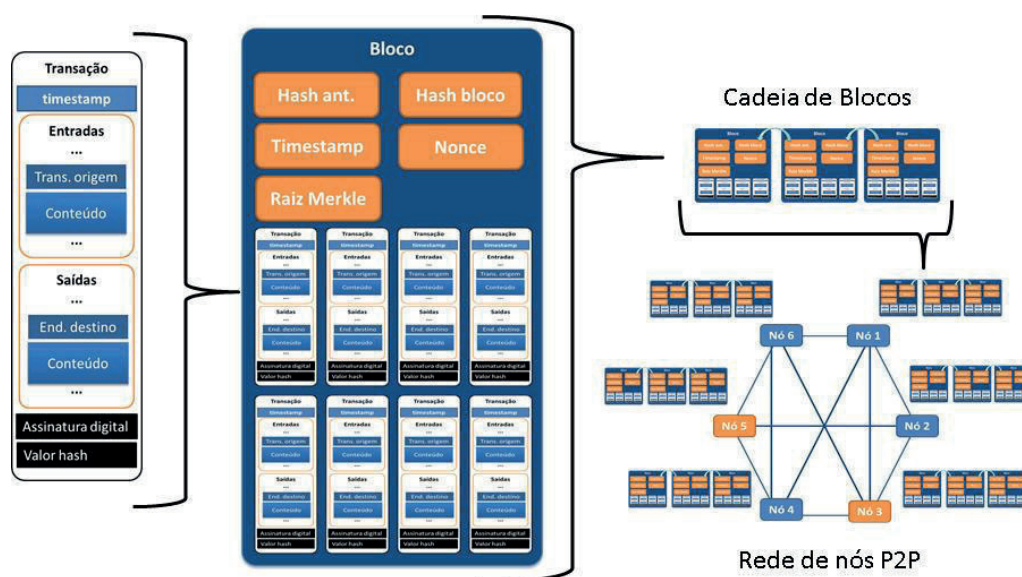


Figura 3.18. Todos os envolvidos na proteção da transação (bloco, cadeia e rede P2P).



### 3.4.1.2. Segurança da conta de usuário

A segunda camada de proteção do blockchain oferece segurança para a conta de usuário. Esta camada de segurança é influenciada por dois fatores: a conscientização dos usuários no uso seguro da tecnologia e a implementação correta dos mecanismos de segurança nos softwares de usuário final (*eWallets* móveis ou web).

Existe a necessidade de soluções de segurança com níveis adequados de usabilidade. Usabilidade para segurança não é um problema novo, mas foi intensificado pelas interfaces ricas e de interação natural dos dispositivos móveis modernos. Já em 1975, foi identificada a necessidade de aceitação psicológica de uma determinada solução de segurança por parte de seus usuários [22].

Tradicionalmente associada a uma “coleção de chaves privadas”, a *eWallet* blockchain tem a função principal de fazer a gestão destas chaves privadas para a manipulação correta dos ativos de valor da aplicação. Ativos associados a cada uma das chaves. Porém, a complexidade de utilização por usuários comuns tem sido a principal barreira para adoção das criptomoedas e outros sistemas baseados em blockchain. Isto se deve a complexidade inerente aos sistemas criptográficos de chave pública que não estão escondidos do usuário final em metáforas adequadas. Ainda assim, a tecnologia blockchain é uma oportunidade para a popularização da criptografia de chave pública.

Melhorias em usabilidade da gestão de chaves estão relacionadas à identificação de metáforas adequadas para o nicho da aplicação e abstrações claras para o usuário. As *eWallets* podem ser classificadas, quanto à gestão de chaves, nos seguintes tipos [18]:

- **Chaves em armazenamento local.** As chaves são armazenadas localmente no dispositivo do usuário, em um arquivo comum ou banco de dados.
- ***eWallets* (encriptadas) protegidas por senhas.** As chaves são armazenadas em arquivos locais e encriptadas por chaves derivadas de senhas escolhidas pelos usuários. Em implementações mais simples, só o acesso à *eWallet* é protegido por senha, sem encriptação de chaves, sendo menos seguras.
- **Armazenamento de chaves off-line.** As chaves são armazenadas em dispositivos de armazenamento portátil (por exemplo, tokens USB), sem poder de computação para fazer assinaturas digitais, acessadas (copiadas) quando necessárias.
- **Armazenamento de chaves segregado (*air gapped*).** As chaves são armazenadas em dispositivos secundários com poder de computação, que geram, assinam e exportam transações. Eles não são conectados a rede nem a outros computadores, e as transações são transportadas entre dispositivos em mídias removíveis.
- **Chaves derivadas de senhas.** Uma chave (ou um par de chaves) é derivada a partir de uma senha escolhida pelo usuário por um método determinístico (por exemplo, PBKDF2). Uma desvantagem é que outra chave requer uma senha diferente.
- **Carteiras hospedadas.** As chaves são hospedadas em sites de serviços de terceiros supostamente confiáveis (similares aos internet bankings). Estes sites atuam como intermediários poderosos que, de fato, controlam as transações.

### 3.4.1.3. Segurança da aplicação e dos *chaincodes*

A terceira camada de proteção de um blockchain contempla a segurança da aplicação e dos *chaincodes*. Fazem parte desta camada as boas práticas de desenvolvimento seguro de software, incluindo (mas não somente) a codificação segura de *smart contracts*, a definição de requisitos de segurança, avaliação de arquitetura e os testes de segurança da aplicação.

De acordo com McGraw [23], segurança de software trata, de modo geral, os aspectos de segurança na construção de sistemas de software, desde o levantamento dos requisitos de segurança até a escolha dos mecanismos de segurança. Inda, para McGraw [23], o software seguro é aquele que contempla, com um grau alto de confiança justificada, mas não com certeza absoluta, as propriedades explícitas de segurança e de funções de segurança, incluindo todas aquelas (funções e propriedades) necessárias para o uso pretendido do software. Assim, o software seguro é resultado de um processo de desenvolvimento capaz de produzir versões com pouquíssimos defeitos, custo de construção relativo baixo, custo de manutenção muito baixo e boa reputação.

Defeitos de software são geralmente convertidos em vulnerabilidades e explorados em ataques. Por isto, um software considerado seguro deve ter pouquíssimos defeitos ou, em termos práticos, apresentar uma taxa de defeitos por linha de código bastante baixa. A redução progressiva da taxa de defeitos só é conseguida com um processo de construção de software bem definido, estável e maduro.

Prover a segurança das aplicações blockchain não é um objetivo fácil de ser alcançado, dada a complexidade da tecnologia envolvida. Facilmente, uma aplicação atinge dezenas de milhares de linhas de código, que contêm uma grande quantidade de defeitos latentes. Alguns desses defeitos têm impacto direto em segurança, podendo acarretar desde a indisponibilidade da aplicação, até o comprometimento do nó da rede P2P ou da *eWallet* por um atacante. Além disto, há casos de vulnerabilidades que independem de implementação defeituosa, pois foram projetadas de maneira insegura.

Assim, a segurança deve ser considerada em todas as fases do ciclo de desenvolvimento de software blockchain, em vez de se tentar embutir segurança, somente depois que ele estiver em produção. Normalmente, esta última abordagem não é efetiva por questões de custo e nem eficaz, pois pode haver funcionalidades ortogonais à segurança e, por isso, sistêmicas. As vulnerabilidades podem ser agrupadas nas seguintes categorias:

- Projeto – independente de quão perfeita seja a implementação do sistema, uma decisão equivocada sobre algum aspecto da aplicação pode levar a fraquezas inerentes ao projeto e, por isto, sempre presentes. Exemplos incluem políticas de controle de acesso inadequadas e vulnerabilidades semânticas que facilitam fraudes.
- Implementação – essas vulnerabilidades resultam da não aplicação de técnicas seguras de programação. Como exemplos, podem ser citadas a validação inadequada de dados de entradas e a falta de tratamento de erros na execução.
- Configuração – ocorrem quando os componentes do software são configurados de maneira a deixar fraquezas que podem ser exploradas por um usuário malicioso. Por

exemplo, considera-se o uso de senhas padrão e a definição de um parâmetro que não limite os recursos alocados por um usuário.

- Operação – resultam da falta de procedimentos bem definidos para a realização de tarefas relacionadas à segurança e também à operação segura do sistema.

Várias boas práticas de segurança genéricas podem ser usadas no desenvolvimento de aplicações blockchain seguras. A Figura 3.19, mostra onde, em uma iteração do ciclo de vida de desenvolvimento, cada prática pode ser aplicada e quais os entregáveis relacionados a cada uma delas. Conforme ilustrado, na concepção do software, as ameaças são modeladas e os casos de abuso são descritos. Estes dois itens, juntamente com normas e padrões, servem de subsídio para o estabelecimento dos requisitos de segurança. Uma análise dos riscos associados às ameaças é usada na priorização de soluções de segurança. A revisão externa (realizada, por exemplo, por especialistas em segurança) pode validar o trabalho da equipe de projeto antes de iniciar a codificação.

No plano de testes de segurança, o foco deve ser dirigido às ameaças de maior risco. Ferramentas automáticas para análise estática de programas podem ser usadas para acelerar a verificação de código fonte e identificar erros comuns. A segurança do protótipo (ou das primeiras versões liberadas para testes) é avaliada em testes de segurança funcional e testes de intrusão. O risco residual ainda presente na aplicação pode ser reavaliado com base nos testes.

Finalmente, quando em produção, o software está exposto aos ataques. Ataques são inevitáveis, mas oferecem informação para realimentar outra iteração do ciclo de identificação de ameaças, estabelecimento de requisitos, priorização baseada em riscos, implementação de proteções, testes e implantação da próxima versão da aplicação.

Os maiores desafios de segurança do software blockchain estão relacionados à codificação segura de *smart contracts*, à definição de requisitos de segurança, à avaliação de arquitetura e aos testes de segurança. A Figura 3.19 representa os entregáveis do desenvolvimento de software seguro organizados em 3 grupos. O primeiro, em azul, é voltado para os requisitos. O segundo, em laranja, é voltado às revisões e inspeções. O terceiro, em vermelho, é voltado aos testes de segurança.

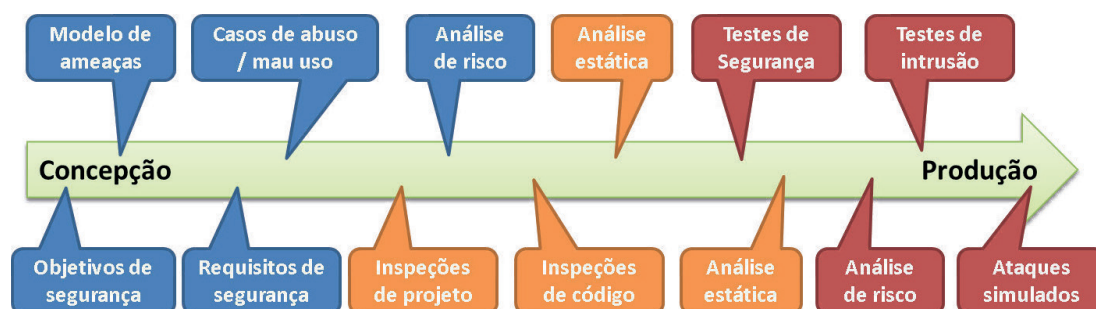


Figura 3.19. Entregáveis de desenvolvimento de software blockchain seguro.

#### 3.4.1.4. Segurança de implantação e de operação da aplicação

A quarta camada de proteção de um blockchain atende à segurança de implantação e de operação da aplicação. Fazem parte desta camada os testes de aceitação e homologação da aplicação e dos *chaincodes* antes de implantação em produção. Uma vez no ambiente de produção, a aplicação deve ser monitorada para detecção de anomalias de funcionamento e comportamento. Monitoramentos avançados podem detectar fraudes.

A metodologia de desenvolvimento de sistemas seguros não é capaz de garantir sozinha a proteção de sistemas em produção, quando eles operam em ambientes sem controle das versões, sem gestão das configurações operacionais e sem controle de mudanças. Um sistema de controle de versão (manual ou automatizado) é imprescindível para garantia de rastreabilidade dos requisitos de segurança em serviços e destes em mecanismos de segurança. Além disto, o controle de versão torna possível o rastro de defeitos ocorridos em produção até as correções em determinadas versões de programas.

A segurança dos sistemas não é garantida somente pela correção do software. Um sistema operacional seguro é uma combinação do software e das configurações do ambiente de execução. Para rastrear os controles de segurança e conhecer as vulnerabilidades presentes nos sistemas em produção, é importante o conhecimento das configurações dos equipamentos e dos sistemas. Além disso, é necessária a gestão das configurações em relação ao versionamento, registro de alterações, permissões de acesso, vulnerabilidades conhecidas e recuperação de incidentes.

Durante a implantação de mudanças nos sistemas, para evitar interrupções desnecessárias ou imprevistas, deve-se realizar um processo preparatório de avaliação e planejamento da mudança. A mudança deve ser documentada, autorizada, homologada e reversível se necessário.

Outro aspecto importante é a geração de *builds* para distribuição de versões de implantação do software. Os compiladores, ligadores (*linkers*), geradores de código, ofuscadores e otimizadores de código devem ser capazes de capturar a intenção do programador, produzindo binários executáveis corretos tanto sintaticamente quanto semanticamente, e não devem cancelar e nem desfazer controles de segurança presentes do código fonte.

Finalmente, durante a operação, é muito importante considerar o balanceamento de carga entre os nós da rede, no que se refere ao ponto (nó) de entrada de transações na rede P2P. Se muitas transações entram na rede pelo mesmo nó, este poderá ficar sobrecarregado, dificultando e, em situações extremas e ambientes restritos, até inibindo a convergência do consenso. A disponibilidade dos nós da rede deve ser considerada também quanto às proteções contra os ataques de negação de serviço e é tratada na próxima camada.

### 3.4.1.5. Segurança da rede P2P

A quinta camada de proteção do blockchain cobre a segurança da rede P2P de seus nós. Nesta camada, proteções específicas devem ser aplicadas para a segurança dos protocolos de comunicação P2P e de consenso. Ainda, deve ser observada a quantidade mínima necessária de nós disponíveis para garantir o consenso.

A proteção específica da rede P2P do blockchain deve fazer parte da proteção da rede de comunicação convencional utilizada pela aplicação. Tipicamente, trata-se de uma rede IP, por isto, os mecanismos de proteção tradicionais das redes IP (tais como sistemas de *firewall*, IDS, IPS, etc.) podem ser aplicados para proteção dos nós da rede P2P do blockchain.

Um firewall é uma combinação de hardware e software que isola a rede interna de uma organização da Internet em geral [24]. Em linhas gerais, um sistema de firewall atende a três condições [24]: todo o tráfego de fora para dentro, e vice-versa, passa por um firewall, somente o tráfego autorizado, como definido pela política de segurança local, poderá passar, e o próprio firewall é imune a intrusões.

Grosso modo, os sistemas de firewalls podem ser classificados em três categorias [24]: os filtros de pacotes tradicionais, os filtros de estado e os gateways de aplicação. Para detectar muitos tipos de ataque, deve ser executada uma inspeção profunda de pacote.

Um dispositivo que alerta quando observa tráfegos potencialmente mal-intencionados é chamado de sistema de detecção de invasão (*Intrusion Detection System – IDS*). Um dispositivo que filtra o tráfego suspeito é chamado de sistema de prevenção de invasão (*Intrusion Prevention System – IPS*). Um IDS pode ser usado para detectar uma série de tipos de ataques, incluindo [24]: mapeamento de rede (provindo, por exemplo, de nmap), varreduras de porta, varreduras de pilha TCP, ataques de DoS, ataques de inundação de largura de banda, *worms* e vírus, ataques às vulnerabilidade de sistema operacional e ataques às vulnerabilidade de aplicações.

Adicionalmente ao sistema de firewall que protege o ambiente de rede, uma plataforma blockchain deve disponibilizar mecanismos similares adicionais, chamados de firewalls de aplicação (*Web Application Firewall – WAF*), a fim de proteger as APIs disponíveis externamente (para integração de sistemas) contra ataques específicos. Além disso, há ataques DoS específicos contra blockchain e criptomoedas e que são discutidos nas próximas seções.

Finalmente, em termos da segurança da comunicação entre os nós da rede P2P e os clientes, observa-se que o protocolo HTTP sobre TLS (HTTPS) tem sido amplamente utilizado nas aplicações com características de web e nuvem (*cloud*). Além das configurações necessárias (tanto em softwares clientes quanto em servidores) para utilização do TLS, existe a necessidade de um módulo de autoridade certificadora (CA) a fim de emitir certificados digitais em quantidade suficiente para comunicação segura com TLS entre nós da rede P2P e uma grande quantidade de aplicações *eWallets*. No caso das blockchains fechadas, o software cliente deve ser plenamente identificado e autenticado, utilizando para tal um certificado específico.



### 3.4.1.6. Governança da aplicação e do blockchain

A sexta camada de segurança se refere à governança da aplicação e do blockchain. Esta camada abriga aquelas decisões sobre a estrutura e projeto do blockchain, discutidas anteriormente, que afetam o funcionamento com segurança, incluindo ainda os controles antifraude, auditoria, privacidade e até conformidade a normas e padrões específicos do nicho de aplicação.

Apenas a utilização de tecnologias de segurança pode não ser suficiente para a proteção de blockchain. Por exemplo, a criptografia forte não protege contra senhas fracas, assim como TLS não protege contra injeção de SQL ou transbordo de buffer no código da aplicação. Daí a necessidade de um Sistema de Gestão de Segurança da Informação (SGSI) capaz de oferecer um conjunto coerente de políticas, processos, práticas e procedimentos para gerenciar os riscos sobre ativos de valor tratados pelo blockchain.

Governança de segurança é um arcabouço normativo que fornece supervisão, prestação de contas, e conformidade. A gestão de segurança incorpora as atividades processuais e administrativas necessárias para apoiar e proteger informações e ativos empresariais manipulados pela aplicação blockchain. A gestão de segurança deve abordar as questões a partir de pontos de vista estratégico (políticas), tático (processos e práticas) e operacional (procedimentos e controles específicos).

A asseguarção da aplicação (um aspecto da garantida de qualidade) oferece as garantias de confiança tanto no funcionamento correto da aplicação blockchain, quanto nas proteções adequadas contra erros e ataques. A asseguarção é obtida com aderência a normas (de segurança), monitoramento, inspeções e testes contínuos. A asseguarção é essencial em plataformas blockchain que transacionam valores, pois oferece os meios para determinar se os requisitos de segurança foram atendidos a contento.

Os aspectos das aplicações blockchain geralmente sujeitos aos controles de garantia de qualidade e de segurança são os seguintes: se a funcionalidade está presente na aplicação e foi implantada corretamente, se existe proteção suficiente contra erros não intencionais, e se existe resistência suficiente contra invasões e outros ataques.

Um aspecto importante da governança de aplicações blockchain é a asseguarção de confiança nas organizações autônomas descentralizadas (*Decentralized Autonomous Organization* – DAO). Uma DAO é um agente autônomo de software formado por *smart contracts* em um blockchain. O termo recebeu notoriedade após o famoso ataque DAO [25] contra o blockchain Ethereum, explicado na seção 3.5.3.

A autonomia requerida por uma DAO deve ser merecida e supervisionada. Por isto, uma DAO deve estar sujeita a testes de aceitação e homologação antes de ser implantada em produção. Além disso, monitoramento e detecção de anomalias devem ser feitas durante o funcionamento da DAO. Finalmente, boas práticas de desenvolvimento seguro e aceitação de software devem ser seguidas a fim de evitar que uma DAO com código “*spaghetti*” funcione com excesso de privilégios, facilitando fraudes.

### 3.4.2. Privacidade e anonimato no blockchain

No blockchain tradicional, derivado da criptomoeda Bitcoin, a privacidade é limitada por dois aspectos. O primeiro é o pseudo-anonimato da transação e o segundo é o fato de que todas as transações estão em claro, isto é, sem criptografia para sigilo. Usa-se o termo pseudo-anonimato em vez de anonimato verdadeiro por que a análise das correlações entre transações, os endereços de destino e outros metadados derivados da lógica da aplicação podem facilitar a revelação da identidade do usuário.

Por exemplo [1], no Bitcoin, a partir da análise das transações de origem e dos endereços de destino das transações, é possível identificar as movimentações financeiras entre endereços específicos, reconhecendo padrões de relacionamento entre usuários do Bitcoin. Este método de mapeamento das entidades que transacionam Bitcoins e seus padrões de relacionamento foi estudado na prática [1] e está ilustrado na Figura 3.20.

A exploração de vulnerabilidades em carteiras eletrônicas, pode tanto facilitar o roubo de criptomoedas, como também revelar a identidade do usuário. Por exemplo [26], em *eWallets* de Bitcoin que utilizam o algoritmo criptográfico ECDSA para assinar transações, cada assinatura requer um número aleatório único e imprevisível. Porém, defeitos de segurança em algumas destas *eWallets* (na geração e utilização de números pseudoaleatórios ruins) podem resultar na revelação de chave privada a partir da geração de assinaturas digitais repetidas, facilitando o roubo de Bitcoins e o rastreamento de transações assinadas com estas chaves inseguras. Na Figura 3.21, a chave privada descoberta com nonce duplicado (vermelho) faz transferências para endereços de usuários maliciosos: um salto (amarelo) e dois saltos (azul) de distancia.

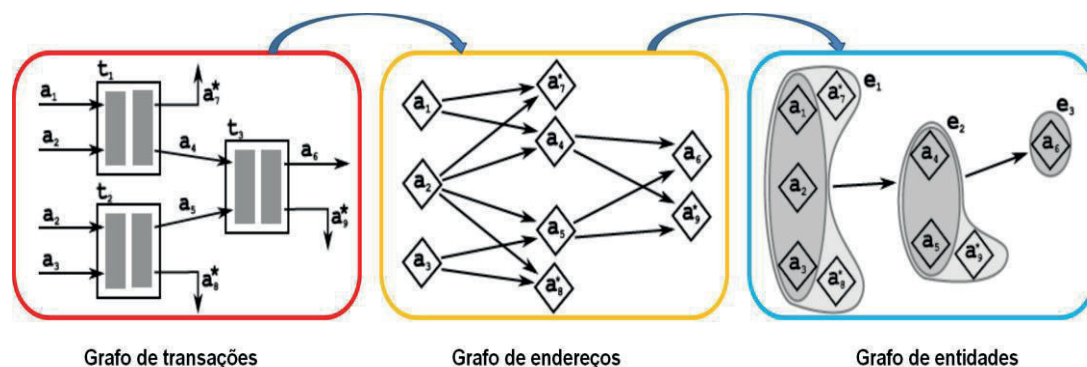


Figura 3.20. Mapeamento de entidades que transacionam Bitcoins (adaptado de [1]).

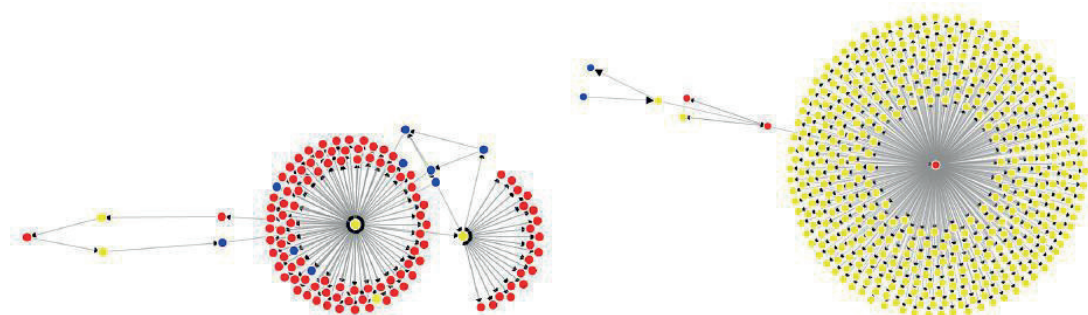


Figura 3.21. Transações que duplicam nonces na assinatura (adaptado de [26]).

### 3.4.3. Gestão de chaves criptográficas em clientes blockchain

A Seção 3.4.1.2, que trata da segurança da conta do usuário, estabeleceu que a principal função das carteiras eletrônicas blockchain é a gestão de chaves criptográficas. Esta seção detalha este aspecto em particular.

Estudos recentes [18] analisaram como as *eWallets* fazem a gestão de suas chaves criptográficas. As estratégias ou técnicas de gestão de chaves detalhadas a seguir são comparadas na Figura 3.22 em relação ao tipo de *eWallet* que as contempla. Uma vez que as *eWallets* são muito associadas às criptomoedas, a figura também compara as *eWallets* com dinheiro vivo (em papel moeda) e Internet banking.

Na Figura 3.22, um critério pode ser atendido totalmente (círculo cheio), parcialmente (círculo semicheio) ou não ser atendido (círculo vazio). No caso das *eWallets* hospedadas, as chaves podem estar on-line e prontas para acesso (*hot*), off-line com algum atraso no acesso (*cold*), ou algum meio termo. O gráfico de barras indica quais *eWallets* estão mais aderentes aos requisitos de proteção de chaves. As estratégias de gestão de chaves são descritas a seguir [18]:

- **Resistência a *malwares*.** As *eWallets* armazenadas em dispositivos conectados à internet ou com poder de computação são vulneráveis aos softwares maliciosos. Se a transação exige a cópia da chave para um destes dispositivos, o requisito é atendido em parte.
- **Chaves armazenadas off-line.** Chaves usadas poucas vezes (frequência baixa) não devem estar armazenadas em dispositivos ligados à Internet. Mas se estiverem armazenadas nestas condições, então devem estar protegidas por senha.
- **Sem terceiro confiável.** Ausência de uma terceira entidade (mediador, atravessador)

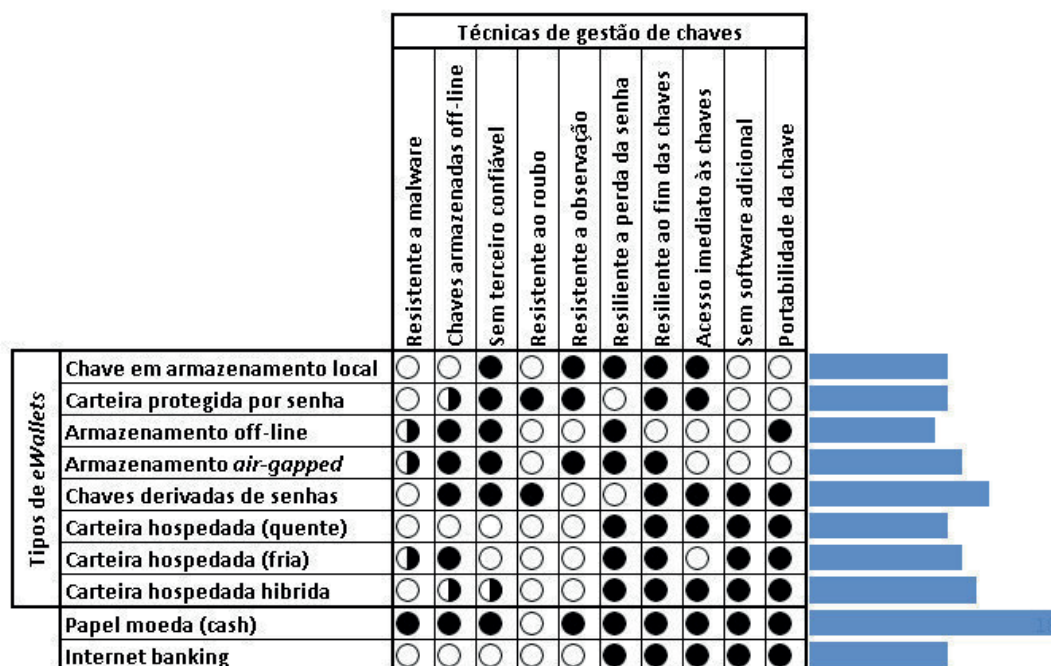


Figura 3.22. Tipos de *eWallets* e como fazem gestão de chaves (adaptada de [18]).

que seja responsável ou tenha autoridade sobre a assinatura da transação.

- **Resistência ao roubo.** O dispositivo onde a chave está armazenada pode ser roubado. Se há proteção por senha, a proteção é apenas parcial.
- **Resistência à observação.** Em um sentido amplo, desde monitoramento de teclado, captura de códigos QR, captura de telas, *shoulder surfing*, entre outros meios eletrônicos e físicos de observação.
- **Resiliência à perda da senha.** Se uma senha é usada como critério de acesso à chave de assinatura, então a perda (ou esquecimento) da senha, pode resultar em indisponibilidade da chave de assinatura de transações, se não houver um mecanismo de recuperação de senhas.
- **Resiliência ao esgotamento de chaves.** Mantém acesso aos ativos mesmo que chaves novas não sejam mais criadas e só haja chaves antigas.
- **Acesso imediato.** Acesso imediato à chave de assinatura e acesso direto às transações, sem a necessidade de acessos aos armazenamentos externos ou aos dispositivos secundários.
- **Sem software adicional.** As *eWallets* podem necessitar de instalação de software adicional no sistema/dispositivo do usuário, ou software não é disponível em várias plataformas, ou está disponível independente de plataforma (ex.: navegadores web).
- **Portabilidade entre dispositivos.** Permite o compartilhamento de chaves entre vários dispositivos do usuário com pouca (ou nenhuma) configuração, mesmo em casos excepcionais (chaves antigas).

Os gráficos da Figura 3.23 (adaptados de [17]) mostram cinco *eWallets* Bitcoin populares e a percepção de seus usuários sobre as características de segurança: se a carteira é encriptada, se ela tem cópia de segurança (backup) e se possui ainda um backup extra.

Sobre a encriptação da carteira, observa-se que muitos usuários conseguem dizer que usam encriptação, porém há uma grande parcela que não sabe. Sobre a existência de backup, a maioria expressiva dos usuários afirma possuir backup (exceto em um caso). Ainda, em relação a um segundo backup, a parcela de usuários que usam está equilibrada com a parte que não usa, havendo pouca dúvida entre os usuários.

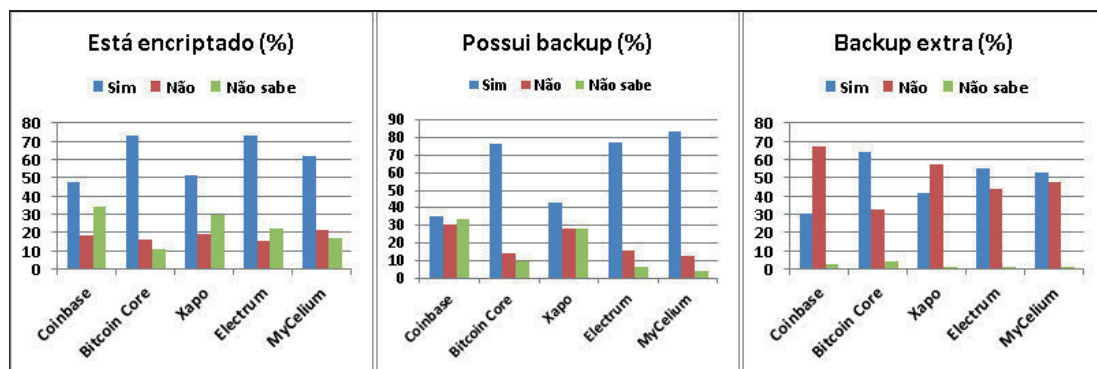


Figura 3.23. Usuários encriptam e fazem backup das suas carteiras (adaptada de [17]).



### 3.5. Codificação segura para blockchain

Programas de computador são gerados e executados por outros programas, em uma pilha de execução, que vai deste o sistema operacional, o compilador e as bibliotecas de apoio, até os controladores de dispositivos e softwares embarcados no hardware, incluindo firmwares. Vulnerabilidades podem ser exploradas em qualquer nível desta pilha. Assim, seria necessário verificar a correção e a integridade de toda a pilha para então obter a confiança plena no funcionamento do programa que está no topo da pilha.

A produção de programas seguros nas plataformas blockchain atuais não é uma tarefa fácil, pois depende de vários fatores, entre eles a integração adequada de diversos componentes de software em níveis de abstração e camadas de software relacionadas. Foge ao escopo deste texto avaliar a programação segura de todos os componentes envolvidos em uma arquitetura de software blockchain. Porém, tentar-se-á oferecer uma visão abrangente, na medida do possível.

No que se refere à programação dos softwares clientes blockchain (*eWallets*) e das aplicações servidoras, que são geralmente escritos em linguagens de programação de propósito geral e tecnologias de mercado, as boas práticas conhecidas para codificação segura podem ser empregadas. Esta seção não detalha estas técnicas gerais. Para ilustração, citam-se aqui o OWASP Top 10 [27] e o SANS top 25 [28] como exemplos de catálogos de vulnerabilidades em software que incluem as técnicas de programação segura necessárias para evita-las. Ainda, citam-se os guias de programação segura para as linguagens de programação Java [29] e C [30].

Esta seção detalha a programação segura das plataformas blockchain de acordo com três aspectos. Primeiro, os defeitos de programação insegura comuns em aplicações blockchain, tais como a falta de verificação de certificados digitais, o transbordamento de buffer e a configuração incorreta de criptografia, entre outros problemas. Em seguida, serão analisados exemplos de código fonte (escritos em Solidity para a plataforma Ethereum) com vulnerabilidades associadas à programação insegura de *chaincodes*, incluindo o ataque DAO.

#### 3.5.1. Defeitos e vulnerabilidades comuns em sistemas blockchain

Um estudo bastante recente [31] identificou os defeitos de software (bugs) mais comuns em sistemas blockchain, que são listados (e explicados na Tabela 3.1) em ordem decrescente, do mais frequente para o menos frequente: semânticos (na lógica da aplicação), ambiente e configurações, interface gráfica, concorrência, geração de *build*, segurança, memória, desempenho, compatibilidade, e bifurcação da *ledger*.

No geral, defeitos semânticos também afetam a segurança das aplicações, uma vez que diferenças entre requisitos e a intenção do programador podem facilitar a fraude (como no ataque DAO). Um exemplo deste tipo de defeito é a transação para endereços inexistentes (possível no Bitcoin) ou órfãos (possível no Ethereum).

Neste estudo [31], os defeitos de segurança não têm novidades e estão relacionados às ocorrências específicas de vulnerabilidades já conhecidas, tais como as seguintes: overflow de inteiros no *timestamp* de um bloco causado por minerador malicioso, ataques por canal lateral de tempo (*timing attack*), viabilizado pelo modo com as senhas



são comparadas na autenticação, e diversas vulnerabilidades de SSL/TLS relacionadas à validação incompleta de certificados que facilitam ataques como o *padding oracle* e BEAST. Em geral, programadores comuns têm muita dificuldade em usar criptografia corretamente [32] e os desenvolvedores de sistemas blockchain não são uma exceção.

Um canal lateral (de tempo) é alimentado pelas variações de tempo das computações sobre valores criptográficos. Por exemplo, quando a comparação de *hashes* ocorre em tempo variável, as pequenas variações de tempo na comparação revelam onde está a diferença entre valores comparados e podem ser utilizadas para deduzir o valor de *hash* original [6]. Para evitar este vazamento de informação, a comparação de *hashes* deve ser realizada em tempo constante e independente de onde ocorre a primeira diferença entre os valores comparados [6].

Os quatro pontos mais importantes na validação do certificado digital são os seguintes: (i) a confirmação do nome do sujeito ou usuário, (ii) a verificação da assinatura digital presente no certificado (e a verificação da cadeia de certificação), (iii) a verificação da data de expiração e (iv) a presença do certificado em uma lista de revogação. Estudos [33][34] revelaram que diversos componentes de software para estabelecimento de conexões TLS permitem que programadores desabilitem partes da verificação do certificado. Porém, a configuração desabilitada pode ser implantada acidentalmente em ambiente de produção. Em particular, a falta de verificação da assinatura ou da cadeia de certificação facilita o ataque do intermediário malicioso (*Man-in-The-Middle* – MiTM).

**Tabela 3.1. Bugs mais comuns em sistemas Blockchain (adaptada de([31]).**

Categoria	Descrição	%
<b>Semântico</b>	Inconsistências entre os requisitos e a intenção do programador	<b>67.23%</b>
▪ <b>Caso ausente</b>	<i>Um caso ou situação em uma função não está implementada</i>	16.07%
▪ <b>Processamento</b>	<i>computações, expressões ou equações incorretas</i>	14.06%
▪ <b>Outros</b>	<i>Qualquer outra funcionalidade errada ou que não atende aos requisitos</i>	8.46%
▪ <b>Tratamento de exceção</b>	<i>Tratamento de exceção inapropriado ou defeituoso</i>	7.40%
▪ <b>Função ausente</b>	<i>Função supostamente presente não está implementada</i>	6.13%
▪ <b>Fluxo de controle</b>	<i>Fluxo de controle implementado incorretamente</i>	5.07%
▪ <b>Situação de borda</b>	<i>Situações de borda e extremos incorretos ou ignorados</i>	4.65%
▪ <b>Tratamento de saída</b>	<i>Apresentação de dados de saída incorreta</i>	3.70%
▪ <b>Tratamento de entrada</b>	<i>Tratamento ou validação de dados de entrada ausente ou incorreto</i>	1.69%
<b>Ambiente/configuração</b>	Erros em dependências, tais como bibliotecas externas, sistemas operacionais e configurações	<b>11.42%</b>
<b>Interface gráfica</b>	Erros de interface, incluindo formatação e ergonomia	<b>6.98%</b>
<b>Concorrência</b>	<i>Deadlocks, data races</i> e outros problemas de sincronização de tarefas	<b>4.44%</b>
<b>geração de build</b>	Erros de compilação, ligação e empacotamento	<b>4.12%</b>
<b>segurança</b>	Vulnerabilidades comuns que se exploradas podem causar danos aos dados, ao software ou ao serviço	<b>1.90%</b>
<b>Memória</b>	Bugs de manipulação imprópria de objetos de memória	<b>1.59%</b>
<b>Desempenho</b>	Bugs que levam a responsividade anormal ou instabilidade mesmo com dados normais (corretos)	<b>1.48%</b>
<b>Compatibilidade</b>	Incompatibilidades que impedem a execução em hardwares, sistemas operacionais, navegadores, etc.	<b>0.74%</b>
<b>Hardfork (Bifurcação)</b>	Mudanças em protocolos fazem transações ou blocos de versões anteriores ficarem inválidos na nova versão	<b>0.11%</b>

### 3.5.2. Vulnerabilidades em *smart contracts*

Blockchain e contratos inteligentes (*chaincodes*) são tecnologias complexas que possuem vários benefícios quando usados em conjunto. A combinação destas tecnologias complexas, em vários contextos de aplicação, levanta novas preocupações com segurança relacionadas não apenas às tecnologias isoladas em seus casos de uso comuns, mas também emergentes das interações desconhecidas e até inesperadas entre estas tecnologias para resolver casos de uso incomuns (inovadores) em novas situações.

Estudos recentes ([35], [36]) já catalogaram defeitos de segurança (vulnerabilidades) em contratos inteligentes. Quatro vulnerabilidades são mais conhecidas: a dependência da ordem de transações, a dependência do carimbo de tempo, o tratamento defeituoso de exceções e a vulnerabilidade de código reentrante.

**Dependência da ordem de transações.** A ordem em que transações são executadas por um contrato inteligente pode alterar o resultado final deste *chaincode*. A vulnerabilidade TOD (*Transaction-Ordering Dependence*) ocorre quando um nó malicioso altera a ordem em que transações são executadas por um contrato. Por exemplo, em transações de compra e venda com criptomoedas, sabendo que o preço vai baixar, um operador malicioso processa primeiro as transações de pagamento com valor alto (processa o máximo que puder, antecipadamente, antes do preço baixar), deixando para depois que o preço baixar (o que é inevitável), o processamento de poucas transações com pagamento mais baixo.

**Dependência do carimbo de tempo.** Há contratos que usam o carimbo de tempo da transação (*timestamp*) como gatilho ou condição para alguma operação crítica. Por exemplo, *timestamp* é utilizado como semente de PRNG. Assim sendo, um nó malicioso, que monta o bloco, pode escolher um *timestamp* válido, porém enviesado, comprometendo a transação ou até a segurança da chave privada do usuário.

**Tratamento de exceções malfeito.** Quando um contrato inteligente aciona (chama) outro, ele deve estar preparado para o caso excepcional do contrato chamado não terminar sua execução corretamente. Se o término anormal não é tratado com a atenção devida, falhas no contrato chamador podem ocorrer e até ser exploradas em ataques e fraudes. Em um exemplo simples, um contrato de crédito, que faz transferência de valores entre origem e destino, que não trata erros no contrato de débito, vai creditar o valor na conta de destino, apesar do erro no débito, sem debitar da conta de origem.

**Vulnerabilidade de código reentrante.** Neste defeito, dois contratos mutuamente dependentes acessam estados intermediários (inseguros por que ainda possivelmente inconsistentes) um do outro. Se o primeiro contrato toma decisões de negócios com base em estados intermediários inconsistentes, a decisão tomada é incorreta. Esta situação pode ser explorada em fraudes e outros ataques.

### 3.5.3. Vulnerabilidades específicas em *smart contracts* Ethereum

Estudos recentes ([35], [36]) categorizam as vulnerabilidades mais comuns em uma das plataformas blockchain mais utilizadas atualmente, a plataforma Ethereum. Existem, em geral, três categorias de vulnerabilidades mais comuns: as da linguagem de programação Solidity, as da máquina virtual Ethereum (*Ethereum Virtual Machine* - EVM), e aquelas associadas ao blockchain Ethereum. A Tabela 3.2 (a seguir) resume esta classificação de vulnerabilidades em Ethereum, que é explicada nos próximos parágrafos.

Todas as vulnerabilidades da linguagem de programação Solidity tem o potencial de exploração em ataques que roubam *ether* (a criptomoeda Ethereum) de contratos. Em relação ao tratamento de exceções malfeito, 28% dos contratos não validam o retorno de funções. Ainda, campos privados de contratos podem ter seus valores públicos na *ledger*, que está em claro. Além disso, há casos de vulnerabilidade relacionadas ao tratamento incorreto de exceções em código reentrante e em conversão de tipos.

Em relação à segurança da Máquina Virtual Ethereum, há vulnerabilidades descobertas nos binários dos contratos inteligentes (*bytecodes*) que são implantados pelos nós da rede P2P e executados pela EVM. Estas vulnerabilidades podem permitir que valores em *ether* sejam enviados para endereços de destino órfãos, ou que exceções disparadas por erros de estouro da limites da pilha de execução sejam explorados em ataques.

Contratos já implantados na EVM são imutáveis, pois são vinculados às transações no blockchain. Por isto, bugs são difíceis de corrigir, por que o contrato defeituoso não pode ser trocado e nem atualizado diretamente. Assim, a recuperação de incidentes associados ao mau funcionamento de *chaincodes* pode ser drástica, por exemplo, com uma bifurcação (*hard fork*) na cadeia de blocos, em que o contrato defeituoso deixa de ser usado e os nós da rede P2P adotam um ramo da cadeia com um novo contrato.

Em relação à segurança da implementação do blockchain Ethereum e como ele gerencia contratos inteligentes, há casos em que a falta de previsibilidade do estado de um contrato no momento em que uma transação é executada (que pode ser diferente do estado quando a transação foi criada), faz com que contratos sejam dependentes da ordem em que as transações são executadas. Isto é, a sequência de execução de transações não é comutativa e muitas vezes é imprevisível devido a existência de transações concorrentes em competição pela precedência na ordem de execução.

Além disso, há vulnerabilidades associadas à criação maliciosa de carimbos de tempo, que afeta contratos que dependem destes carimbos de tempo para geração de sequências pseudoaleatórias ou para a tomada de decisões baseadas em tempo, como no ataque de *timejacking*.

Em geral, para que contratos inteligentes mantenham o determinismo de execução, mesmo quando usam valores pseudoaleatórios na computação do estado, a semente das sequências pseudoaleatórias deve ser obtida de uma fonte acessível a todos os nós da rede que executarão o contrato. Muitos contratos usam os *hashes* ou *timestamps* de blocos como sementes. Porém, estes valores podem ser maliciosamente manipulados por mineradores interessados em influenciar a execução do contrato.

As próximas subseções detalham quatro casos em particular daqueles apresentadas na Tabela 3.2: a dependência da ordem das transações, a dependência do carimbo de tempo, o tratamento de exceção malfeito e o bug de reentrância de código.

**Tabela 3.2. Classificação das vulnerabilidades em Ethereum (adaptada de [36]).**

Nível	Vulnerabilidade	Descrição
Solidity	Chamada para o desconhecido	Mecanismos para invocação de funções tem o efeito colateral de ativar a rotina de fallback (alternativa) do contrato chamado se a função desejada não existir, com o potencial de executar código arbitrário.
	Envio sem <i>Gas</i>	Função <i>send</i> para transferência de <i>ether</i> para um contrato pode disparar uma exceção de falta de <i>Gas</i> , por não ter fundos ( <i>Gas</i> ) suficientes para executar código da função de fallback do contrato chamado, se ele for acionado em caso de erro
	Tratamento de exceção inconsistente	Inconsistente no tratamento de exceções, mostrando comportamentos distintos que dependem de como contratos chamam uns aos outros, podendo até não disparar exceção alguma em caso de erros.
	Conversão de tipos ( <i>type casting</i> )	Capacidade limitada de detecção de erros de conversão de tipo por <i>casting</i> . O <i>casting</i> estático de tipos simples pode ser detectado na compilação. Já o <i>casting</i> dinâmico entre interfaces de contratos, resolvidos em tempo de execução, podem levar a erros não detectáveis (não disparam exceções).
	Reentrância de código	Código não reentrante pode apresentar comportamento anormal se for invocado antes da execução anterior terminar. Funções não recursivas podem ser reentradas se a função de <i>fallback</i> ativar a função chamadora.
	Guarda de segredos	Campos em contratos podem ser públicos ou privados. Campos privados podem ter seus valores revelados se estiveram em uma transação que ficará registrada em claro.
EVM	Bugs imutáveis	Contratos são implantados por meio de transações e por isto os códigos binários implantados são imutáveis. Se um contrato possui um defeito grave que requer sua substituição, não é possível substituí-lo diretamente. Requer medidas de autodestruição explícitas elaboradas pelo programador.
	<i>Ether</i> perdido na transferência	Na transferência de <i>ether</i> para um endereço (de usuário ou de contrato), a EVM não consegue determinar automaticamente se o endereço de destino é válido ou órfão (inexistente ou inválido). Neste caso, o <i>ether</i> enviado é perdido para sempre.
	Limite de tamanho da pilha	Quando um contrato invoca outro, a pilha de chamadas ( <i>call stack</i> ) do contrato é incrementada de um <i>frame</i> (a estrutura de dados da chamada). O limite desta pilha é 1024 <i>frames</i> . Quando o limite é atingido, uma exceção é disparada e pode ser explorada se não for tratada corretamente.
Blockchain	Estado imprevisível	Um usuário que consulta o estado de um contrato e em seguida envia uma transação para o blockchain com base na consulta, não sabe com certeza o estado do contrato quando a sua transação for realizada, por que outras transações concorrentes podem mudar o estado do contrato antes.
	Geração de aleatoriedade	Valores pseudoaleatórios em contratos devem ser obtidos a partir de sementes comuns a todos os nós que executam o contrato. Por exemplo, a partir de informações de blocos (timestamp ou hash) no Blockchain. Um bloco malicioso pode ser construído para enfraquecer a aleatoriedade.
	Restrições de tempo	O timestamp do bloco é escolhido pelo minerador que o cria. Um minerador malicioso pode escolher um timestamp (dentro de um intervalo válido) que beneficie transações específicas no bloco, facilitando ataques e fraudes.

### 3.5.3.1. Vulnerabilidade de dependência da ordem das transações

A vulnerabilidade relacionada à dependência da ordem das transações não é uma exclusividade dos contratos inteligentes em blockchain, sendo um ponto de atenção na garantia de consistência em sistemas distribuídos em geral [4].

Sejam duas transações,  $T_1$  e  $T_2$ , no mesmo bloco e tratadas pelo mesmo contrato  $C(t)$ , onde  $t$  é a transação que parametriza a execução corrente do contrato. A ordem de processamento das transações  $T_1$  e  $T_2$  pelo contrato  $C$  pode afetar o estado final de  $C$ . Em outras palavras, a ordem de processamento das transações não é comutativa:  $C(T_1).C(T_2) \neq C(T_2).C(T_1)$ . Logo, o estado do contrato depende da ordem em que as transações ocorrem.

O nó da rede P2P que processa as transações não apenas conhece a ordem de execução das transações, mais também pode influenciar esta ordem e, assim, influenciar o estado final do contrato. O Programa 3.2 foi adaptado de [35] e contém um trecho de código fonte escrito em Solidity que funciona como *Market Place*, onde usuários compram e vendem *tokens* associados a algum ativo de valor.

Neste contrato, dois métodos são ilustrados: *buy(...)* e *updatePrice(...)*. Eles são invocados assincronamente e por usuários diferentes em transações distintas. Por isto, o preço pode mudar antes ou depois da compra. Logo, o método *buy(...)* depende do método *updatePrice(...)* na medida em que a transação de compra utilizará o preço corrente, de acordo com a atualização mais recente do contrato.

Assim, um operador malicioso pode manipular a ordem de execução das transações de compra e atualização de preço, de modo que as transações que aumentam o preço sejam processadas antes das transações de compra (garantindo a compra a um preço mais alto) e, só depois de tudo, as transações que baixam o preço seriam processadas.

**Programa 3.2. Contrato com dependência da ordem das transações (adaptado de [35]).**

```

01 contract Marketplace {
02     uint public price;
03     uint public stock;
04
05     /.../
06     function updatePrice ( uint _price ){
07         if ( msg.sender == owner ) price = _price;
08     }
09
10     function buy ( uint quant ) returns ( uint ){
11         if ( msg.value < quant * price || quant > stock ) throw;
12         stock -= quant;
13         /.../
14 }}

```



### 3.5.3.2. Vulnerabilidade de dependência do carimbo de tempo

A vulnerabilidade relacionada à dependência de carimbos de tempo ocorre quando um contrato usa o timestamp do bloco como gatilho ou condição para alguma operação crítica sobre dados sensíveis. Tal como, por exemplo, a geração de números pseudoaleatórios, em que o timestamp é usado como (parte da) semente de um PRNG.

Contratos podem necessitar de PRNGs para produzir valores pseudoaleatórios de maneira determinística e replicável pelos outros nós da rede P2P executando o mesmo contrato. Na plataforma Ethereum, até o momento de escrita deste texto, não havia PRNGs disponíveis para uso programático na codificação de contratos em Solidity. Por esta razão, muitos contratos usam o timestamp do bloco na geração de valores pseudoaleatórios, com a suposição de que ele seja uma boa semente. Porém, o operador do nó da rede P2P, que monta o bloco, pode escolher um timestamp válido mas enviesado, influenciando a execução do contrato para obter algum benefício indevido.

O Programa 3.3 foi adaptado de [35] e contém um trecho de código escrito em Solidity em que o contrato depende do timestamp do bloco para gerar um valor randômico. Neste trecho de contrato, a função *random()* calcula um valor aleatório que usa o timestamp do bloco como parte da semente. O timestamp do bloco (que é obtido da variável predefinida *block.timestamp*) é atribuído à variável *salt* e utilizado na computação da variável *seed*.

Vale lembrar que os bons geradores de números pseudoaleatórios produzem sequências numéricas que se comportam, para a maioria dos usos práticos, como números aleatórios verdadeiros. Porém, estas sequências numéricas pseudoaleatórias são, de fato, geradas por algoritmos determinísticos, os PRNGs, cujo fluxo de execução pode ser reproduzido a partir da repetição do parâmetro de entrada chamado de semente (*seed*). Isto é, duas execuções distintas do mesmo PRNG, que são alimentadas (parametrizadas) com a mesma semente, produzirão a mesma sequência numérica pseudoaleatória.

Esta vulnerabilidade não deve ser confundida com o ataque de *timejacking*, descrito na seção 3.5.4.

**Programa 3.3. Contrato com dependência de carimbos de tempo (adaptado de [35]).**

```

01 contract theRun {
02   uint private Last_Payout = 0;
03   uint256 salt = block.timestamp;
04
05   function random returns ( uint256 result ){
06     uint256 y = salt * block.number / (salt %5);
07     uint256 seed = block.number /3 + (salt %300) + Last_Payout + y;
08     //h = the blockhash of the seed -th last block
09     uint256 h = uint256 ( block.blockhash ( seed ));
10     return uint256 (h % 100) + 1; //random number between 1 and 100
11   }
12 }

```

### 3.5.3.3. Vulnerabilidade de tratamento de exceções malfeito

A vulnerabilidade relacionada ao tratamento malfeito de exceções se manifesta, muitas vezes, quanto uma transação de semântica complexa está dividida em transações menores processadas por vários contratos distintos, mas aninhados, que invocam uns aos outros. Neste cenário, falhas nos contratos internos afetam os contratos externos e a atomicidade da transação de mais alto nível é comprometida.

No trecho de código a seguir (adaptado de [36]), sejam dois contratos, Alice e Bob, o Bob chama Alice. Um erro em Alice pode disparar uma exceção e impedir a sua finalização correta. O término anormal de Alice pode não ter sido previsto por Bob e, por isto, não é tratado, levando Bob a falhar também.

```
contract Alice { function ping(uint) returns (uint) }
contract Bob { uint x=0; function pong(Alice c){x=1; c.ping(42);x=2;}}
```

Um exemplo simples desta situação ocorre em uma transação de débito e crédito processada por dois contratos, em que o contrato de crédito não trata um erro no contrato de débito e credita mesmo assim o endereço de destino sem antes debitar o valor da conta de origem. O Programa 3.4 foi adaptado de [35] e contém um trecho de código escrito em Solidity em que o contrato não verifica o retorno da chamada de outros contratos (linhas 14 e 15). Neste código, um usuário paga para outro usuário com o objetivo de obter o título de *KingOfTheEtherThrone* (o rei do trono de *ether*). Com uma lógica simples, o contrato faz com que o novo rei pague mais pela coroa que o rei anterior, de modo que o valor do trono sempre aumenta.

Erros no processamento da transferência de valor do rei novo para o rei atual (nas linhas 14 e 15, via uma chamada direta para a função *address.send()* do endereço do rei atual) podem fazer com que o rei novo perca o seu dinheiro (seja debitado) sem que o rei antigo tenha recebido (seja creditado). Outra situação de erro possível seria o crédito do rei antigo sem o débito do novo rei.

**Programa 3.4. Contrato com tratamento de exceção malfeito (adaptado de [35]).**

```
01 contract KingOfTheEtherThrone {
02
03     struct Monarch {
04         address ethAddr; // address of the king
05         string name;
06         uint claimPrice; // how much he pays to previous king
07         uint coronationTimestamp;
08     }
09
10     Monarch public currentMonarch;
11     // claim the throne
12     function claimThrone( string name ) {
13         /.../
14         if ( currentMonarch.ethAddr != wizardAddress )
15             currentMonarch.ethAddr.send( compensation );
16         /.../
17         // assign the new king
18         currentMonarch = Monarch(
19             msg.sender, name, valuePaid, block.timestamp);
20     }}
```

### 3.5.3.4. Vulnerabilidade de reentrância de código

Um programa de computador é reentrante quando ele pode ser interrompido no meio de sua execução e ser chamado novamente, a partir do início, antes que a execução anterior tenha terminado. Uma vez que a nova execução tenha terminado, a execução anterior pode prosseguir sem erros. Funções recursivas devem ser reentrantes por natureza. Funções que dependem de variáveis globais geralmente não são reentrantes, uma vez que múltiplas invocações da mesma função podem modificar o valor da variável global, interferindo nas outras invocações em execução.

Esta vulnerabilidade pode se manifestar no seguinte exemplo (adaptado de [36]). Sejam dois contratos, Bob e Mallory, em que Mallory chama Bob. A função *Bob.ping(c)* deveria enviar apenas 2 wei (1 wei =  $10^{-18}$  ether) para o endereço *c* via uma chamada genérica *call*. A variável global em Bob *sent* sinaliza se o envio já ocorreu alguma vez. Se esta função é invocada para o endereço de Mallory, a função de *fallback* será ativada e, por sua vez, invocará *Bob.ping(c)* para o endereço de Mallory, que invocará a *fallback* novamente, iniciando um laço infinito em que a variável *sent* nunca recebe o valor *true*. O laço perdura até que não haja mais espaço na pilha de chamadas de função, ou que Bob fique sem *gas* para executar código, ou sem saldo para a transferência.

```
contract Bob { bool sent = false; function ping(address c) {
  if (!sent) { c.call.value(2)(); sent = true; }}
```

```
contract Bob { function ping(); } // isto eh uma interface
contract Mallory { function(){ Bob(msg.sender).ping(this); }}
```

O Programa 3.5 (adaptado de [35]) contém um trecho de código em que um contrato de conta bancária sofre da vulnerabilidade de reentrância de código similar àquela que causou o ataque DAO. Neste contrato, a função de saque, *withdrawBalance()*, não é reentrante, mas pode ser erroneamente interrompida antes do saldo da conta do sacado ser zerado (linha 13, a variável global *userBalances[msg.sender]* é zerada). Nesta situação, outros contratos ainda perceberiam a conta bancária com saldo positivo e poderiam conseguir sacar antes do saldo zerar de fato.

#### Programa 3.5. Exemplo da vulnerabilidade de código reentrante (adaptado de [35]).

```
01 contract SendBalance {
02   mapping( address => uint ) userBalances;
03   bool withdrawn = false;
04
05   function getBalance( address u ) constant returns ( uint )
06   { return userBalances[u]; }
07
08   function addToBalance() {userBalances[msg.sender] += msg.value;}
09
10   function withdrawBalance () {
11     if (!(msg.sender.call.value( userBalances[msg.sender] ) ()))
12       { throw; }
13     userBalances[msg.sender] = 0;
14   }
```

### 3.5.3.5. O ataque DAO

O ataque DAO [25] aconteceu sobre uma Organização Autônoma Descentralizada (*Decentralized Autonomous Organization* – DAO) da plataforma Ethereum que implementava um sistema de arrecadação tipo *crowd-funding*, que arrecadou milhões antes de ser atacado em junho de 2016, quando um atacante conseguiu desviar mais de um terço do saldo da DAO para contas sob seu controle, até que uma bifurcação drástica (*hard fork*) no blockchain reverteu os efeitos do ataque.

O exemplo apresentado nesta seção é uma versão simples do ataque DAO que foi adaptada de [36] e tem uma atualização feita pelos autores do trabalho original disponível na URL [co2.unica.it/ethereum/doc/attacks.html](http://co2.unica.it/ethereum/doc/attacks.html). O Programa 3.6 ilustra dois contratos: SimpleDAO e Mallory. O primeiro, SimpleDAO, permite que um doador faça doações de *ether* para financiar contratos da sua escolha. Os contratos beneficiados pela doação podem sacar o que for doado para eles.

No exemplo, para o atacante poder roubar todo o *ether* do contrato SimpleDAO, primeiro, ele publica o contrato Mallory. Então, o atacante doa (com SimpleDAO) algum *ether* para Mallory, o que ativa a função de *fallback* de Mallory, que por sua vez faz um saque (função *withdraw* de SimpleDAO). A função de saque faz a transferência para o endereço de Mallory, o que ativa novamente a função de *fallback*, que fará um novo saque, em um laço infinito. A chamada anterior de *withdraw* foi interrompida antes de atualizar a variável global *credit*, por isso a nova chamada ainda consegue fazer saques. O laço continua até que a pilha de execução se esgote, ou que não haja mais *gas* para executar código, ou que o saldo de SimpleDAO tenha se esgotado. O atacante pode tanto realizar o ataque em série quanto dar mais *gas* no início. Este ataque explora duas vulnerabilidades: a reentrância de função e a chamada de função desconhecida.

**Programa 3.6. Um exemplo simples de código para o ataque DAO (adaptado de [36]).**

```

01 pragma solidity ^0.4.2;
02
03 contract SimpleDAO {
04     mapping (address => uint) public credit;
05
06     function donate(address to) payable { credit[to] += msg.value; }
07
08     function query(address to) returns (uint){ return credit[to]; }
09
10     function withdraw(uint amount) {
11         if (credit[msg.sender]>= amount) {
12             bool res = msg.sender.call.value(amount) ();
13             credit[msg.sender]-=amount;
14     }}}
15
16 contract Mallory {
17     SimpleDAO public dao; address owner;
18
19     function Mallory(SimpleDAO addr){owner = msg.sender; dao = addr;}
20
21     function getJackpot() { bool res = owner.send(this.balance); }
22
23     function() payable { dao.withdraw(dao.query(this)); }
24 }

```

### 3.5.4. Ataques específicos contra a criptomoeda Bitcoin

O conhecimento e a análise de ataques contra criptomoedas em geral e o Bitcoin em particular são instrutivos para evitar casos semelhantes em construções análogas. A maioria dos ataques se refere ao gasto repetido ou duplicado de criptomoedas (*double spending*). Primeiro, esta seção analisa a segurança de transações no Bitcoin em dois aspectos: assinaturas duplicadas e maleabilidade. Em seguida, ataques bem documentados [2] são descritos, tais como: o ataque 51%, o ataque de competição (*race attack*), o ataque do minerador malicioso, a enxurrada de transações e o *timejacking*.

#### 3.5.4.1. Segurança da transação no Bitcoin

No Bitcoin, o *double spending* pode ser entendido da seguinte forma. Sejam duas transações duplicadas que usam o mesmo valor de origem. A primeira transação a entrar no blockchain é considerada válida e a outra é descartada. Um atacante poderoso poderia substituir uma transação que já entrou no blockchain por outra que usa o mesmo valor de origem. Fazendo com que um valor que já foi gasto seja utilizado novamente.

Existe uma variedade de tipos de *eWallets* Bitcoin. As carteiras em software são as mais comuns e podem ser aplicações autônomas ou serviços on-line (*online eWallets*). As *eWallets* on-line podem manter informações centralizadas ou manter informações de usuário cifradas e apenas disponíveis às aplicações clientes. Ainda, muitas *eWallets* Bitcoin podem usar hardware seguro, ou simplesmente imprimir as chaves públicas em papel. Existe ainda a opção de geração de chaves a partir de senhas (as *brainwallets*). Em qualquer caso, o software é suscetível a exploração de vulnerabilidades comuns.

Transações multi-assinadas (assinadas mais de uma vez por entidades diferentes) aumentam a segurança das transações no Bitcoin, mas precisam que todas as entidades estejam disponíveis no momento de realização da transação. Multi-assinaturas com limiar, em que uma quantidade mínima de assinantes é necessária (mas não requer todos os assinantes presentes), aumenta a flexibilidade de utilização de multi-assinaturas.

O Bitcoin utiliza a criptografia de curvas elípticas para a assinatura de transações. Em particular, é utilizado o algoritmo criptográfico padronizado ECDSA com a curva *secp256k1*. O ECDSA pode ser mal utilizado e produzir assinaturas digitais vulneráveis. Cada assinatura requer um número aleatório único e imprevisível (*nonce*). Por isto, ele é vulnerável à geração de números pseudoaleatórios ruins. Assinaturas digitais geradas com o mesmo *nonce*, ou *nonces* parecidos (alguns bits em comum) podem revelar a chave privada (facilitando o roubo de bitcoins). Por isto, estas implementações são muito sensíveis aos defeitos de segurança no uso do ECDSA, tais como sementes do PRNG com entropia baixa e *nonces* repetidos (fixos nos programas).

Além disso, no Bitcoin, as transações são maleáveis. O *hash* da transação (usado como identificador) é computado sobre dados diferentes daqueles usados na assinatura digital da transação. Esta decisão de projeto faz com que seja possível alterar o *hash* de identificação da transação sem que a assinatura digital seja adulterada ou necessite de modificação. Transações idênticas com identificadores (*hashes*) diferentes podem ser usadas em ataques de *double spending*.



### 3.5.4.2. Outros ataques contra o Bitcoin

Em criptomoedas como o Bitcoin, a segurança da transação também depende do fato de que não há um minerador (ou grupo de mineradores) que controla a maioria da rede de mineração. Se os mineradores são pequenos e em grande quantidade, a chance de um minerador ser bem-sucedido na validação de um bloco é pulverizada. Para aumentar suas chances, mineradores se associam em *pools* de mineração.

No **ataque de 51%** (*51%+ attack*), para substituir uma transação em um bloco, o atacante deve minerar de novo todos os blocos anteriores e acompanhar o passo de geração de blocos novos pelo restante da rede. Para tal, a capacidade de computação de valores *hash* do atacante (*hash rate*) deve ser maior que a da rede P2P restante. Por isto, este ataque só é viável para um atacante que tem domínio da rede P2P, isto é, ele controla mais de 50% dos nós (ou da capacidade de *hash*) da rede. Este atacante poderoso pode sequestrar a rede, se recusando a minerar blocos de outros mineradores. Por isto, este também é um ataque de negação de serviço contra outros mineradores.

No **ataque de competição** (*race attack*), transações duplicadas em nós diferentes da rede P2P, causam a falsa impressão de *double-spending* devido a latência da rede e às diferenças de tempo de propagação de blocos a partir de nós próximos ou de nós distantes. Os nós próximos de onde a transação iniciou percebem a transação antes dos nós mais distantes. O *double spending* aparente só existe até que a transação entre em um bloco e tal bloco alcance os nós da rede P2P. Os nós mais próximos percebem a duplicação antes dos nós mais distantes.

No **ataque do minerador malicioso** (*Finney attack*), o atacante minera um bloco em segredo com uma transação sua para si mesmo (autotransação). A transação não é difundida na rede P2P. Antes de liberar o bloco secreto, o atacante emite outra transação duplicada tendo como destino uma vítima, que aceita o pagamento sem confirmar o bloco (uma prática ruim). Então, o atacante libera o bloco secreto, passando o seu pagamento na frente, antes que outro minerador valide algum outro bloco com a transação de pagamento para a vítima. Daí o *double-spending*.

O ataque de *spam* ou **enxurrada de transações** (*Transaction spamming/flooding*) é um ataque de negação de serviço contra a rede P2P de um blockchain. Neste ataque, uma enxurrada de autotransações inibe que o nós atacados processem outras transações. Análises indicam que este ataque não é viável no Bitcoin, principalmente, por três motivos: (i) apenas poucas transações gratuitas são permitidas em um bloco, (ii) a taxa de serviço do minerador encarece um ataque volumoso, e (iii) as transações de valor muito baixo podem até ser descartadas. Porém, o ataque pode ser viável em outros blockchains com mecanismos de incentivo diferentes para o consenso de mineradores.

No **ataque de timejacking**, mineradores maliciosos (ou apenas um atacante poderoso o suficiente) manipulam o tempo das transações direcionadas para um nó alvo do ataque, fazendo com que estas transações estejam atrasadas no tempo (dentro de um limite de tolerância de sincronismo da rede). O processamento atrasado de blocos pode fazer com que o nó vítima fique adicionando blocos em um ramo da cadeia que não foi escolhido pela maioria dos outros nós, e também só aceite blocos para este ramo órfão. O nó alvo do ataque fica isolado, por isto, este é um tipo de ataque de negação de serviço.

### 3.6. Considerações finais

Este texto abordou, a partir da análise de estudos recentes, dois assuntos importantes do universo blockchain: desenvolvimento de aplicações e segurança de software. O objetivo foi o de fomentar o desenvolvimento de aplicações blockchain seguras.

Blockchain é uma tecnologia em que a prática parece estar à frente de teoria em vários aspectos [13]. A tecnologia blockchain (e as criptomoedas como o Bitcoin) está na interseção entre segurança de software, sistemas distribuídos e sistemas dinâmicos [13] (como os sistemas econômicos).

O nível de descentralização obtido com blockchain era considerado inatingível na teoria [12]. Porém a abordagem de que pequenas falhas são aceitas pelo sistema de consenso viabiliza a tecnologia na prática [12]. Há diversas oportunidades para a inovação tecnológica em aplicações desta tecnologia, dentre as quais estão as plataformas para desintermediação, o armazenamento de dados globalmente distribuído, e as transações de semânticas específicas [13].

A próxima geração da tecnologia blockchain, apoiada por transações sofisticadas e contratos inteligentes, tem o potencial de viabilizar as organizações verdadeiramente autônomas [12]. Pesquisadores alegam que a lacuna entre teoria e prática faz com que a tecnologia blockchain não seja ainda totalmente entendida [12]. Além disso, a grande variedade de implementações disponíveis e de estudos empíricos realizados ou em andamento dificultam saber qual (plataforma de) blockchain vai prevalecer [12].

Além disso, existe uma imaturidade relativa das plataformas blockchain para desenvolvimento de aplicações. Plataformas mais maduras, como a Ethereum, tendem a ser mais estáveis, possuem documentação melhor e têm sido mais estudadas do ponto de vista da segurança. Porém, por serem mais conhecidas e utilizadas, também estão mais expostas a ataques. Por outro lado, plataformas mais novas, como a Hyperledger, ainda apresentam uma instabilidade relativa, documentação dispersa e poucos exemplos de utilização com código fonte disponível, além de ainda não terem sofrido o escrutínio da comunidade sobre sua segurança.

Finalmente, de modo geral, o desenvolvedor de software interessando na tecnologia precisa investir bastante tempo no aprendizado das plataformas blockchain e suas ferramentas para desenvolvimento de aplicações. Além disso, mesmo adotando métodos ágeis para o desenvolvimento rápido de aplicações, as questões de segurança precisam ser tratadas com atenção, levando em conta segurança em camadas.

**Agradecimentos.** Os autores agradecem à Fundação [CPqD](#) pelo apoio dado à elaboração deste capítulo de livro. O CPqD oferece uma série de [webinars](#) [37] com o objetivo de desmistificar a tecnologia blockchain.

### 3.7. Referências

- [1] S. Underwood, “Blockchain beyond bitcoin,” *Communications of the ACM*, vol. 59, no. 11, pp. 15–17, 2016.
- [2] P. Franco, *Understanding Bitcoin: Cryptography, engineering and economics*. John Wiley & Sons, 2014.

- [3] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, “A Fistful of Bitcoins: Characterizing Payments Among Men with No Names,” *Commun. ACM*, vol. 59, no. 4, pp. 86–93, 2016.
- [4] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.
- [5] I. Bashir, *Mastering Blockchain*. Packt Publishing, 2017.
- [6] A. Braga and R. Dahab, “Introdução à Criptografia para Programadores,” in *Caderno de minicursos do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg*, 2015, pp. 1–50.
- [7] D. Hankerson, S. Vanstone, and A. Menezes, *Guide to elliptic curve cryptography*. 2004.
- [8] M. Swan, *Blockchain - Blueprint for a New Economy*. 2015.
- [9] “Hyperledger - Blockchain Technologies for Business.” [Online]. Available: <https://www.hyperledger.org>.
- [10] “Ethereum Blockchain App Platform.” [Online]. Available: <https://www.ethereum.org>.
- [11] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, “The blockchain as a software connector,” *Proc. of 13th Working IEEE/IFIP Conference on Software Architecture, WICSA*, pp. 182–191, 2016.
- [12] F. Tschorsch and B. Scheuermann, “Bitcoin and beyond: A technical survey on decentralized digital currencies,” *IEEE Comm. Surveys and Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.
- [13] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “Research Perspectives and Challenges for Bitcoin and Cryptocurrencies,” *IEEE Symposium on Security and Privacy*, pp. 104–121, 2015.
- [14] M. Giancaspro, “Is a ‘smart contract’ really a smart idea? Insights from a legal perspective,” *Computer Law and Security Review*, vol. 1, pp. 1–11, 2017.
- [15] C. Barski and C. Wilmer, *Bitcoin for the Befuddled*. No Starch Press, 2014.
- [16] C. Dannen, *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Apress, 2017.
- [17] K. Krombholz, A. Judmayer, M. Gusenbauer, and E. Weippl, “The Other Side of the Coin: User Experiences with Bitcoin Security and Privacy,” *Financial Cryptography and Data Security 2016*, 2016.
- [18] S. Eskandari, D. Barrera, E. Stobert, and J. Clark, “A First Look at the Usability of Bitcoin Key Management,” *USEC, San Diego, CA, USA*, no. February, 2015.
- [19] “Bitcoin Project.” [Online]. Available: <https://bitcoin.org>.
- [20] “Ripple and RippleNet - The world’s only enterprise blockchain solution for global payments.” [Online]. Available: <https://ripple.com>.
- [21] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,”

- www.bitcoin.org*. p. 9, 2008.
- [22] J. Saltzer and M. Schroeder, “The protection of information in computer systems,” *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.
- [23] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way*. 2001.
- [24] J.Kurose and K.Ross, “Redes de Computadores e a Internet,” *Person*, p. 28, 2006.
- [25] D. Siegel, “Understanding the DAO attack.” [Online]. Available: <https://www.coindesk.com/understanding-dao-hack-journalists/>.
- [26] J. W. Bos, J. A. Halderman, N. Heninger, J. Moore, M. Naehrig, and E. Wustrow, “Elliptic curve cryptography in practice,” in *Financial Cryptography and Data Security*, Springer, 2014, pp. 157–175.
- [27] OWASP, “OWASP Top Ten Project,” *OWASP*, 2013. [Online]. Available: [https://www.owasp.org/index.php/Top\\_10](https://www.owasp.org/index.php/Top_10).
- [28] SANS/CWE, “TOP 25 Most Dangerous Software Errors,” *SANS/CWE*. [Online]. Available: [www.sans.org/top25-software-errors](http://www.sans.org/top25-software-errors).
- [29] F. Long, C.-M. U. C. C. Center, D. Mohindra, and R. C. Seacord, *The CERT Oracle Secure Coding Standard for Java*. Addison-Wesley, 2011.
- [30] L. Notice, “CERT C Programming Language Secure Coding Standard,” 2007.
- [31] Z. Wan, D. Lo, X. Xia, and L. Cai, “Bug characteristics in blockchain systems: a large-scale empirical study,” in *Proceedings of the 14th International Conference on Mining Software Repositories*, 2017, pp. 413–424.
- [32] A. Braga and R. Dahab, “Mining Cryptography Misuse in Online Forums,” in *2nd Int. Workshop on Human and Social Aspect of Software Quality*, 2016.
- [33] M. Georgiev, S. Iyengar, and S. Jana, “The most dangerous code in the world: validating SSL certificates in non-browser software,” in *Proc. of the ACM conference on Computer and Comm. Security - CCS*, 2012, pp. 38–49.
- [34] S. Fahl, M. Harbach, and T. Muders, “Why Eve and Mallory love Android: An analysis of Android SSL (in)security,” in *ACM Conf. on Comp. and comm. security CCS*, 2012, pp. 50–61.
- [35] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making Smart Contracts Smarter,” *CCS*, pp. 254–269, 2016.
- [36] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts (SoK),” *LNCS (subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10204 LNCS, no. July, pp. 164–186, 2017.
- [37] CPqD, “Webinars Blockchain.” [Online]. Available: <https://www.cpqd.com.br/midia-eventos/webinars/webinars-blockchain/>.