MINICURSOS

SBSeg 2017

XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

6 A 9 DE NOVEMBRO - BRASÍLIA - DF

XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

6 a 09 de novembro de 2017
Brasília - DF - Brasil

MINICURSOS

Editora
Sociedade Brasileira de Computação - SBC

Organizadores
Raul Ceretta Nunes, UFSM
Edna Dias Canedo, UnB
Rafael Timóteo de Sousa Júnior, UnB

Realização
FGA/UnB – Faculdade UnB Gama, Universidade de Brasília
ENE/UnB – Departamento de Engenharia Elétrica, Universidade de Brasília

Promoção
SBC – Sociedade Brasileira de Computação

Capa: Diego Oliveira, UnB
Editoração: Raul Ceretta Nunes, UFSM

Dados Internacionais de Catalogação na Publicação (CIP)

S612m Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (17. : 2017 : Brasília, DF)
Minicursos [do] XVII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, 06 a 09 de novembro de 2017, Brasília, DF, Brasil ; organizadores, Raul Ceretta Nunes, Edna Dias Canedo, Rafael Timóteo de Sousa Júnior. – Brasília, DF : Sociedade Brasileira de Computação, 2017.
viii, 200 p. ; il. ; 23 cm

Acima do título: SBSeg 2017
ISBN: 978-85-7669-410-6

1. Ciência da Computação – Eventos 2. Informática – Eventos 3. Sistemas de Informação – Eventos 4. Segurança de Sistemas – Eventos 5. Segurança de Informação – Eventos I. Nunes, Raul Ceretta II. Canedo, Edna Dias III. Sousa Júnior, Rafael Timóteo IV. Título V. Título: Minicursos SBSeg 2017.

CDU 004(063)

Ficha catalográfica elaborada por Alenir Goularte - CRB-10/990
Biblioteca Central da UFSM

Sociedade Brasileira de Computação – SBC

Presidência
Lisandro Zambenedetti Granville (UFRGS), Presidente
Thais Vasconcelos Batista (UFRN), Vice-Presidente

Diretorias
Renata de Matos Galante (UFRGS), Diretora Administrativa
Carlos André Guimarães Ferraz (UFPE), Diretor de Finanças
Antônio Jorge Gomes Abelém (UFPA), Diretor de Eventos e Comissões Especiais
Avelino Francisco Zorzo (PUC-RS), Diretor de Educação
José Viterbo Filho (UFF), Diretor de Publicações
Claudia Lage da Motta (UFRJ), Diretora de Planejamento e Programas Especiais
Marcelo Duduchi Feitosa (CEETEPS), Diretor de Secretarias Regionais
Eliana Silva de Almeida (UFAL), Diretora de Divulgação e Marketing

Diretorias Extraordinárias
Ricardo de Oliveira Anido (UNICAMP), Diretor de Relações Profissionais
Esther Colombini, Diretora de Competições Científicas
Raimundo José de A. Macêdo (UFBA), Diretor de Cooperação com Sociedades Científicas
Cláudia Cappelli (UNIRIO), Diretora de Articulação com Empresas

Conselho

Mandato 2015-2019
Altigran Soares da Silva (UFAM)
Ana Carolina Salgado (UFPE)
Fabio Kon (USP)
Rodolfo Azevedo (UNICAMP)
Paulo Roberto Freire Cunha (UFPE)

Mandato 2017-2021
José Carlos Maldonado (USP)
Roberto da Silva Bigonha (UFMG)
Rosa Maria Vicari (UFRGS)
Cristiano Maciel (UFMT)
Itana Maria De Souza Gimenes (UEM)

Suplentes - Mandato 2017-2019
Alex Sandro Gomes (UFPE)
Ismar Frango Silveira (UNICSUL, Mackenzie)
Sérgio Lifschitz (PUC-RIO)
Hermano Perrelli de Moura (UFPE)
André Luís Alice Raabe (UNIVALI)

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS

CNPJ: 29.532.264/0001-78
http://www.sbc.org.br

Comissão Especial em Segurança da Informação e de Sistemas Computacionais – CESeg

Coordenadores
Eduardo L. Feitosa, UFAM (Coordenador)
Altair Santin, PUCPR (Vice)

Comitê consultivo
Leonardo Barbosa, UFMG
Diego Aranha, UNICAMP
Michelle Silva Wangham, UNIVALI
Carlos Maziero, UFPR
Igor Moraes, UFF
Pedro Braconnot Velloso, UFRJ
Daniel Macêdo Batista, USP

Mensagem do Coordenador de Minicursos

O Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg) é um evento científico promovido anualmente pela Sociedade Brasileira de Computação (SBC) e representa o principal fórum no país para a apresentação de pesquisas e atividades relevantes ligadas à segurança da informação e de sistemas. O evento promove a integração da comunidade brasileira de pesquisadores e profissionais atuantes nessa área. Entre as principais atividades técnicas do SBSeg encontram-se a trilha de Minicursos, que têm como objetivo a atualização em temas normalmente não cobertos nas grades curriculares e que despertem grande interesse entre acadêmicos e profissionais.

Nesta edição do SBSeg (2017), 20 propostas de minicursos foram submetidas, um número expressivo que demonstra a importância deste evento no panorama nacional de pesquisa. Destas, 4 foram selecionadas para publicação e apresentação, representando assim uma taxa de aceitação de 20%. O Comitê de Avaliação dos Minicursos foi composto por 14 renomados pesquisadores para a elaboração dos pareceres. Cada proposta recebeu ao menos 3 pareceres, gerando ao todo mais 60 revisões. Além disto, inúmeras mensagens foram trocadas entre os membros do comitê durante a fase de discussão. Cada minicurso selecionado corresponde a um capítulo deste livro e sua apresentação é parte da programação do SBSeg 2017.

Este livro reúne 4 capítulos produzidos pelos autores das propostas de minicursos aceitas. O Capítulo 1, alinhado à tendência crescente e inevitável de uso de pequenos dispositivos computacionais, discute questões contemporâneas na área de segurança e privacidade no contexto da computação ubíqua, bem como aponta para problemas de pesquisa ainda não solucionados. O Capítulo 2, de maneira similar, apresenta as principais tecnologias de segurança baseada em hardware, abordando funcionalidades e usos da arquitetura Intel *Software Guard Extentions* (SGX). Os Capítulos 3 e 4 discutem a tecnologia blockchain, uma tecnologia baseada em difusão e que tem impulsionado novas pesquisas, desenvolvimentos e inovações na área de segurança computacional. O Capítulo 3, apresenta os fundamentos da tecnologia e os aspectos de programação que envolvem o desenvolvimento de aplicações e segurança de sistemas. O Capítulo 4, apresenta como a tecnologia blockchain pode ser usada para prover segurança e privacidade no contexto da Internet das Coisas (IoT). Enfim, o resultado é um livro com assuntos atuais e que permitem uma atualização em temas de interesse da academia e indústria.

Como Coordenador de Minicursos, gostaria de expressar o meu agradecimento aos membros do Comitê de Avaliação dos Minicursos por terem aceitado participar voluntariamente dessa empreitada e pelo excelente trabalho que fizeram no processo de avaliação e seleção dos minicursos. Gostaria de também agradecer aos Coordenadores Gerais do SBSeg 2017, Edna (UnB) e Rafael (UnB), pela disponibilidade e suporte providos ao longo de todo o processo e pela confiança depositada em mim para coordenar estes minicursos. Finalmente, gostaria de agradecer aos autores por terem prestigiado este evento ao submeterem suas propostas de minicursos.

Raul Ceretta Nunes, UFSM
Coordenador de Minicursos do SBSeg 2017

Comitês

Comitê de Organização

Coordenação Geral
Edna Dias Canedo, FGA – UnB
Rafael Timóteo de Sousa Junior, ENE – UnB

Coordenação de Minicursos
Raul Ceretta Nunes, UFSM

Comitê de Organização Local
Aletéia Patrícia Favacho de Araújo, CIC – UnB
Alex Sandro Roschildt Pinto, Campus Blumenau – UFSC
César Augusto Borges de Andrade, CDS – Exército Brasileiro
Diego Martins de Oliveira, IFB/UnB
Edison Pignaton de Freitas, Inf – UFRGS
Eduardo Adilio Pelinson Alchieri, CIC – UnB
Fabiano Cavalcanti Fernandes, IFB
Fábio Lúcio Lopes de Mendonça, ENE – UnB
Georges Daniel Amvame Nze, ENE – UnB
Giovanni Almeida Santos, FGA – UnB
João Paulo Carvalho Lustosa da Costa, ENE – UnB
Priscila América Solís Mendez Barreto, CIC – UnB
Robson de Oliveira Albuquerque, ENE – UnB
Sérgio Andrade de Freitas, FGA – UnB
Ugo Silva Dias, ENE – UnB
William Ferreira Giozza, ENE – UnB

Comitê Técnico
Jackson Ferreira do Nascimento, FT – UnB
Wesley Gongora de Almeida, FT – UnB

Comitê de Programa dos Minicursos

Altair Santin, PUCPR
Carlos Raniery Paula dos Santos, UFSM
Célio Vinicius Neves de Albuquerque, UFF
Eduardo Souto, UFAM
Julio Hernandez, UNICAMP
Laerte Peotta, UNB
Leonardo Oliveira, UFMG
Luis Kowada, UFF
Luiz Fernando Rust da Costa Carmo, Inmetro
Marcos Simplicio Jr, Escola Politécnica, USP
Michelle Wangham, Univali
Raul Ceretta Nunes, UFSM - Coordenador
Ricardo Dahab, UNICAMP
Rossana Andrade, UFC

Sumário

Capítulo 1

O Computador para o Século 21: Desafios de Segurança e Privacidade após 25 Anos[1]

Leonardo B. Oliveira[a], **Fernando Magno Quintão Pereira**[a], **Rafael Misoczki**[b], **Diego F. Aranha**[c], **Fábio Borges**[d], **Michele Nogueira**[e], and **Michelle Wangham**[f]

[a]Universidade Federal de Minas Gerais (UFMG)
[b]Intel Corporation
[c]Universidade Estadual de Campinas (Unicamp)
[d]Laboratório Nacional de Computação Científica (LNCC)
[e]Universidade Federal do Paraná (UFPR)
[f]Universidade do Vale do Itajaí (UNIVALI)

Resumo

Passaram décadas desde Mark Weiser publicou seu trabalho influente sobre como seria um computador do século 21. Ao longo dos anos, alguns dos recursos da UbiComp apresentados nesse mincurso foram sendo gradualmente adotados pelas indústrias no mercado de tecnologia. Embora tal evolução tecnológica tenha resultado em muitos benefícios para a nossa sociedade, ela também colocou, ao longo do caminho, incontáveis desafios que ainda temos que transpor. Neste minicurso, abordamos grandes desafios de duas áreas que mais afligem a revolução UbiComp: segurança e privacidade. Em particular, examinamos problemas abertos em proteção de software, segurança de longo prazo, engenharia de criptografia, implicações de privacidade, resiliência cibernética e gestão de identidade. Nós também apontamos direções promissoras para as soluções desses problemas. Acreditamos que se compreendermos tanto os desafios como os novos rumos corretamente, então conseguiremos tornar a ficção científica de UbiComp em ciência de fato.

[1]Uma versão preliminar deste trabalho foi apresentada em ICCCN 2017 [Oliveira et al. 2017].

1.1. Introdução

Em 1991, Mark Weiser descreveu uma visão do Computador para o Século 21 [Weiser 1991]. Weiser, em seu artigo profético, argumentou que as tecnologias mais abrangentes são aquelas que se tornam transparentes. De acordo com Weiser, esse esquecimento é um fenômeno humano – não tecnológico –: "Sempre que as pessoas aprendem algo suficientemente bem, eles deixam de estar cientes disso", afirmou. Este evento é chamado de "dimensão tácita"ou "compilação"e pode ser testemunhado, por exemplo, quando os motoristas reagem aos sinais de rua sem conscientemente ter que processar as letras P-A-R-E [Weiser 1991].

Um quarto de século depois, no entanto, o sonho de Weiser está longe de se tornar realidade. Ao longo dos anos, muitos dos seus conceitos sobre Computação Ubíqua (UbiComp) [Weiser 1993, Lyytinen and Yoo 2002] foram materializados no que hoje chamamos de Redes de Sensores Sem Fio (*Wireless Sensor Networks* – WSNs [Estrin et al. 1999, Pottie and Kaiser 2000]), Internet das Coisas (*Internet of Things* – IoT [Ashton 2009, Atzori et al. 2010]), Computação Vestível (Wearable Computing [Mann 1997, Martin and Healey 2007]), e Sistemas-Ciber-Físicos (*Cyber-Physical Systems* – CPSs [Lee 2006, Rajkumar et al. 2010]). As aplicações desses sistemas variam de monitoramento de acidentes de trânsito e de emissões de CO_2 para o carro autônomo, a cuidados para pacientes domésticos. No entanto, além de todos os seus benefícios, o advento desses sistemas também provoca vulnerabilidades. E, a menos que as abordemos adequadamente, o futuro da profecia de Weiser estará em jogo.

A UbiComp apresenta novas vulnerabilidades porque exibe uma perspectiva totalmente diferente da computação tradicional [Abowd and Mynatt 2000]. Os elementos computacionais em UbiComp, por exemplo, apresentam sensores, CPU e atuadores. Respectivamente, isso significa que eles conseguem ouvir (ou espionar) você, processar seus dados (e, possivelmente, descobrir algo sigiloso sobre você), e responder às suas ações (ou, em última análise, te expor revelando algum segredo). Estas capacidades, por sua vez, fazem propostas para computadores convencionais inadequados no contexto de UbiComp e, por fim, apresentam novos desafios.

Muitos desses desafios estão nas áreas de Segurança e Privacidade [Stajano 2002]. Isto ocorre porque o mercado e os usuários muitas vezes buscam um sistema cheio de recursos à custa de operação e proteção adequadas; embora, contraditoriamente, na medida que elementos computacionais permeiam nossas vidas, a demanda por esquemas de segurança mais fortes se torna ainda maior [Souza et al. 2017]. Particularmente, existe uma necessidade extrema de um mecanismo de segurança capaz de abarcar todos os aspectos e manifestações de UbiComp, tanto no tempo como no espaço, e de maneira perfeita e eficiente.

Neste minicurso, discutimos questões contemporâneas de segurança e privacidade no contexto da UbiComp. Fazemos uma revisão bibliográfica e examinamos vários problemas de pesquisa ainda em abertos e apresentamos rumos promissores para suas soluções. Mais precisamente, ao longo deste minicurso, investigaremos os desafios abaixo e suas ramificações (vide Figura 1.1).

Figura 1.1. Tópicos abordados no minicurso

Proteção de Software (Seção 1.2) Linguagens de programação são a ferramenta fundamental para a construção de software. Assim, a implementação de sistemas confiáveis e seguros depende, não somente, da habilidade de programadores, mas também das linguagens de programação usadas. Uma das partes deste minicurso irá cobrir essa dependência intrínseca entre linguagens de programação e segurança de software. Iremos analisar as dificuldades que as linguagens fracamente tipadas impõem à construção de software seguro, e estudaremos como programação poliglota. O advento da Internet das Coisas tem tornado a programação poliglota cada vez mais comum. Finalmente, abordaremos também os desafios que a programação distribuída traz para a detecção de vulnerabilidades de software, mostrando como ferramentas modernas podem ser usadas para mitigar tais problemas [Teixeira et al. 2015].

Segurança de Longo Prazo (Seção 1.3) Diversos sistemas de UbiComp são projetados para oferecer uma vida útil de vários anos, até mesmo décadas [Poovendran 2010, Conti 2006]. No contexto de infraestrutura crítica, por exemplo, um enorme investimento financeiro pode ser necessário para que sistemas sejam projetados e efetivamente implantados. Tais sistemas ofereceriam um melhor retorno sobre investimento caso permanecessem em funcionamento por mais tempo. A área automotiva, a qual pertence a UbiComp dado que carros atualmente dispõem de diversos componentes eletrônicos (sendo alguns já conectados à Internet e futuramente ainda mais dependente dessa troca de dados [DoT 2013]), é outra área que demanda sistemas de longa vida útil, uma vez que veículos devem ser funcionais por várias décadas. Renovar frotas veiculares e até mesmo atualizar características pontuais de um veículo (processo conhecido como *recall*) levam ao aumento de custo por parte do usuário, ressaltando a importância de se oferecer soluções robustas por um longo prazo.

Uma maneira efetiva de reduzir a vida útil desses sistemas consiste em comprometer as técnicas de segurança da informação por eles utilizadas. Neste contexto, iremos examinar os principais fatores de risco que ameaçam técnicas criptográficas atuais quando integradas a sistemas com longa expectativa de vida útil. Entre tais ameaças, discutiremos avanços recentes de técnicas de criptanálise, a futura disrupção causada por ataques de computadores quânticos contra criptossistemas convencionais e os novos desafios de implementação de criptossistemas resistentes a ataques de computadores quânticos. Por fim, apresentaremos direções (e inerentes desafios) em como se garantir que sistemas estejam preparados para oferecer segurança de longo prazo em UbiComp.

Engenharia Criptográfica (Seção 1.4). Os sistemas UbiComp envolvem blocos de construção de naturezas muito diferentes: componentes de hardware como sensores e atuadores, software embarcado implementando protocolos de comunicação, interface com provedores de computação em nuvem e, finalmente, procedimentos operacionais e outros fatores humanos. Como resultado, sistemas pervasivos exibem uma ampla superfície de ataque que deve ser protegida utilizando uma combinação de técnicas. A criptografia é peça fundamental nessa proteção, mas é improvável que seja o componente mais fraco. Os protocolos de rede, as rotinas de tratamento de entrada e mesmo o código de interface com mecanismos criptográficos são componentes muito mais propensos a serem vulneráveis à exploração. No entanto, um ataque bem-sucedido em propriedades de segurança criptográficas costuma ser desastroso. Essa seção discutirá boas práticas e problemas de pesquisa no projeto e implementação de mecanismos criptográficos em computação ubíqua, de forma que criptografia cumpra seu papel de representar o componente mais robusto de um sistema computacional moderno.

Resiliência Cibernética (Seção 1.5). A disponibilidade dos serviços essenciais, tais como comunicação fim-a-fim, roteamento e conectividade, na UbiComp é um dos principais pilares da segurança, em conjunto com a confidencialidade e integridade dos dados. A resiliência cibernética visa identificar, prevenir, detectar e responder a processos ou falhas tecnológicas, e recuperar ou reduzir danos e as suas perdas financeiras [Lima et al. 2009]. A violação da disponibilidade dos serviços vem sendo cada vez mais frequente, vide o último ataque de negação de serviço contra a empresa DYN, um dos principais provedores de DNS, e o ataque de negação de serviço contra a OVH, a gigante empresa francesa. O primeiro chegou a um intenso volume de tráfego malicioso de aproximadamente 1 Tbps, gerado a partir de dispositivos que compõem os modernos sistemas UbiComp [ddo]. Como não há nenhuma maneira garantida de evitar estes ataques, as empresas devem se concentrar na criação de soluções mais resistentes aos seus danos e rapidamente retomar a disponibilidade dos serviços. Nos concentraremos nessa seção em apresentar a importância da resiliência cibernética no contexto de sistemas UbiComp e apresentar uma visão geral do estado da arte e direções futuras de pesquisa e inovação [Santos et al. 2017, Macedo et al. 2016, Soto and Nogueira 2015, Lipa et al. 2015].

Gestão de Identidade (Seção 1.6). Um sistema de gestão de identidades (GId) consiste na integração de políticas, processos de negócios e tecnologias de identificação, autenticação, autorização e auditoria. Uma abordagem apropriada de gestão de identidade

(pervasiva) é necessária para que a computação ubíqua seja invisível para usuários [Arias-Cabarcos et al. 2015]. Os sistemas UbiComp são compostos de dispositivos heterogêneos que precisam provar a sua autenticidade para as entidades com as quais se comunicam. Um problema neste cenário é garantir a autenticidade não apenas de usuários mas também de dispositivos que se comunicam entre si e que podem estar localizados em domínios administrativos de segurança diferentes [Wangham et al. 2013]. A segurança naturalmente consome recursos e, assim, acrescentar soluções de IdM em dispositivos embarcados com restrições computacionais pode ser um desafio [Oliveira et al. 2011]. Os dispositivos empregados em sistemas UbiComp, normalmente, atuam no mundo físico. Isso significa que uma ameaça à segurança de um dispositivo pode ter impactos no mundo físico. Ou seja, há o risco de que uma violação influencie o mundo físico ao ponto de atentar contra o bem-estar e até à vida das pessoas [Wu et al. 2017, Domenech et al. 2016]. Nessa seção, os principais desafios para prover GId serão discutidos, por meio da análise das principais IAAs e dos trabalhos recentes que tratam dos problemas de pequisa levantados. Por fim, oportunidades de pesquisa e de inovação nesta área serão indicadas.

Implicações na Privacidade (Seção 1.7) A segurança – ou, em particular, a confidencialidade – é uma condição necessária mas não suficiente para um protocolo garantir Privacidade [Borges 2016b]. Para garantir a segurança, os sistemas UbiComp devem implementar mecanismos de proteção como canais de comunicação seguros, de modo que apenas os remetentes autorizados cifrem mensagens para os receptores autorizados para decrifrá-las. Para garantir a privacidade, é importante determinar as fontes relevantes de vazamento e empregar protocolos de preservação da privacidade para manipular dados confidenciais [Borges de Oliveira 2017b]. Tais protocolos geralmente precisam usar criptossistemas para garantir que o receptor possa obter as informações necessárias sem inferir dados confidenciais sobre o remetente.

Acreditamos que se compreendermos tanto os desafios como os novos rumos corretamente, então conseguiremos tornar a ficção científica da UbiComp em Ciência e realidade de fato.

1.2. Proteção de Software

Os sistemas modernos de computação ubíqua raramente são criados a partir do zero. Os componentes desenvolvidos por diferentes organizações, com diferentes modelos e ferramentas de programação, e sob diferentes pressupostos, são integrados para oferecer recursos complexos aos seus usuários. Essa complexidade traz consigo desafios de segurança, que serão analisados neste capítulo. Para este fim, nesta seção, analisamos o ecossistema de software que caracteriza o campo de computação ubíqua. A Figura 1.2 fornece uma representação de alto nível desse ecossistema. Nos concentramos especialmente em três aspectos desse ambiente, que impõem desafios de segurança aos desenvolvedores: deficiências de segurança de C e C++, as linguagens de programação dominantes em implementações de sistemas ciber-físicos; as interações entre essas linguagens com outras e as consequências dessas interações na natureza distribuída das aplicações ubíquas. Começamos nosso estudo mergulhando mais profundamente nas idiossincrasias das linguagens C e C++.

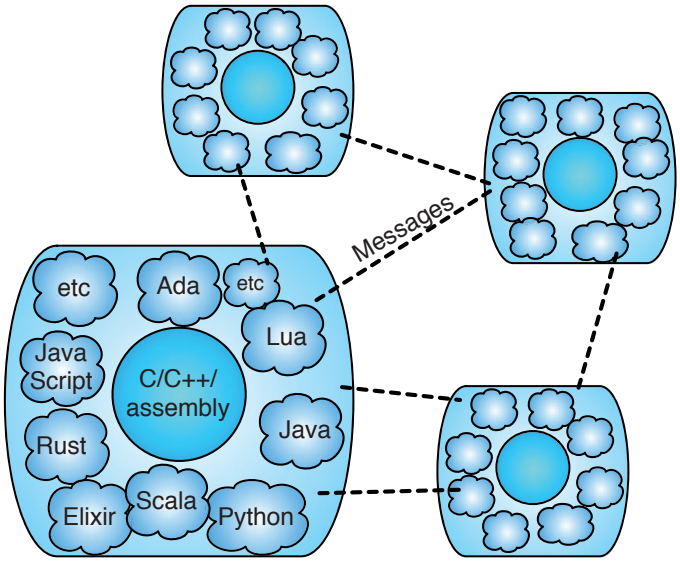


Figura 1.2. Um sistema de computação ubíqua é formado por módulos implementados como uma combinação de diferentes linguagens de programação. Essa diversidade traz consigo desafios para a construção de software seguro.

1.2.1. Segurança de tipos

Uma grande parte do software usado nos sistemas ubíquos é implementado em C ou em C++. Esse fato é natural, dada a eficiência incomparável dessas linguagens de programação. No entanto, se C e C++ são eficientes, por um lado, seu sistema de tipos, conhecido como *tipagem fraca* dá origem a uma infinidade de vulnerabilidades de software. Em jargão de linguagens de programação, dizemos que um sistema de tipo é fraco quando ele não suporta duas propriedades principais: *progresso* e *preservação* [Pierce 2002]. As definições formais dessas propriedades são imateriais para a discussão que segue. Basta saber que, como consequência da tipagem fraca, nem C, nem C++, protegem, por exemplo, a memória contra acessos inválidos. Portanto, os programas escritos nessas linguagens podem ler ou escrever posições inválidas de memória. Como uma ilustração dos perigos incorridos por esta possibilidade, basta lembrar que o acesso fora de limites alocados é o princípio por trás de um tipo de ataque conhecido como estouro de buffer.

A comunidade de segurança de software vem desenvolvendo diferentes técnicas para lidar com as vulnerabilidades intrínsecas de C e C++, e também de linguagens Assembly. Tais técnicas podem ser totalmente estáticas, totalmente dinâmicas ou um híbrido de ambas. Os mecanismos de proteção estática são implementados no nível do compilador; mecanismos dinâmicos são implementados no nível do ambiente de execução. No restante desta seção, listamos os elementos mais conhecidos em cada categoria.

As análises estáticas fornecem uma estimativa conservadora do comportamento do programa, sem exigir a sua execução. Esta ampla família de técnicas inclui, por exemplo, *Interpretação Abstrata* [Cousot and Cousot 1977], *verificação de modelos* [McMillan 1993] e *provas guiadas* [Leroy 2009]. A principal vantagem das análises estáticas é a baixa sobrecarga em tempo de execução, e sua solidez: as proprieda-

des inferidas são garantidas para sempre serem verdadeiras. No entanto, análises estáticas também apresentam desvantagens. Em particular, a maioria das propriedades interessantes dos programas é indecidível [Rice 1953]. Além disso, a verificação de muitas propriedades formais, ainda que decidível, implica um custo computacional proibitivo [Wilson and Lam 1995].

As análises dinâmicas vêm em vários sabores: teste (KLEE [Cadar et al. 2008]), perfilamento (Aprof [Coppa et al. 2012], Gprof [Graham et al. 1982]), execução simbólica (DART [Godefroid et al. 2005]), emulação (Valgrind [Nethercote and Seward 2007]), e instrumentação binária (Pin [Luk et al. 2005]). As virtudes e limitações das análises dinâmicas são exatamente o oposto daquelas encontrados em técnicas estáticas. Análises dinâmicas geralmente não geram falsos alarmes: os erros são descritos por exemplos, que normalmente levam a uma reprodução consistente [Rimsa et al. 2011]. No entanto, elas não precisam sempre encontrar vulnerabilidades de segurança em programas. Além disso, a sobrecarga de tempo de execução das análises dinâmicas ainda as torna proibitivas para serem implantadas em software de produção [Serebryany et al. 2012].

Como um meio termo, vários grupos de pesquisa propuseram formas de combinar análises estáticas e dinâmicas, produzindo diferentes tipos de abordagens híbridas para proteger código de baixo nível. Esta combinação pode render garantias de segurança que são estritamente mais poderosas do que o que poderia ser obtido por técnicas estáticas ou a dinâmicas, quando usado separadamente [Russo and Sabelfeld 2010]. No entanto, os resultados negativos ainda são válidos: se um atacante pode assumir o controle de um programa, geralmente ele ou ela pode contornar proteções híbridas de última geração, tais como a integridade do fluxo de controle [Carlini et al. 2015]. Esse fato é, em última análise, uma consequência do sistema de tipo fraco adotado por linguagens normalmente vistas na implementação de sistemas de computação ubíqua. Portanto, o projeto e implantação de técnicas que podem proteger tais linguagens de programação, sem comprometer sua eficiência, continua a ser um problema aberto.

1.2.2. Programação Poliglota

A *Programação Poliglota* é a arte e a disciplina de escrever código fonte envolvendo duas ou mais linguagens de programação. É comum entre implementações de sistemas ciberfísicos. Como exemplo, Ginga, o protocolo brasileiro para TV digital, é principalmente implementado em Lua e C [Soares et al. 2007]. A Figura 1.3 mostra um exemplo de comunicação entre um programa C e um programa Lua. Outros exemplos de interações entre linguagens de programação incluem ligações entre C e Python [Maas et al. 2016] e C e Elixir [Fedrecheski et al. 2016]. Outro exemplo de programação poliglota é o uso de *Java Native Interface* [Rellermeyer et al. 2008], uma forma de combinar código Java com código escrito em outras linguagens. A programação poliglota complica a proteção de sistemas. Dificuldades surgem devido à falta de ferramentas multilínguas e devido à falta de controle de ligações de memória entre C/C++ e outras linguagens.

Um obstáculo à validação do software Polyglot é a falta de ferramentas para analisar o código-fonte escrito em diferentes linguagens de programação, sob uma estrutura unificada. Retornando à Figura 1.3, temos um sistema formado por dois programas es-

C

```c
#include <stdio.h>
# include <lua5.2/lua.hpp>
// Reads data from Lua, and then sends data to it.
int hello(lua_State* state) {
  int args = lua_gettop(state);
  printf("hello() was called with %d arguments:\n", args);
  for ( int n=1; n<=args; ++n) {
    printf("  arg %d: '%s'\n", n, lua_tostring(state, n));
  }
  lua_pushnumber(state, 123);
  return 1;
}
// Register a call-back function and invoke a Lua program to run it
void execute(const char* filename) {
  lua_State *state = luaL_newstate();
  luaL_openlibs(state);
  lua_register(state, "hello", hello);
  int result = luaL_loadfile(state, filename);
  result = lua_pcall(state, 0, LUA_MULTRET, 0);
}
int main(int argc, char** argv) {
  for (int n=1; n<argc; ++n) { execute(argv[n]); }
}
```

Lua

```lua
io.write("Calling hello() ...\n")
local value = hello("First", "Second", 112233)
io.write(string.format("hello() returned: %s\n", tostring(value)))
```

Figura 1.3. Comunicação bidirecional entre um programa escrito em C e um programa escrito em Lua

critos em diferentes linguagens de programação. Qualquer ferramenta que analise este sistema como um todo deve ser capaz de analisar essas duas sintaxes distintas e inferir os pontos de conexão entre eles. Alguns trabalhos já foram realizados para esse fim, mas as soluções ainda são muito preliminares. Como exemplo, Maas *et al.* [Maas et al. 2016] implementaram modos automáticos para verificar se os arranjos C são lidos corretamente por programas Python. Como outro exemplo, Furr e Foster [Furr and Foster 2008] descreveram técnicas para proteger ligações entre OCaml e C e entre Java e C.

Uma direção promissora para analisar sistemas poliglotas é baseada na idéia de compilação do código fonte parcialmente disponível. Esta façanha consiste na reconstrução da sintaxe faltante, o que inclui as declarações necessárias para produzir uma versão mínima do programa original que pode ser analisado por ferramentas tradicionais. A análise do código parcialmente disponível possibilita testar partes de um programa poliglota em separado, de forma a produzir uma visão coesa do sistema inteiro. Pesquisadores já demonstraram que essa técnica produz código fonte analisável. A título de exemplo, cita-se o trabalho de Dagenais *et al* [Dagenais and Hendren 2008]. Observe que esse tipo de reconstrução não se restringe a linguagens de programação de alto nível. Testemunho deste fato é a noção de *micro-execução*, introduzida por Patrice Godefroid [Godefroid 2014]. A ferramenta construída por Godefroid permite o teste de código binário x86, mesmo

quando arquivos de objeto estão faltando. No entanto, apesar desta evolução, a reconstrução ainda é restrito à semântica estática de programas. A síntese do comportamento é uma disciplina próspera no computador Ciência [Manna and Waldinger 1971], mas ainda está longe de permitir a certificação de sistemas poliglotas.

1.2.3. Programação Distribuída

Os sistemas de computação ubíquos tendem a ser distribuídos. É mesmo difícil conceber qualquer uso para uma aplicação neste mundo que não interage com outros programas. E, é de conhecimento geral que a programação distribuída abre várias portas para usuários mal-intencionados. Portanto, para tornar a tecnologia ciberfísica mais segura, as ferramentas de segurança devem ser conscientes da natureza distribuída de tais sistemas. No entanto, dois desafios principais estão diante dessa exigência: a dificuldade de construir uma visão holística da aplicação distribuída e a falta de informação semântica vinculada a mensagens trocadas entre processos que se comunicam através de uma rede.

Para ser eficaz, a análise de um sistema distribuído precisa ser responsável pela interações entre as diversas partes do programa que constituem esse sistema [López et al. 2015]. Descobrir tais interações é difícil, mesmo quando nos restringimos à código escrito em uma única linguagem de programação. As dificuldades advêm da falta de informação semântica associada às operações que enviam e recebem mensagens. Em outras palavras, tais operações são definidas como parte de uma biblioteca, não como parte da linguagem de programação em si. Não obstante este fato, existem várias técnicas que inferem canais de comunicação entre diferentes partes do código-fonte. Como exemplos, temos os algoritmos de Greg Bronevetsky [Bronevetsky 2009], E Teixeira *et al.* [Teixeira et al. 2015], que criam uma visão distribuída do gráfico de fluxo de controle de um programa (CFG[2]). A beleza de tais técnicas vêm do fato de que as análises estáticas clássicas funcionam sem mais modificações nesse *CFG distribuído*. No entanto, o CFG distribuído ainda é uma aproximação conservadora do comportamento do programa. Assim, ele ainda leva análises estáticas, já imprecisas, a terem de lidar com canais de comunicação que talvez nunca existam durante a execução real de um programa.

As ferramentas que realizam análises automáticas em programas dependem de informações estáticas para produzir resultados mais precisos. Nesse sentido, os tipos são fundamentais para a compreensão dos programas. Por exemplo, em Java e outras linguagens de programação orientadas a objetos, o tipo de objetos determina como a informação flui ao longo do código do programa. No entanto, apesar desta importância, as mensagens trocadas na grande maioria dos sistemas distribuídos não possuem tipos. A razão para isso é o fato de que tais mensagens, pelo menos em C, C++ e linguagens de montagem, são arranjos de *bytes*. Para mitigar esse problema, pesquisadores vêm desenvolveram análises que inferem o conteúdos [Cousot et al. 2011] e o tamanho de arranjos [Nazaré et al. 2014] em linguagens de programação fracamente tipadas. No entanto, muitos problemas ao longo dessa direção de pesquisa permanecem abertos. Por exemplo, a análise de arranjos em linguagens de programação de baixo nível requer frequentemente a capacidade de lidar com a aritmética do ponteiro. Análises estáticas que lidam com ponteiros ainda sofrem severas restrições que dificultam a análise de programas re-

[2]Sigla transcrita do inglês *Control Flow Graph*

ais [Paisante et al. 2016]. Portanto, o projeto de ferramentas capazes de analisar software usado em sistemas de computação ubíqua ainda é um problema aberto.

1.3. Segurança de Longo Prazo

Diversos sistemas de UbiComp são projetados para oferecer uma vida útil de vários anos, até mesmo décadas [Poovendran 2010, Conti 2006]. Por exemplo, sistemas no contexto de infraestrutura crítica podem necessitar de um enorme investimento financeiro para serem projetados e efetivamente implantados [Rinaldi et al. 2001]. Logo, tais sistemas ofereceriam um melhor retorno sobre investimento caso pudessem permanecer em uso por um maior período de tempo. A área automotiva é outra área de particular interesse no que concerne a segurança de longo prazo. Veículos devem ser funcionais e confiáveis por vários anos, geralmente décadas [BTS 2017]. A renovação de frotas de veículos e até mesmo a atualização de algumas de suas características (*recall*) implicam, obviamente, em aumento de custos para seus usuários. Note que veículos fazem parte do ecossistema UbiComp pois são equipados com vários dispositivos eletrônicos embarcados, dos quais alguns já oferecem conectividade à Internet. Futuramente, tais veículos dependerão ainda mais fortemente de dados coletados e compartilhados por meio de tecnologias de rede sem fio [DoT 2013] a fim de oferecer experiências mais ricas de condução, tais como as esperadas por veículos autônomos [Maurer et al. 2016].

Vale também ressaltar que os sistemas concebidos de forma a oferecer uma vida útil de vários anos ou décadas podem estar sujeitos à escassez de manutenções futuras. Por exemplo, a concorrência na indústria de tecnologia é bastante agressiva, levando assim a uma alta taxa de empresas que saem do mercado dentro de poucos anos [Patel 2015]. Logo, o cenário de um mundo inundado por dispositivos dos quais ninguém é responsável pelo seu bom funcionamento e operação adequada certamente oferecerá importantes desafios [Jacobsson et al. 2016].

Os exemplos mencionados acima, dentre muitos outros possíveis, ilustram um crescente interesse em se projetar sistemas de UbiComp que sejam confiáveis por mais tempo e, sempre que possível, exigindo o menor número possível de atualizações. Tais requisitos têm um impacto direto sobre os mecanismos de segurança desses sistemas: um número relativamente menor de oportunidades para corrigir eventuais brechas de segurança do que em sistemas convencionais. Esta é uma situação crítica dado o intenso e dinâmico campo de pesquisa de criação e exploração de brechas de segurança. Portanto, é de extrema importância entender quais são os desafios em se prover mecanismos de segurança de longo prazo desde o início do projeto de um sistema para, desta forma, não precisar recorrer a medidas paliativas *a posteriori*.

1.3.1. A Criptografia como Componente Central

Garantir segurança de longo prazo é uma tarefa bastante desafiadora para qualquer sistema, não apenas para os sistemas de UbiComp. No mínimo, isso exige que cada componente de segurança seja à prova de ataques futuros tanto individualmente quanto quando considerando a sua integração/conexão com outros sistemas. O ingrediente fundamental dos mecanismos de segurança é a Criptografia, como será destacado na Seção 1.4 e, portanto, esse será o foco de nossa avaliação.

As técnicas de criptografia tradicionais dependem de problemas computacionais, tais como a fatoração de números inteiros [Rivest et al. 1978] e o problema do logaritmo discreto [Miller 1985, Koblitz 1987]. Acredita-se que esses problemas sejam intratáveis pelas técnicas de criptoanálise e recursos tecnológicos atuais, o que possibilitou alguns criptógrafos a criarem instâncias atualmente seguras destes criptossistemas. Por várias razões (a serem discutidas a seguir), no entanto, tais construções tendem a oferecer sérios riscos em um futuro não tão distante.

1.3.2. Avanços em Criptoanálise Clássica

A primeira ameaça para a segurança de longo prazo de qualquer criptossistema se refere a potenciais avanços em criptoanálise, ou seja, em técnicas que visam a resolver o problema de segurança subjacente a esses criptossistemas de forma mais eficiente do que atualmente (e.g., com menor tempo de processamento, menos memória, entre outros). Esquemas amplamente utilizados oferecem uma longa história de escrutínio acadêmico e industrial e, portanto, seria de se esperar pouco ou nenhum progresso nas técnicas de criptoanálise desses esquemas. No entanto, a literatura mostra alguns resultados recentes interessantes que podem sugerir o oposto.

Em [Barbulescu et al. 2014], por exemplo, os autores introduziram um novo algoritmo de complexidade quase-polinomial para resolver o problema do logaritmo discreto em corpos finitos de pequenas características. Vale relembrar que se espera que problemas matemáticos subjacentes de criptossistemas tenham complexidade exponencial para um atacante não dispondo de alçapões para a resolução de tal problema, como a chave privada associada. O problema do logaritmo discreto é, de maneira geral, o problema de segurança subjacente do algoritmo de troca de chaves *Diffie-Hellman* [Diffie and Hellman 2006], do algoritmo de assinatura digital DSA [NIST 2013] e de suas variantes em curva elíptica (ECDH [NIST 2006] e ECDSA [NIST 2013], respectivamente), apenas para mencionar alguns criptossistemas amplamente implantados. É importante ressaltar que o resultado criptoanalítico mencionado acima é restrito a corpos finitos de pequena característica, o que representa uma limitação importante para atacar implementações reais dos esquemas mencionados. No entanto, qualquer algoritmo sub-exponencial que resolva um problema conhecido de longa data pode ser visto como uma indicação relevante de que a literatura em criptoanálise está, e sempre estará, sujeita a eventuais avanços importantes.

Tais observações sobre futuros avanços em criptoanálise devem ser consideradas pelos arquitetos de sistemas de UbiComp que tenham a segurança de longo prazo como um pré-requisito. Por exemplo, implementações que suportem vários níveis de segurança são preferíveis quando comparadas àquelas que suportem apenas um nível de segurança (e.g., tamanho de chave) fixo. Desta forma, os sistemas de UbiComp devem conscientemente acomodar recursos para lidar com futuros avanços em criptoanálise ou, pelo menos, reduzir os custos referentes a atualizações de segurança.

1.3.3. Futura Disrupção Devido A Ataques Quânticos

Espera-se que computadores quânticos ofereçam melhorias dramáticas às técnicas para resolver certos problemas computacionais, conforme mostrado por Daniel R. Simon em seu artigo seminal sobre algoritmos quânticos [Simon 1994]. Por um lado, tal área de

pesquisa poderá promover avanços consideráveis em diversas tecnologias atualmente limitadas devido à ineficiência de seus algoritmos [Knill 2010]. Entretanto, por outro lado, alguns dos problemas afetados são utilizados para proteger criptossistemas amplamente implantados e que, portanto, deveriam ser sempre de difícil resolução.

Em 1996, Lov K. Grover propôs um algoritmo quântico [Grover 1996] capaz de encontrar com alta probabilidade um elemento no domínio de uma função (a qual tenha N possíveis saídas) que leva a uma saída específica em apenas $O(\sqrt{N})$ passos. Este algoritmo é extremamente importante, uma vez que pode ser utilizado para acelerar a criptoanálise da vários criptossistemas simétricos.

Devido ao algoritmo de Grover, cifras de bloco com chave de n bits, por exemplo, oferecerão apenas $n/2$ bits de segurança contra um adversário quântico, ao invés dos n bits de segurança contra um adversário clássico (não-quântico). As funções de hash também serão afetadas por esse ataque de diferentes formas, dependendo da propriedade de segurança almejada. Por exemplo, funções de hash com saída de tamanho n bits oferecerão apenas $n/3$ bits de segurança contra ataques de colisão e $n/2$ bits de segurança contra ataques de pré-imagem, ao considerar um adversário quântico. Isso seria um nível de segurança sensivelmente menor do que contra um adversário clássico: $n/2$ e n para resistência à colisão e pré-imagem, respectivamente. Tabela 1.1 resume esta avaliação. Neste contexto, AES-128 e SHA-256 (considerando resistência à colisão) não atingiriam o nível de segurança considerado mínimo (128 bits de segurança) e, portanto, não deveriam ser utilizados. É importante ressaltar, no entanto, que tanto as construções de cifras de bloco quanto de funções de hash sobreviverão a esse ataque quântico desde que tamanhos apropriados de chaves e de saída de função de hash sejam utilizados (e.g., AES-256 e SHA-384). Infelizmente, chaves e saída de função de hash maiores acarretarão em importantes desafios de desempenho. AES-256, por exemplo, é cerca de 40% menos eficiente do que o AES-128 (devido ao diferente número de *rounds*, 14 ao invés de 10).

Algoritmo	**Segurança Clássica**	**Segurança Quântica**
Cifra de Bloco (n bits)	n	$n/2$
Hash Pré-Imagem (n bits)	n	$n/2$
Hash Colisão (n bits)	$n/2$	$n/3$

Tabela 1.1. Níveis de Segurança para Criptografia Simétrica

Ainda mais crítico do que o cenário para a criptografia simétrica será o cenário para a criptografia assimétrica (também conhecida como criptografia de chaves públicas). Os computadores quânticos oferecerão uma vantagem exponencial para atacar a maioria dos criptossistemas de chave pública utilizados em larga escala atualmente. Isto ocorrerá devido ao algoritmo de Peter Shor [Shor 1997] que permite fatorar eficientemente (em tempo polinomial) números inteiros grandes e também calcular o logaritmo discreto de um elemento em grandes grupos. O impacto deste trabalho será devastador para esquemas tais como RSA e ECC, por exemplo, pois o aumento de tamanho de chaves não resolveria efetivamente esta fragilidade. Em resumo, tais criptossistemas de chaves públicas baseados em fatoração e logaritmo discreto precisarão ser completamente aposentados e, portanto, substituídos.

Existem alternativas criptográficas que são resistentes a ataques quânticos. No entanto, diversos desafios ainda precisam ser superados. O primeiro se refere ao estabelecimento de um consenso na academia e na indústria sobre quais alternativas são as mais promissoras. De maneira geral, existem duas principais técnicas que oferecem funcionalidades similares à criptografia de chaves públicas e que são capazes de sobreviver a ataques quânticos, a saber: criptografia pós-quântica (PQC, do acrônimo em inglês *Post-Quantum Cryptography*) e criptografia quântica. O primeiro é baseado em problemas computacionais clássicos que devem ser difíceis mesmo para computadores quânticos. Os esquemas PQC podem ser implementados nos computadores atuais [McEliece 1978, Merkle 1979, Regev 2005, Buchmann et al. 2011, McGrew et al. 2017], apenas para mencionar alguns poucos trabalhos. Já a outra alternativa (criptografia quântica) depende de toda uma infra-estrutura quântica a ser implantada e pode ser usado principalmente para fins de troca de chaves [Bennett and Brassard 1984]. A dificuldade e os altos custos da implantação de infra-estrutura quântica necessários à criptografia quântica favorecem o crescente interesse em *criptografia pós-quântica*.

Os criptossistemas pós-quânticos são baseados em diferentes problemas computacionais. No contexto de assinaturas digitais, assinaturas baseadas em hash são soluções bastante promissoras. Tais esquemas [Buchmann et al. 2011, McGrew et al. 2017] são variantes do esquema de assinaturas baseadas em hash proposto por Ralph C. Merkle [Merkle 1979] e sua segurança depende exclusivamente de certas propriedades já bastante estudadas de funções hash. Como funções de hash são resistentes a ataques de computadores quânticos (dependendo do tamanho da saída, conforme discutido acima), tais esquemas de assinaturas também serão. Diferentemente do contexto de assinaturas digitais, outras funcionalidades tais como esquemas de troca de chaves e criptografia assimétrica, ainda não têm um consenso na comunidade.

Ainda não existem padrões para criptossistemas pós-quânticos. Entretanto, esforços de padronização já foram iniciados tais como o liderado pelo *National Institute of Standards and Technology* (NIST) [NIST 2016]. Este processo deve levar pelo menos mais alguns anos até ser concluído. Note que a ausência atual de padrões pode representar um desafio de interoperabilidade aos sistemas que precisam oferecer segurança de longo prazo.

Outro desafio no contexto dos criptossistemas pós-quânticos de chave pública se refere aos diferentes requisitos de implementação. Como mencionado anteriormente, as assinaturas baseadas em hash são alternativas pós-quânticos interessantes (em termos de eficiência e segurança associada somente a funções de hash), mas também trazem um conjunto de desafios de implementação completamente novo. Em particular, assinaturas baseadas em hash são esquemas classificados como *stateful*, o que nesse contexto significa que, entre a emissão de duas assinaturas, o sistema precisa atualizar a sua chave privada. Caso políticas robustas de proteção e gerenciamento desse *state* não estejam em vigor, um assinante poderá forçar com que chave privada não seja atualizada, algo que anularia totalmente as garantias de segurança oferecidas pelo esquema. Recentemente, trabalhos iniciais para enfrentar esses novos desafios de implementação começaram a aparecer na literatura [McGrew et al. 2016]. Também relacionado a esse ponto, uma construção recente [Bernstein et al. 2015] mostrou como construir assinaturas baseadas em hash sem *state*. Entretanto, essa proposta vem associada com um importante aumento no tamanho

das assinaturas. Estes exemplos indicam que os novos criptossistemas trarão novos desafios de implementação, os quais devem ser corretamente considerados pelos arquitetos de sistemas de UbiComp.

1.4. Engenharia Criptográfica

A criptografia é uma peça fundamental na proteção de qualquer sistema computacional moderno, mas raramente o componente mais fraco em sua superfície de ataque. Enquanto essa posição é normalmente ocupada por fatores humanos e vulnerabilidades de *software*, mecanismos criptográficos concentram risco e um ataque com sucesso às suas propriedades termina sendo desastroso. Por exemplo, violações de confidencialidade causadas por mecanismos fracos de encriptação podem causar vazamentos massivos de dados envolvendo informação sensível. A interferência maliciosa na integridade de comunicação pode permitir ataques de injeção de comandos que forçam o sistema a desviar do comportamento esperado. A disponibilidade dos serviços é também crítica para manter o sistema acessível a usuários legítimos e garantir fornecimento contínuo do serviço, portanto mecanismos criptográficos precisam também ser leves para minimizar o potencial de abuso por atacantes.

O acesso físico por adversários a partes da superfície de ataque são um aspecto particularmente desafiador de se implementar criptografia em sistemas ubíquos. Por construção, os adversários podem ser capazes de recuperar chaves de longa duração e credenciais que terminam por fornecer certo controle sobre porções do sistema. A seguir alguns dos principais desafios na engenharia de mecanismos criptográficos serão explorados no contexto de computação ubíqua, incluindo cuidados com implantação segura, soluções para gerência de chaves e implementação eficiente de criptografia.

1.4.1. Boas Práticas

A segurança real de uma técnica criptográfica depende de uma série de premissas, que precisam ser satisfeitas para a técnica funcionar a contento. O projeto teórico de uma técnica criptográfica começa com a detecção de um *problema* computacionalmente difícil, que pode variar desde a simples seleção de uma cadeia de bits dentro de um espaço grande de chaves, caso da criptografia simétrica, até problemas difíceis em Teoria dos Números em que se baseia a criptografia assimétrica. Após detectado um problema computacionalmente difícil, é importante projetar um *algoritmo* criptográfico que utiliza o problema como fonte de confiança; e um *protocolo* em que múltiplas partes executam diferentes algoritmos para obter algum tipo de funcionalidade conjunta. A análise formal de algoritmos e protocolos permite relacionar a segurança fornecida por estes com a dificuldade de resolver os problemas subjacentes. Considerando aspectos mais aplicados de engenharia, o esquema para distribuição de chaves determina como chaves criptográficas são geradas, entregues a cada uma das partes participantes e posteriormente armazenadas. Por fim, o protocolo precisa ser concretizado em uma implementação em software ou hardware para ser implantado em produção. O ciclo de vida típico de uma aplicação criptográfica está ilustrado na Figura 1.4.

Há muitos cuidados que precisam ser tomados durante a implantação de uma técnica criptográfica:

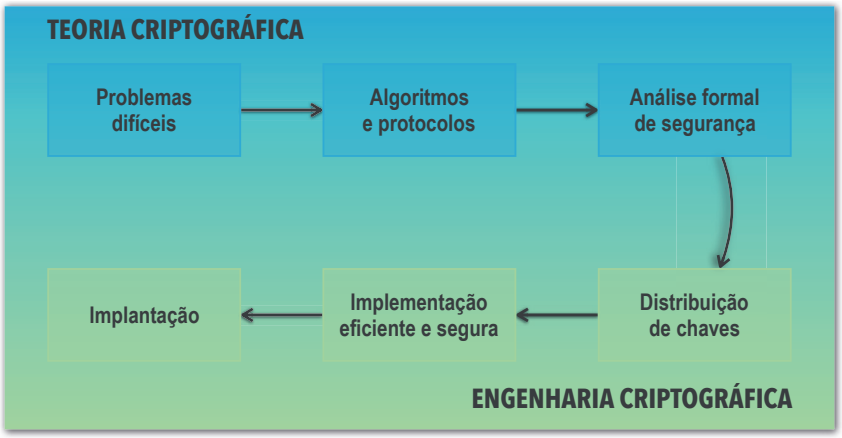


Figura 1.4. Tópicos abordados no minicurso

- **Geração de chaves:** uma fonte de entropia de alta qualidade precisa ser determinada na plataforma alvo para geração de números aleatórios. A escolha de gerador tipicamente depende da combinação de linguagem de programação, sistema operacional e bibliotecas de suporte, mas tem se tornado mais fácil em plataformas modernas. Não é preciso dizer que a prática comum de utilizar tomadas de tempo ou outros valores previsíveis é altamente desaconselhada [Aranha et al. 2014]. Recomendações incluem a utilização do gerador `/dev/urandom` do sistema operacional GNU/Linux ou da instrução `RDRAND` introduzida recentemente pela Intel.

- **Escolha de algoritmos e parâmetros:** é fundamental determinar com precisão quais as propriedades dentre sigilo, autenticação e integridade precisam ser fornecidas pelo sistema; e posteriormente quais os algoritmos que serão responsáveis por fornecer essas propriedades. Encriptação utilizando cifras de bloco e fluxo fornecem confidencialidade, funções de resumo fornecem integridade; autenticadores e assinaturas digitais fornecem autenticação de mensagem e origem, respectivamente. Resta definir quais os tamanhos de chaves e parâmetros para esses algoritmos, após exame cuidadoso do conhecimento na literatura e estimativas de dificuldade de solução dos problemas subjacentes [3].

- **Escolha de bibliotecas criptográficas:** é preferível reusar implementações disponíveis a customizar implementações de terceiros. Utilizar bibliotecas criptográficas como a NaCl[4], com interfaces amigáveis que demandem menor conhecimento especializado do desenvolvedor também é recomendável [Bernstein et al. 2012b].

- **Distribuição de chaves:** é desafiador determinar com chaves criptográficas serão distribuídas entre os múltiplos participantes. É importante evitar o compartilhamento direto de chaves simétricas ou sua inserção direta em partes desprotegidas do sistema, sendo favorável utilizar protocolos de acordos de chaves baseados em criptografia assimétrica [Oliveira et al. 2011]. Desta forma, o problema de compartilhamento de segredo se converte na obtenção de chaves públicas autênticas,

[3] https://www.keylength.com
[4] https://nacl.cr.yp.to/

possivelmente tratado com o uso de certificados digitais rigorosamente validados. Outra alternativa é transformar o problema de compartilhamento de segredo em autenticação, onde um nó restrito em recursos obtém acesso a uma chave criptográfica armazenada em um nó mais poderoso após satisfazer requisitos de autenticação, talvez com ajuda do usuário final.

A seguir, algumas das questões acima são tratadas em maior nível de detalhe, como gerência de chaves criptográficas, recomendações de algoritmos criptográficos leves e implementação segura.

1.4.2. Gerência de Chaves

Por definição, sistemas ubíquos são plataformas heterogênas, conectando dispositivos de poder computacional e capacidade de armazenamento massivamente distintas. Projetar uma arquitetura criptográfica para qualquer sistema heterogêneo requer determinar claramente os papéis e propriedades de segurança para cada entidade no sistema. Dispositivos carentes em recursos devem receber tarefas computacionalmente menos intensivas para executar, e sua falta de proteções contra ataques físicos indicam que segredos de longo prazo não devem residir nesses dispositivos. Tarefas mais críticas envolvendo criptografia de chave pública, como acordos de chaves e assinaturas digitais, devem ser portanto delegadas aos componentes mais poderosos em recursos disponíveis. Um compromisso cuidadoso entre propriedades de segurança, funcionalidade e primitivas criptográficas deve então ser construído para cada dispositivo ou classe de dispositivos [Barker et al. 2012], levando em consideração uma série de recomendações:

- **Eficiência:** protocolos de gerência de chaves devem ser leves e fáceis de implementar, direcionando a interface com Infra-estruturas de Chaves Públicas (ICPs) e operações custosas de tratamento de certificados digitais para os componentes mais poderosos da arquitetura. Modelos alternativos com suporte a certificação implícita que evitam esses problemas incluem *criptografia baseada em identidades* [Boneh and Franklin 2003] e *sem certificados* [Al-Riyami and Paterson 2003], ressaltando que a utilização de criptografia baseada em identidades implica custódia inerente de chaves criptográficas por uma autoridade central. As dificuldades com revogação de chaves ainda impõem obstáculos para adoção irrestrita desses modelos, apesar do progresso recente no projeto desses mecanismos [Boldyreva et al. 2012].

- **Funcionalidade:** protocolos para gerência de chaves devem gerenciar todo o tempo de vida de chaves criptográficas e garantir acessibilidade apenas para usuários autorizados em um certo instante de tempo. Entretanto, gerenciar chaves criptográficas e autorizar usuários em processos separados pode aumentar a complexidade e introduzir vulnerabilidades. Uma forma promissora de se combinar os dois serviços de segurança com controle de acesso de forma a satisfazer garantias criptográficas formais é a *criptografia baseada em atributos* [Waters 2011, Liu and Wong 2016], onde chaves possuem um conjunto de atributos que descrevem capacidades passíveis de autorização ou revogação sob demanda.

- **Comunicação:** componentes devem minimizar a quantidade de tráfego transmitido, sob risco de se tornarem inoperantes caso o meio de comunicação seja atacado ou se torne indisponível. Abordagens não-interativas para distribuição de chaves [Oliveira et al. 2011] são desejáveis aqui, mas protocolos avançados baseados em emparelhamentos bilineares sobre curvas elípticas devem ser cautelosamente evitados após avanços recentes no cálculo do logaritmo discreto [Kim and Barbulescu 2016]. Esses avanços forçam o aumento no tamanho dos parâmetros e reduzem desempenho e escalabilidade, favorecendo formas mais tradicionais de criptografia assimétrica.

- **Interoperabilidade:** sistemas pervasivos são compostos por componentes com naturezas distintas e origens em múltiplos fabricantes. Desta forma, mecanismos de autenticação e autorização precisam suportar diferentes domínios de aplicação para fornecer interoperabilidade entre esses dispositivos [Neto et al. 2016].

Primitivas criptográficas que participam em qualquer tipo de funcionalidade conjunta fornecida por um sistema ubíquo devem então ser compatíveis com todos os componentes e respeitar as limitações de dispositivos equipados com menos recursos.

1.4.3. Criptografia Leve

O surgimento de ecossistemas gigantescos de dispositivos conectados motiva o desenvolvimento de primitivas criptográficas adequadas a esse cenário, sob o termo *criptografia leve*. Esse termo não denota criptografia *fraca*, mas técnicas criptográficas customizadas para uma aplicação específica e cuidadosamente projetadas para serem eficientes em termos do consumo de recursos como ciclos de processamento, capacidade energética e armazenamento em memória [Mouha 2015]. Técnicas criptográficas leves procuram satisfazer requisitos de segurança usuais para algoritmos criptográficos de sua classe, mas podem adotar escolhas menos conservadoras de projeto ou construções mais recentemente introduzidas na literatura.

Como um primeiro exemplo, várias cifras de bloco para encriptação simétrica foram propostas como alternativas leves ao algoritmo *Advanced Encryption Standard* (AES) [Daemen and Rijmen 2002]. Construções importantes são *LS-Designs* [Grosso et al. 2014], redes modernas de Feistel e Adição-Rotação-Soma (ARX) [Dinu et al. 2016], e redes de substituição-permutação [Albrecht et al. 2014, Beierle et al. 2016]. Um candidato notável é a cifra de bloco PRESENT, que vem se tornando mais eficiente [Reis et al. 2017] mesmo com a maturidade alcançada após resistir a 10 anos de criptanálise [Bogdanov et al. 2007].

No caso de funções de resumo criptográficas, um projeto pode trocar propriedades de segurança avançadas (como resistência a colisões) por simplicidade em alguns cenários. Um exemplo claro dessa idéia é a construção de códigos curtos para autenticação de mensagens (*Message Authentication Codes – MACs*) a partir de funções de resumo que não são resistentes a colisões, com no caso do algoritmo SipHash [Aumasson and Bernstein 2012], ou assinaturas digitais construídas a partir de funções de resumo com entradas curtas [Kölbl et al. 2016]. Em aplicações convencionais que exigem resistência a colisões, a função BLAKE2 [Aumasson et al. 2013] tem se apre-

sentado na prática como candidata mais interessante para substituir a função padronizada SHA-1 recentemente quebrada [Stevens et al. 2016] do que o próprio padrão SHA-3 [].

Outra tendência importante é fornecer confidencialidade e autenticação em um único passo, por meio de algoritmos para cifração autenticada (*Authenticated Encryption with Associated Data – AEAD*). Essa técnica pode ser implementada como um modo de operação de uma cifra de bloco (*Galois Counter Mode – GCM* [McGrew and Viega 2004]) ou um projeto dedicado. Espera-se que a competição CAESAR [5] selecione um novo algoritmo AEAD para padronização até o final do ano. A derivação de chaves simétricas a partir de senhas pode ser realizada por uma função de derivação de chaves criptográficas que consome quantidades configuráveis de ciclos e memória, como o algoritmo Argon2 recentemente selecionado como finalista na competição *Password Hashing Competition* [Biryukov et al. 2016].

Em termos de criptografia assimétrica, criptografia de curvas elípticas (*Elliptic Curve Cryptography – ECC*) [Miller 1985, Koblitz 1988] continua sendo o principal candidato para aplicação nesse espaço, em contraste com criptossistemas baseados na fatoração de inteiros [Rivest et al. 1978]. As principais vantagens vêm da complexidade conjecturada como completamente exponencial do problema do logaritmo discreto em curvas elípticas, que permite diminuir o tamanho de chaves criptográficas e parâmetros de domínio. Instanciações modernas de ECC fornecem alto desempenho e simplicidade de implementação absolutamente compatíveis com sistemas embarcados [Bernstein 2006, Bernstein et al. 2012a, Costello and Longa 2015].

A longo prazo, a dominância de primitivas criptográficas com dificuldade baseada em problemas de Teoria dos Números é entretanto ameaçada por computadores quânticos, conforme descrito na Seção 1.3. É importante notar que essa miríade de novas primitivas sendo propostas precisa ser rigorosamente avaliada tanto do ponto de vista de segurança quanto desempenho, requerendo tanto trabalho teórico quanto aspectos de engenharia. Implementações devem consumir quantidades cada vez menores de energia [Banik et al. 2015], ciclos de processamento e memória [Dinu et al. 2015] em dispositivos cada vez menores sob ataques cada vez mais invasivos.

1.4.4. Resistência a Ataques de Canal Lateral

Caso implementados sem cuidado, um algoritmo criptográfico ou protocolo originalmente seguro pode vazar informação crítica útil a um adversário. Ataques de canal lateral [Kocher 1996] são uma ameaça significativa à segurança criptográfica e podem utilizar informação de tempo, latência da hierarquia de memória, consumo de energia e emanações eletromagnéticas para recuperar segredos criptográficos. Esses ataques emergem da interação entre implementação e arquitetura computacional subjacente e representam um problema de segurança intrínseco a ambientes de computação pervasiva, já que assume-se que o atacante possui acesso físico irrestrito a pelo menos alguns dos dispostiivos legítimos em operação.

Proteger contra ataques de canal lateral é um problema de pesquisa desafiador e contramedidas tipicamente fornecem algum grau adicional de regularidade du-

[5] `https://competitions.cr.yp.to/caesar.html`

rante a computação. Implementações *isócronas* ou em tempo constante estão entre as primeiras estratégias de defesa para esses problema de segurança quando incidem sobre variações no tempo de execução ou tempo de resposta da hierarquia de memória. Técnicas simples envolvem eliminação de desvios condicionais com condição baseada em informação secreta e endereçamento de tabelas utilizando índices calculados a partir de segredos, dado que estas podem ou não estar armazenadas em memória *cache* [Faz-Hernández et al. 2015]. A aplicação de métodos formais tem desenvolvido as primeiras ferramentas para análise estática de execução em tempo constante da implementação, como análise de fluxo de informação [Rodrigues et al. 2016] e transformações entre programas [Almeida et al. 2016]. Análise dinâmica é normalmente realizada com heurísticas [Reparaz et al. 2017, Langley 2010], mas igualmente importante para detectar variações de tempo introduzidas pelo compilador [Kaufmann et al. 2016] ou mesmo execução do conjunto de instruções em uma certa arquitetura [Großschädl et al. 2009, Pornin 2017].

Enquanto há uma tendência recente na direção de construir e padronizar algoritmos criptográficos com alguma resistência embutida contra ataques simples de tempo ou potência [Grosso et al. 2014], ataques mais poderosos como análise diferencial de potência [Kocher et al. 1999] ou injeção de falhas [Biham and Shamir 1997] são difíceis de prevenir ou mitigar para implementações em software. Em particular, ataques de injeção de falhas se tornaram muito mais poderosos após serem demonstrados como de execução factível em software [Kim et al. 2014].

Técnicas de mascaramento [Ishai et al. 2003] são frequentemente investigadas como uma contramedida para eliminar a correlação entre informação vazada e informação secreta, mas frequentemente requerem fontes robustas de entropia para atingir seu objetivo. Reciclagem de aleatoriedade tem sido útil como uma heurística para suavizar esse requisito, mas análise formal da seguranças dessa abordagem é um problema em aberto [Balasch et al. 2014]. Modificações na arquitetura subjacente para introduzir extensões no conjunto de instruções, simplificar o ambiente de execução ou fornecer mecanismos transacionais para reiniciar computação manipulada são outra direção promissora de pesquisa, mas podem envolver alterações radicais às arquiteturas computacionais em uso, talvez proibitivas para sistemas embarcados restritos em recursos.

1.5. Resiliência Cibernética

A disponibilidade de serviços essenciais no contexto da computação ubíqua (UbiComp) é um dos principais pilares da segurança, juntamente com a confidencialidade e a integridade dos dados. A resiliência visa identificar, prevenir, detectar e responder a falhas tecnológicas ou processuais, a fim de recuperar ou reduzir danos e perdas financeiras [Lima et al. 2009]. A violação da disponibilidade do serviço está se tornando cada vez mais frequente e disruptiva. Um exemplo é o último ataque de Negação de Serviço Distribuído (DDoS) contra a provedora de nomes DYN e o ataque DDoS contra a empresa OVH, o gigante provedor francês de hospedagem de sites. O primeiro atingiu um volume intenso, e inédito, de tráfego malicioso de aproximadamente 1.2 Tbps, gerado a partir de uma grande quantidade de dispositivos geograficamente distribuídos e infectados, como impressoras, câmeras IP, *gateways* residenciais e monitores para bebês, que compõem os modernos sistemas UbiComp [ddo]. Como não existe uma maneira garantida

de evitar esses ataques, as soluções resilientes tornam-se uma forma de mitigar os danos e retomar rapidamente a disponibilidade de serviços. Esta seção se concentra em apresentar a importância da resiliência no contexto dos sistemas UbiComp, aborda o estado da arte, os requisitos de resiliência, e aponta as futuras direções para pesquisa e inovação [Santos et al. 2017, Macedo et al. 2016, Soto and Nogueira 2015, Lipa et al. 2015].

As melhorias nas tecnologias da informação e da comunicação, como as redes sem fio, aumentaram a importância dos sistemas distribuídos em nossas vidas diárias. A primeira década do século 21 testemunhou o aumento dos sistemas de grande escala, como as redes sociais de escala global, os mercados globais de publicidade, e o comércio global [Delic 2016]. O acesso à rede está se tornando onipresente através de dispositivos portáteis e comunicações sem fio, tornando as pessoas cada vez mais dependentes dela. Isso aumenta a demanda por um alto nível de confiabilidade, disponibilidade e segurança simultânea em transações comerciais, financeiras, médicas e sociais suportadas pela UbiComp.

A Internet de coisas (IoT), as redes veiculares e as redes de corporais sem fio são exemplos de sistemas distribuídos comuns e que compõem a UbiComp. Essas redes são compostas por dispositivos portáteis heterogêneos, comunicando-se entre eles, em geral, de maneira multi-salto e através de tecnologias de comunicação sem fio [Zhang et al. 2008]. Os pesquisadores vêm trabalhando para que essas redes sem fio possam se adaptar de forma autônoma às mudanças em seu ambiente, como por exemplo às mudanças na posição do dispositivo, o padrão de tráfego e à interferência. Cada dispositivo pode reconfigurar dinamicamente sua topologia, cobertura e alocação do canal de acordo com as mudanças [Cremonezi et al. 2017]. Além disso, em geral, nenhuma entidade centralizada controla a rede, exigindo uma abordagem de gerenciamento descentralizada [Lima et al. 2009, Nogueira 2009].

Como já enfatizado ao longo deste capítulo, a segurança é crucial para os sistemas UbiComp, particularmente quando eles oferecem suporte a aplicativos sensíveis à segurança como em contextos militares, de segurança pública, finanças e de saúde. As ameaças de segurança aproveitam falhas de protocolos e vulnerabilidades dos sistemas operacionais, bem como as características da rede. Essas redes representam desafios não triviais para o projeto de segurança devido às suas características, como o meio sem fio compartilhado, a topologia de rede altamente dinâmica, a comunicação multi-salto e a baixa proteção física dos dispositivos portáteis [Salem and Hubaux 2006, Yang et al. 2004]. Além disso, a ausência de entidades centrais aumenta a complexidade das operações de gerenciamento de segurança, particularmente o controle de acesso, a autenticação dos dispositivos e a distribuição de chaves criptográficas, tais como discutido na Seção 1.4.

Várias soluções de segurança vêm sendo propostas [Hu et al. 2003, Lou et al. 2004, Marti et al. 2000, Papadimitratos and Haas 2003, Refaei et al. 2005, Yi et al. 2001, Zapata 2002]. Essas soluções empregaram duas linhas de defesa bem definidas como preventiva ou reativa, nas quais a primeira tenta prevenir ataques por criptografia, autenticação e mecanismos de controle de acesso, e a última procura detectar intrusões e reagir de acordo [Lima et al. 2009, Nogueira 2009]. No entanto, naturalmente cada mecanismo de segurança aborda problemas específicos com limitações para lidar com diferentes tipos de ataques e intrusões. As defesas preventivas, por exemplo, são

vulneráveis a dispositivos infectados que participam de operações da rede e do sistema, enquanto que as reativas em sua maioria funcionam eficientemente somente contra ataques ou intrusões conhecidos.

Devido às limitações das linhas de defesa preventiva e reativa, os pesquisadores propuseram complementá-las através da abordagem de tolerância a intrusões [Yang et al. 2004], tal como ilustrado na Figura 1.5. Esta linha de defesa tem como objetivo melhorar a resiliência da rede e do ciberespaço contra ataques e intrusões usando técnicas de tolerância a falhas, geralmente redundância e mecanismos de recuperação.

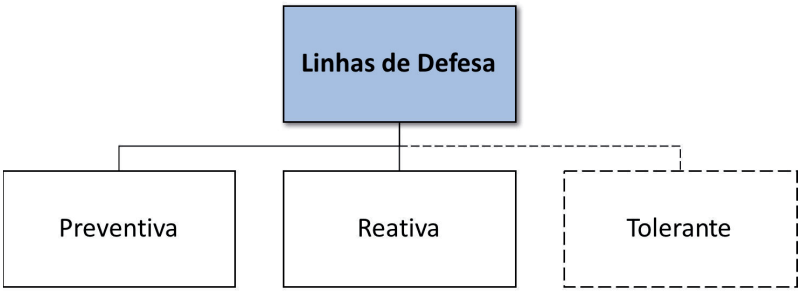


Figura 1.5. Linhas de Defesa: Composição da Resiliência [Nogueira 2009]

As características da rede e do ciberespaço, e as limitações nas linhas de defesa, reforçam o fato de que nenhum sistema é totalmente imune a ataques e intrusões. Portanto, novas abordagens são necessárias para garantir a integridade, confidencialidade, autenticação e, especialmente, a disponibilidade de serviços. Esses aspectos motivam o projeto de serviços resilientes. A resiliência é a capacidade de um sistema oferecer serviços essenciais, mesmo em face de ataques e intrusões [Laprie et al. 2004, Lima et al. 2009, Nogueira 2009]. Neste minicurso focamos na comunicação (a entrega de dados de um dispositivo na UbiComp para outro) em seus diferentes níveis como serviço essencial para o contexto da UbiComp. Desta forma, destacamos três serviços: conectividade física e de camada de enlace, roteamento e comunicação lógica de fim-a-fim. Seguimos a afirmação de que a resiliência é alcançada quando uma solução integra e gerencia as linhas de defesa preventivas, reativas e tolerantes de uma maneira auto-adaptativa e coordenada [Lima et al. 2009, Nogueira 2009]. A seguir listamos os requisitos para atingir resiliência no contexto UbiComp diante do nosso ponto de vista.

1.5.1. Requisitos de Resiliência no Contexto UbiComp

Os requisitos de resiliência podem variar substancialmente de acordo com o escopo do sistema, a criticidade dos serviços oferecidos e as consequências da interrupção do serviço [Ellison et al. 1997, Linger et al. 1998]. As redes no contexto da UbiComp apresentam diversas funções, operações e serviços influenciados direcionados pelos requisitos das aplicações. Em uma situação crítica em que partes do sistema são comprometidas por ataques ou intrusões, a prioridade é dada para manter a conectividade de rede em três níveis principais: camada de física, roteamento e comunicação fim-a-fim. Portanto, para alcançar a resiliência, alguns requisitos devem ser atingidos na engenharia de soluções resilientes. Os requisitos apontam para funções ou recursos necessários para melhorar a capacidade da rede e do sistema de fornecer ou recuperar os serviços essenciais mesmo

diante de ataques ou intrusões.

Neste capítulo, categorizamos os requisitos de resiliência em dois grupos: os requisitos relacionados aos *serviços essenciais* e aqueles requisitos relacionados às *características da rede/sistema*. Os requisitos no primeiro grupo são resumidos como:

- **heterogeneidade** - uma abordagem de resiliência deve considerar a heterogeneidade dos dispositivos, da tecnologia de comunicação e da capacidade de recursos dos dispositivos.
- **autoconfiguração** - a rede e o sistema devem ser capazes de alterar dinamicamente os valores dos parâmetros das conexões, dos dispositivos e dos protocolos, bem como dos mecanismos de segurança, como regras de *firewall* e os limites dos sistemas de reputação, por exemplo.
- **autoadaptação** - capacidade da rede ou sistema de se ajustar em resposta às condições, como mobilidade, e aos requisitos das atividades, como níveis exigidos de Qualidade de Experiência (QoE) pelos usuários.
- **eficiência** - a abordagem de resiliência deve suportar o uso eficiente de recursos dos dispositivos e da rede, como uso energético do dispositivo e largura de banda da rede quando uma falha maliciosa é suspeita ou acontecendo.
- **controle de acesso** - os mecanismos devem controlar o acesso dos dispositivos na rede, bem como monitorar suas atividades.
- **proteção** - os mecanismos de segurança precisam ser gerenciados e combinados para proteger a comunicação em todas as camadas da pilha de protocolos.
- **integridade**, **confidencialidade** e **não-repúdio** – princípios de segurança que precisam ser assegurados para a comunicação.
- **redundância** - a rede deve tolerar e mitigar os ataques por meio de técnicas de tolerância a intrusões, como protocolos duplos ou operações de criptografia, uso simultâneo de rotas múltiplas e outros.
- **robustez** - a rede e o sistema devem continuar seus serviços mesmo durante uma eventual desconexão.

O segundo grupo inclui requisitos de resiliência associados às características gerais das redes ou sistema que compõem o contexto da UbiComp, tais como:

- **descentralização** - a resiliência na UbiComp deve ser fornecida por uma abordagem descentralizada para evitar um ponto de falha, sensível a ataques.
- **auto-organização** - os mecanismos para apoiar a resiliência devem ser auto-organizados sem exigir intervenção humana diante das necessárias mudanças para se adaptar às condições da rede e do sistema.

- **escalabilidade** - os mecanismos para fornecer resiliência devem considerar a variabilidade e o crescimento no número total de dispositivos.
- **autodiagnóstico** - a rede e o sistema devem monitorar-se e detectar falhas, indisponibilidades, mau-comportamento ou dispositivos maliciosos.
- **autorrecuperação** - as abordagens resilientes devem evitar disrupções de conectividade e recuperar a rede de problemas que possam ter acontecido. Eles também devem encontrar uma maneira alternativa de usar recursos e reconfigurar dispositivos, redes ou protocolos para mantê-los em operação normal.
- **auto-otimização** - os mecanismos devem otimizar o uso de recursos da rede e do sistema, minimizando a latência e mantendo a qualidade do serviço.

1.5.2. Questões em Aberto

Quais são os desafios abertos para alcançar a resiliência no contexto UbiComp? Em primeiro lugar, enfatizamos a heterogeneidade dos dispositivos e tecnologias que compõem os ambientes UbiComp. A integração de sistemas de grande escala, como os centros de dados Cloud, para dispositivos minúsculos, como sensores portáteis e implantáveis, é um enorme desafio, devido à complexidade resultante. Então, além disso, a integração das três linhas de defesa e sua adaptação é ainda mais difícil diante dos diferentes requisitos desses dispositivos, suas capacidades em termos de memória e processamento, e os requisitos das aplicações. Ainda, lidar com a heterogeneidade em termos de tecnologia e protocolos de comunicação torna mais difícil analisar o comportamento e a topologia da rede, o que poderia ser usado para auxiliar no projeto de soluções resilientes.

Outro desafio é como lidar com a escala. Primeiro, os sistemas UbiComp tendem a ser de grande escala e geograficamente distribuídos. Como lidar com a complexidade resultante disso? Como definir e construir modelos para entender esses sistemas e oferecer serviços resilientes? Finalmente, destacamos como desafios a incerteza e a velocidade. Se, por um lado, é tão difícil modelar, analisar e definir serviços resilientes neste sistema complexo, por outro lado incerteza é uma norma sobre eles, sendo a velocidade e o baixo tempo de resposta um forte requisito para as aplicações nesses sistemas. Por isso, como abordar esses elementos juntos? Como gerenciá-los para oferecer serviços resilientes considerando diversos tipos de requisitos das várias aplicações?

Todas essas questões levam a profundas investigações e desafios. No entanto, eles também mostram oportunidades para pesquisa aplicada na concepção e engenharia de sistemas resilientes, principalmente para o contexto UbiComp. Particularmente, se nos concentramos na concepção de sistemas resilientes que possam coordenar de forma adaptativa as três linhas de defesa podem ser um grande avanço para a pesquisa aplicada e para a resiliência dos sistemas e das redes.

1.6. Gestão de Identidade

Uma forma de atender aos requisitos de segurança de sistema UbiComp é por meio de uma infraestrutura de autenticação e de autorização (*Authentication and Authorization Infrastructure* – AAI). AAI é conhecida como o elemento central para prover a segurança

em aplicações distribuídas. Com esta infraestrutura, é possível implantar a gestão de identidade – GId (do inglês, *Identity Management* - IdM) de forma a impedir que usuários ou dispositivos (ilegítimos ou não) tenham acesso aos recursos que estes não estão autorizados [Lopez et al. 2004]. A gestão de identidade pode ser entendida como o conjunto de processos e tecnologias usados para garantir a identidade de uma entidade (usuário, dispositivo ou sistema), garantir as informações de uma identidade (identificadores, credenciais e atributos) e para prover procedimentos de autenticação, autorização e auditoria [ITU 2009]. Uma abordagem apropriada de gestão de identidade é necessária para que UbiComp seja invisível para usuários [Arias-Cabarcos et al. 2015].

De acordo com [ITU 2009], uma identidade eletrônica pode ser definida como um conjunto de dados sobre uma entidade que é suficiente para identificar esta entidade em um contexto digital particular. Uma identidade é constituída de:

- Identificador – conjunto de dígitos, caracteres e símbolos ou qualquer outra forma de dados usados para identificar unicamente a identidade de uma entidade (p.ex., UserID, e-mail, URI e endereço IP). IoT requer um identificador único e global para cada entidade da rede;
- Credenciais – um objeto que pode ser usado para provar uma identidade. Exemplo de credenciais incluem certificados digitais X.509 assinados por uma autoridade certificadora, senhas, *tokens* entre outras;
- Atributos – conjunto de dados descritivos ligado a uma entidade que especifica suas características. Por exemplo, nome completo, endereço domiciliar e data de nascimento.

1.6.1. Sistema de Gestão de Identidade

Um sistema de GId lida com todo o ciclo de vida da identidade, que consiste de seu registro, armazenamento, recuperação, provisionamento e revogação de atributos da identidade [Bhargav-Spantzel et al. 2007]. Vale destacar que gerenciar o ciclo de vida da identidade de dispositivos é mais complexo e custoso do que gerenciar a identidade de pessoas [Garcia-Morchon et al. 2017]. Um sistema de GId é caraterizado pelos seguintes elementos [Bhargav-Spantzel et al. 2007]:

- Entidade: usuário, dispositivo ou serviço que deseja acessar um recurso;
- Provedor de identidade (*Identity Provider* – IdP) - responsável por gerenciar identidades de entidades e pelo processo de autenticação;
- Provedor de Serviço (*Service Provider* – SP) - oferece recursos as entidades autorizadas.

A disposição destes elementos em um sistema de GId e a forma com que estes interagem entre si caraterizam os modelos de gestão de identidades, sendo estes classificados como [Bhargav-Spantzel et al. 2007]: tradicional (isolado ou silo), centralizado, federado e centrado no usuário.

Os sistemas UbiComp são compostos de dispositivos heterogêneos que precisam provar a sua autenticidade para as entidades com as quais se comunicam. Um problema neste cenário é garantir a autenticidade de dispositivos e de usuários que se comunicam entre si e que podem estar localizados em domínios administrativos de segurança diferentes, bem como podem utilizar mecanismos de autenticação e autorização distintos [Wangham et al. 2013].

Uma das maneiras de prover a GId em um cenário com múltiplos domínios de segurança é por meio de uma AAI baseada no modelo de gestão de identidades federadas. Em uma federação relações de confiança são estabelecidas entre IdPs e SPs para possibilitar a troca de informações relacionadas a identidade e o compartilhamento de serviços. Conforme ilustrado na Figura 1.6, neste modelo, os acordos de confiança existentes garantem que usuários/dispositivos autenticados em seus IdPs de origem podem acessar recursos protegidos providos por SPs de outros domínios da federação. A autenticação única (*Single Sign-On – SSO*) é obtida quando o mesmo evento de autenticação pode ser usado para acessar diferentes serviços federados [Bhargav-Spantzel et al. 2007].

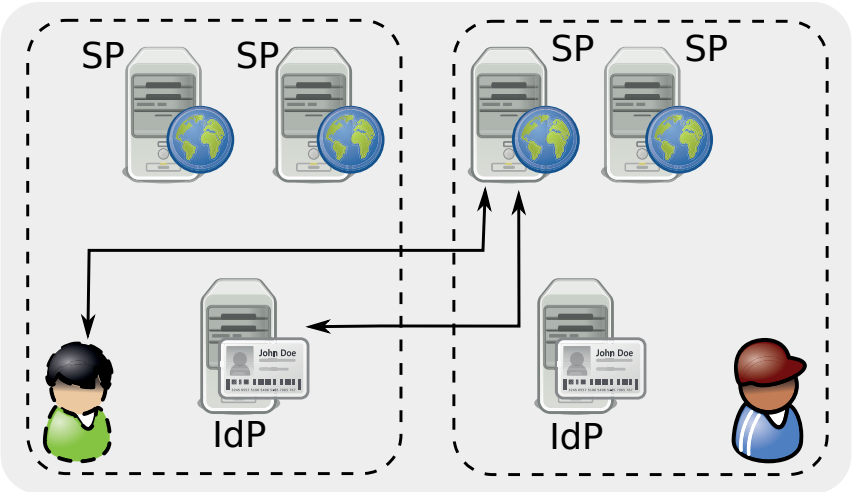


Figura 1.6. Modelo de Identidades Federadas [Wangham et al. 2010]

No modelo federado, a tradução de credenciais de autenticação para sistemas de controle de acesso físico (do inglês, *Physical Access Control Systems* - PACS), ou seja, acesso físico e acesso lógico federado unificado, também é um desafio.

1.6.1.1. Autenticação

A autenticação de dispositivos e de usuários em uma mesma infraestrutura é considerada em [Nguyen et al. 2010, Hanumanthappa and Singh 2012] usando o modelo de GId centralizado e em [Gusmeroli et al. 2013] usando o modelo de GId tradicional. Em [Akram and Hoffmann 2008, Liu et al. 2012, Ndibanje et al. 2014], as IAAs propostas para IoT, adequadas aos sistemas UbiComp, seguem o modelo federado, entretanto, tratam apenas da autenticação SSO de usuários, não da autenticação de dispositivos. Assim, a gestão de identidades federadas de dispositivos ainda é um desafio de pesquisa.

[Kim et al. 2015] propuseram uma solução centralizada que usa diferentes mecanismos de autenticação de dispositivos que são selecionados tendo como base a autonomia de energia do dispositivo. Entretanto, esta solução não provê autenticação de usuários. Baseado no modelo tradicional, uma AAI composta por um conjunto de protocolos que provê autenticação e controle de acesso durante todo o ciclo de vida de um dispositivo da IoT é proposto em [Neto et al. 2016]. [Domenech et al. 2016] propõem uma AAI para Web das Coisas, baseada no modelo de identidades federadas, que provê na mesma infraestrutura a autenticação única de usuários e de dispositivos, por meio de diferentes mecanismos de autenticação. Nesta AAI, um IdP pode ser implementado como um serviço na nuvem (IdPaaS) ou *on premise*

Os mecanismos e protocolos de autenticação consomem recursos computacionais, portanto, integrar uma AAI em um dispositivo com recursos limitados pode ser um desafio. Conforme mencionado na seção 1.4.3, um conjunto de criptografias leves, que não impõe sobrecargas relacionadas ao uso de certificados em dispositivos, pode ser usado para fornecer autenticação de dispositivo em sistemas UbiComp. Existe uma tendência recente que investiga os benefícios do uso da criptografia baseada em identidade (*Identity-Based Cryptography* –IBC) para fornecer autenticação entre domínios para dispositivos restritos [Markmann et al. 2015, Neto et al. 2016, Domenech et al. 2016].

A autenticação multi-fator (*Multi-Factor Authentication* – MFA) é uma solução criada para melhorar a robustez do processo de autenticação e, geralmente, combina dois ou mais fatores de autenticação ("algo que você sabe", "algo que você possui"e "algo que você é") [NIST 2017]. Em [Brainard et al. 2006], é apresentado outro fator: “alguém que você conhece” – relações humanas para intermediar a autenticação de um usuário. Neste tipo de autenticação, um invasor precisa comprometer dois ou mais fatores, o que torna a tarefa mais complexa. Muitos IdPs e SPs já oferecem multi-fator para autenticar seus usuários, porém, para para autenticar dispositivos ainda não há soluções amplamente aceitas.

1.6.1.2. Autorização

Em sistemas UbiComp, consideramos que um domínio de segurança pode ter dispositivos cliente e dispositivos SP, que são dispositivos com um provedor de serviço incorporado. Neste contexto, dispositivos físicos e SPs on-line podem oferecer serviços. Neste sistemas dinâmicos, os dispositivos entram e saem da rede, os SPs aparecem e desaparecem, logo o controle de acesso deve se adaptar para manter a percepção do usuário de estar sendo automaticamente e continuamente autenticado [Arias-Cabarcos et al. 2015]. O controle de acesso fornecido pela AAI embarcada no dispositivo também é um requisito significativo. Como esses dispositivos funcionam no mundo real, uma ameaça de segurança contra esses dispositivos pode afetar o mundo físico. Assim, se um dispositivo for acessado de forma incorreta, há uma chance de que essa violação afete o mundo físico arriscando o bem-estar das pessoas e até mesmo a vida [Domenech et al. 2016].

No contexto da IoT, os mecanismos de autorização são baseados em modelos de controle de acesso usados na Internet clássica, como o discricionário (Lista de Controle de Acesso - ACL [Guinard et al. 2010]), Controle de Acesso Baseado em Capacidades (*Ca-*

pability)- CapBAC [Rotondi et al. 2011, Mahalle et al. 2013], Controle de Acesso Baseado em Papel - RBAC [De Souza et al. 2008, Liu et al. 2012, Jindou et al. 2012] e Controle de Acesso Baseado em Atributos - ABAC [Han and Li 2012, Zhang and Liu 2011, Neto et al. 2016]. ABAC e RBAC são os modelos mais alinhados aos sistemas de identidades federadas [Wangham et al. 2013]. Conforme proposto em [Domenech et al. 2016], um sistema de GId que suporte diferentes modelos de controle de acesso, como RBAC e ABAC, pode se adaptar mais facilmente às necessidades dos processos de administração no contexto da UbiComp.

Em relação ao modelo de gestão de políticas de acesso aos dispositivos, as soluções de autorização para sistemas UbiComp podem implementar duas abordagens: *provisioning* [Gardel et al. 2013, Domenech et al. 2016] ou *outsourcing* [Alam et al. 2011, Seitz et al. 2013, Fremantle et al. 2014, Domenech et al. 2016]. Na abordagem *provisioning*, a tomada de decisão de autorização, ocorre no próprio dispositivo, o que exige que a política esteja em um repositório local. Nesta abordagem, o *Policy Enforcement Point* (PEP), que controla os acessos ao dispositivo, e o *Policy Decison Point* estão juntos em cada dispositivo do domínio. Na abordagem *outsourcing*, a tomada de decisão de acesso ocorre fora do ambiente computacional do dispositivo (SP), em um serviço externo centralizado, que responde a pedidos de avaliação de políticas de todos os guardiões de recursos (PEPs) dos dispositivos (SPs) de um domínio. Neste caso, a tomada de decisão pode ser oferecida como um serviço (*PDPaaS - Policy Decison Point as a Service*) na nuvem ou *on premise* conforme proposto em [Domenech et al. 2016].

Para dispositivos com restrições computacionais, a abordagem de *provisioning*, apesar de ser mais robusta, por não depender de um serviço externo, pode ser muito custosa tanto para processar a tomada de decisão quanto para administração das políticas de autorização. Já a abordagem de *outsourcing*, que simplifica a gestão das políticas de autorização em um domínio, tem como desvantagem o sobrecusto de comunicação e a criação de um ponto de único de falhas (PDP centralizado).

1.6.2. Protocolos de Gestão de Identidade Federada

Os modelos de GId guiam a construção de políticas e processos de negócios para sistemas GId, mas não indicam quais protocolos ou tecnologias devem ser adotados. SAML 2.0 (Security Assertion Markup Language) [Committee et al. 2012], OpenId Connect [Foundation 2014] e OAuth 2 [Hardt 2012a] se destacam no contexto de identidades federadas [Maler and Reed 2008].

SAML, desenvolvido pela OASIS, é uma estrutura baseada em XML para descrever e trocar informações de segurança entre parceiros de negócios. Esta define a sintaxe e as regras para solicitar, criar, comunicar e usar asserções SAML, que possibilita a autenticação SSO por diversos domínios administrativos. Além disso, SAML pode descrever eventos de autenticação que usam diferentes mecanismos de autenticação [OASIS 2005]. Essas características são muito importantes para que a interoperabilidade entre domínios administrativos diferentes seja alcançado. Segundo [Paci et al. 2009], o primeiro passo para alcançar a interoperabilidade é a adoção do SAML.

Um dos perfis de utilização do SAML é o ECP (*Enhanced Client and Proxy*). O ECP define a troca de informações de segurança envolvendo clientes ativos que não usam

um navegador web e permite a autenticação única de dispositivos. Contudo, o ECP exige o uso do protocolo SOAP nas trocas de mensagens, o que muitas pode não ser adequado devido ao seu alto custo computacional [Zeng et al. 2011].

O OpenID Connect é um *framewok* aberto que adota o modelo GId federado e centrado no usuário. Este é descentralizado, o que significa que nenhuma autoridade central aprova ou registra SPs. Com o OpenID, um usuário pode escolher o Provedor OpenID (IdP) que ele quer usar. O OpenID Connect é uma camada de identidade simples em cima do protocolo OAuth 2.0, que permite que os Clientes (SPs) verifiquem a identidade do usuário ou do dispositivo com base na autenticação realizada em um Servidor de Autorização OAuth (Provedor OpenID), bem como obtenha informações básicas do perfil do usuário ou dispositivo, de uma maneira interoperável e REST [Foundation 2014].

O protocolo OAuth [Hardt 2012b] é um *framework* de autenticação e autorização aberto que permite que um usuário/aplicação compartilhe recursos na web (delegue acesso a um recurso) com terceiros sem ter que compartilhar sua credencial de autenticação. Com o protocolo OAuth 2.0 é possível autorizar o acesso a esses recursos por um tempo determinado. Em [Fremantle et al. 2014], os autores exploram o uso de OAuth para sistemas IoT que usam o protocolo MQTT, um protocolo de fila de mensagem leve (*publish/subscribe*) para pequenos sensores e dispositivos móveis.

Um padrão conhecido para prover autorização em sistemas distribuídos é o XACML (*eXtensible Access Control Markup Language*). O XACML é uma linguagem baseada em XML para descrição de políticas de autorização e para requisição/resposta de decisões de controle de acesso. Decisões de autorização podem ser baseadas em atributos do usuário/dispositivo, das ações solicitadas e das características do ambiente. Estas funcionalidades possibilitam a criação de mecanismos de autorização flexíveis. Além disso, o XACML é genérico, independentemente do modelo de controle de acesso usado (RBAC, ABAC) e permite o uso do modelo de tomada de decisão local (*provisioning*) ou de um provedor de serviço externo (modelo *outsourcing*). Outro aspecto importante é que existem perfis e extensões que fornecem interoperabilidade entre XACML e SAML [OASIS 2013].

1.6.3. Outros Desafios da Gestão de Identidade Pervasiva

As tecnologias de federação atuais dependem de acordos estáticos pré-configurados entre suas entidades (IdPs, SPs), que não são adequados para ambientes abertos como os ambientes de computação ubíqua. Ambientes de UbiComp são dinâmicos, multi-domínios e com múltiplos SPs. Estas pre-configurações afetam negativamente a escalabilidade e a flexibilidade das soluções. O estabelecimento dinâmico de confiança é a chave para a escalabilidade. Embora os protocolos de identidade federada citados possam cobrir aspectos de segurança, os desafios de usabilidade e confiança (*trust*) são questões em aberto [Arias-Cabarcos et al. 2015].

A interoperabilidade é outro requisito fundamental para gestão de identidade pervasiva. Os sistemas UbiComp podem ser formados por domínios heterogêneos (organizações) que vão além das barreiras de uma Federação (com o mesmo protocolo de GId). A interoperabilidade entre federações baseadas no protocolo SAML foi tratada em alguns trabalhos [Silva et al. 2015, Souza and Wangham 2015], porém, alguns requisitos

para computação ubíqua (usabilidade, estabelecimento dinâmico de relações de confiança, autenticação de dispositivos e uso de criptografia leve) não foram considerados nestes trabalhos. A interoperabilidade entre diferentes protocolos de identidades federadas (SAML, OpenId Connect e OAuth) ainda é um problema em aberto e uma oportunidade de pesquisa.

Por fim, sistemas de GId federadas para sistemas Ubicomp devem proteger adequadamente as informações do usuário e devem aderir adequadamente às política de privacidade definidas para os dados do mesmo. A seção a seguir trata especificamente dos desafios para prover privacidade em sistemas UbiComp.

1.7. Implicações de privacidade

Sistemas de UbiComp tendem a coletar uma grande quantidade de dados e gerar muita informação. Usada de forma correta, a informação gera inúmeros benefícios para nossa sociedade que vem nos propiciando uma vida melhor ao longo dos anos. No entanto, a informação pode ser usada para fins ilícitos, assim como sistemas computacionais são usados para ataques. Proteger informações privadas é um grande desafio que muitas vezes pode parecer impraticável, por exemplo, proteger da companhia de energia elétrica local os dados de consumo elétrico gerados pelos respectivos clientes [Borges et al. 2014, Borges de Oliveira 2017a, Borges de Oliveira 2017d]

O texto desta seção apresenta uma visão ampla das questões de privacidade. Para uma introdução mais detalhada, recomendamos a leitura do trabalho *Introdução à Privacidade:Uma Abordagem Computacional* [Borges 2016a] apresentado nesta mesma série.

1.7.1. Desafios do cenário da aplicação

Quando remetentes e receptores são considerados no contexto de um cenário de aplicação particular, há dois grandes desafios. O primeiro é a identificação de dados sensíveis quem podem diferir de cultura para cultura. O segundo é como lidar com várias leis e regulamentações potencialmente conflitantes.

1.7.1.1. Identificando dados sensíveis

Decidir quais dados podem ser sensíveis pode ser uma tarefa desafiadora. O artigo 12 da Declaração Universal dos Direitos Humanos proclamada pela Assembleia Geral das Nações Unidas em Paris, em 10 de dezembro de 1948, afirma: *"Ninguém deve ser submetido a uma interferência arbitrária em sua privacidade, família, lar ou correspondência, nem ataques a sua honra e reputação. Todos têm o direito à proteção da lei contra tais interferências ou ataques."* Em geral, é necessária mais atenção aos dados que permitem o monitoramento de produtos, animais e pessoas. Esses dados podem ser usados para criar um perfil comportamental que permite predição e manipulação. Os protocolos práticos de preservação da privacidade permitem ao receptor obter informações consolidadas ou coletar dados agregados por adição ou concatenação, anonimamente. Por exemplo, os dados de localização apresentam vários problemas de privacidade e, portanto, poucas pessoas compartilham sua localização [Gu et al. 2016], embora os smartphones possam divulgar dados mais sensíveis do que a localização [Ge et al. 2016]. Os sistemas

de UbiComp na área de saúde processam dados sensíveis [Liu et al. 2014], e as câmeras de segurança podem transmitir dados confidenciais [Tekeoglu and Tosun 2015]. Os desafios de privacidade não têm a ver apenas com os cidadãos, mas também as indústrias [Sadeghi et al. 2015]. Mesmo que uma mensagem seja criptografada, os metadados - ou seja, o tráfego de rede - podem revelar informações confidenciais [Hu et al. 2016].

1.7.1.2. Regulamentação

Os regulamentos sobre privacidade estão a crescer em todo o mundo e as leis variam de país para país. Isso é um desafio para as instituições internacionais que desenvolvem sistemas de UbiComp comercializá-los em todo o mundo de acordo com a lei. Outro problema pode ser a ausência de lei, permitindo que os concorrentes desenvolvam estratégias baseadas em informações privadas que aumentam a vantagem no mercado em relação a corporações mais éticas. Em geral, violações da privacidade podem mudar uma eleição e podem levar a uma sociedade de vigilância [Holvast 2009]. Independentemente dos cenários de aplicação, os protocolos de preservação da privacidade enfrentam seus próprios desafios tecnológicos.

1.7.2. Desafios tecnológicos

Em casos de termos uma base de dados pronta para ser anonimizada, ou seja, os dados sensíveis já foram identificados, então podemos usar alguns softwares para localizar os dados e anonimizar a base [Prasser and Kohlmayer 2015, Dai et al. 2009, Poulis et al. 2015]. As técnicas de k-anonimato, l-diversidade, e t-proximidade [Li et al. 2007] estão nas bases de muitas ferramentas.

Os desafios são maiores quando os dados sensíveis são continuamente coletados e transmitidos em uma rede, em vez de ficarem estáticos em um banco de dados. Em séries temporais, poderíamos medir a privacidade de acordo com as possibilidades de encontramos os números em parte da série [Borges de Oliveira 2017c].

1.7.2.1. Computação de todos os operadores

Em teoria, qualquer operador pode ser computado sobre dados criptografados [Gentry 2009], tais técnicas são conhecidas como criptografia totalmente homomórfica. Na prática, é um desafio construir um sistema de criptografia totalmente homomórfica para muitos cenários de aplicativos. Os pesquisadores geralmente desenvolvem protocolos de preservação da privacidade baseados em criptografia homomórfica aditiva [Borges de Oliveira 2017f] e DC-Nets (de "Dining Cryptographers") [Borges de Oliveira 2017e]. Ambas as técnicas computam o operador de adição sobre dados criptografados. No entanto, as primeiras são funções, e estas são famílias de funções. Dada uma criptografia homomórfica aditiva, podemos construir uma DC-Net assimétrica [Borges de Oliveira 2017e].

1.7.2.2. Troca entre aplicação e maleabilidade

DC-Nets pode impor privacidade, garantindo que ninguém pode decifrar mensagens, a menos que todas as mensagens cifradas tenham sido levadas em consideração. A criptografia homomórfica não impõe a privacidade, ou seja, as mensagens individuais podem ser decifradas. Isso acontece porque as DC-Nets não são maleáveis e a criptografia homomórfica é maleável. Por um lado, os esquemas de DC-Net permitem que os invasores alterem mensagens criptografadas e usem outras mensagens criptografadas para deduzir o valor associado a outras pessoas no processo de agregação. Por outro lado, é mais fácil gerar e distribuir chaves para sistemas de criptografia homomórfica do que para esquemas DC-Net.

1.7.2.3. Distribuição de chave

Usando a criptografia homomórfica, o receptor precisa apenas gerar um par de chaves público-privado e transmitir a chave pública para os remetentes de forma autenticada, o que compartilhará a mesma chave para criptografar suas mensagens. Embora as mensagens possam ter o mesmo conteúdo, as mensagens criptografadas serão diferentes umas das outras porque os esquemas de criptografia homomórfica são probabilísticos. Usando esquemas DC-Net, um protocolo não precisa de um receptor porque os remetentes podem ser os receptores. Por esse motivo, vamos apenas chamá-los de participantes, que geram suas chaves privadas. Para redes DC simétricas, cada participante compartilha duas chaves entre si, ou seja, envia um segredo para e recebe outro segredo entre si. Depois, cada participante tem uma chave privada dada pela lista de segredos. Por isso, cada participante criptografa uma mensagem $m_{i,j}$ usando a função

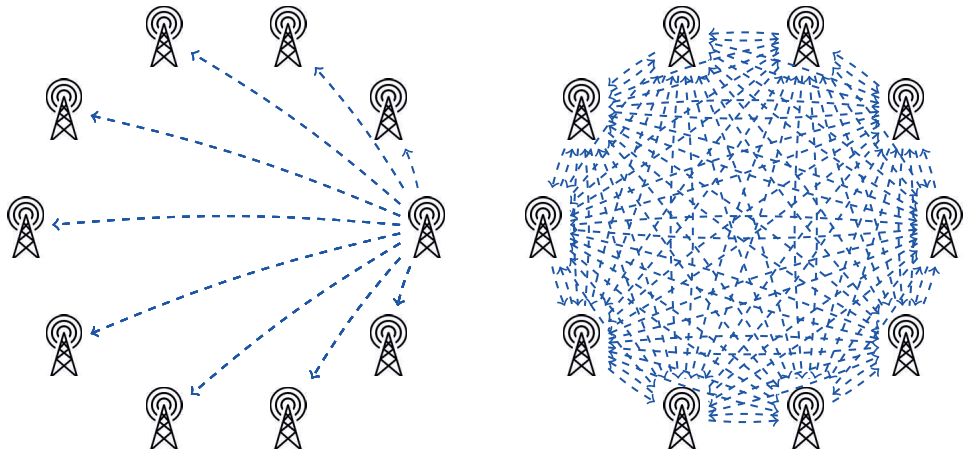
$$\mathfrak{M}_{i,j} \leftarrow \mathrm{Enc}(m_{i,j}) = m_{i,j} + \sum_{o \in \mathscr{M} - \{i\}} \mathrm{Hash}(s_{i,o} \,||\, j) - \mathrm{Hash}(s_{o,i} \,||\, j),$$

onde $m_{i,j}$ é a mensagem enviada pelo participante i no tempo j, Hash é uma função de hash segura predefinida pelos participantes, $s_{i,o}$ é o segredo enviado do participante i para o participante o, da mesma forma, $s_{o,i}$ é o segredo recebido por i de o e $||$ é o operador de concatenação. Cada participante i pode enviar a mensagem criptografada $\mathfrak{M}_{i,j}$ uns para os outros. Assim, eles podem decifrar as mensagens criptografadas agregadas que computam

$$\mathrm{Dec} = \sum_{i \in \mathscr{M}} \mathfrak{M}_{i,j} = \sum_{i \in \mathscr{M}} m_{i,j}.$$

Ninguém pode decifrar se faltar uma ou mais mensagens. Para DC-Nets assimétricas, cada participante gera uma chave privada usada para cifrar e, em seguida, eles adicionam as chaves usando uma criptografia homomórfica ou DC-net simétrica para gerar a chave de decifrar, que pode ser tornada pública.

Observe que os esquemas de criptografia homomórfica tendem a ter baixa sobrecarga para configurar chaves e para distribuí-las. O receptor das mensagens criptografadas agregadas apenas gera um par de chaves pública-privada e envia a chave pública para cada remetente. DC-Nets simétricas precisam de $O(I^2)$ mensagens para configurar as chaves, onde I é o número de participantes. Figura 1.7 representa as mensagens para configurar chaves usando (a) criptografia homomórfica e (b) DC-nets simétricas.



(a) Criptografia homomórfica (b) DC-Nets simétricas

Figura 1.7. Configurando as chaves

1.8. Conclusão

Nas palavras de Mark Weiser, a Ubiquitous Computing é "a ideia de integrar computadores perfeitamente no mundo em geral" [Weiser 1991]. Assim, longe de ser um fenômeno dessa época, o projeto e a prática dos sistemas de UbiComp foram discutidos um quarto de século atrás. Neste capítulo, revisamos a noção, que permeia os mais variados níveis de nossa sociedade, sob um ponto de vista de segurança e privacidade. Nos próximos anos, esses dois tópicos ocuparão muito do tempo de pesquisadores e engenheiros. Em nossa opinião, o uso deste tempo deve ser guiado por algumas observações.

Conforme discutido na Seção 1.2, o software voltado para o contexto da de UbiComp é muitas vezes produzido como a combinação de diferentes linguagens de programação, compartilhando um núcleo comum implementado em um tipo inseguro Linguagem como C, C++ ou montagem.

A Seção 1.3 apresenta uma discussão sobre segurança a longo prazo. As infraestruturas críticas como redes elétricas são projetadas para durar muitas décadas e não podem ser atualizadas com frequência. Não sabemos se problemas matemáticos usados em técnicas criptográficas modernas podem ser resolvidos em um tempo que comprometa a técnica, ou se serão resolvidos algum dia. A vida útil de tais sistemas, juntamente com a difícil atualização e reimplantação, os torna vulneráveis ao inexorável progresso da tecnologia, que traz novos jogadores, como a criptografia pós-quântica.

Na Seção 1.4, é apresentado que o acesso físico e os recursos restritos compliquem o projeto de algoritmos criptográficos eficientes e seguros, que são frequentemente acessíveis aos ataques de canais laterais.

A Seção 1.5 discute sobre resiliência cibernética, ou seja, a capacidade que um sistema de UbiComp tem de recuperar-se e reduzir danos de eventuais falhas dos diversos equipamentos tecnológicos e oriundas de ataques cibernéticos.

Conforme discutido na Seção 1.6, para que a computação ubíqua seja de fato invisível para usuários, uma abordagem também pervasiva de gestão de identidade federada deve ser provida. Um dos desafios da gestão de identidade (GId) em sistemas de UbiComp

é garantir a autenticidade de dispositivos e de usuários em cenários com múltiplos e heterogêneos domínios administrativos de segurança. Diante da necessidade de controlar o acesso a recursos e serviços nos dispositivos de computação ubíqua, as infraestruturas de autorização devem ser flexíveis para que diferentes modelos de controles de acesso e abordagens de tomadas de decisão possam ser implantados de acordo com os requisitos de cada aplicação. Logo, o desafio é projetar ou aprimorar sistemas GId considerando os cenários de aplicação. Requisitos, muitas vezes conflitantes, precisam ser avaliados em cada cenário, tais como: flexibilidade, usabilidade, escalabilidade, interoperabilidade, segurança forte, mas leve e estabelecimento dinâmico de relações de confiança.

Na Seção 1.7, vemos que a disponibilidade e o volume de dados frequentemente manipulados pelos sistemas de UbiComp dificulta inda mais a proteção da privacidade dos usuários. Sistemas de UbiComp que levam em conta os problemas de privacidade têm todos os desafios encontrados em segurança, porém, com mais os desafios provenientes da proteção à privacidade.

Dadas essas observações e a importância da computação ubíqua, é fácil concluir que o futuro tem desafios fascinantes à espera da atenção da academia e da indústria.

Referências

[ddo] DDoS attacks: For the hell of it or targeted – how do you see them off? `http://www.theregister.co.uk/2016/09/22/ddos_attack_defence/`. Accessed: 2017-02-14.

[Abowd and Mynatt 2000] Abowd, G. D. and Mynatt, E. D. (2000). Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(1):29–58.

[Akram and Hoffmann 2008] Akram, H. and Hoffmann, M. (2008). Supports for identity management in ambient environments-the hydra approach. In *Proceedings...*, pages 371–377. 3rd International Conference on Systems and Networks Communications, 2008. ICSNC'08.

[Al-Riyami and Paterson 2003] Al-Riyami, S. S. and Paterson, K. G. (2003). Certificateless public key cryptography. In *ASIACRYPT*, volume 2894 of *LNCS*, pages 452–473. Springer.

[Alam et al. 2011] Alam, S., Chowdhury, M. M., and Noll, J. (2011). Interoperability of security-enabled internet of things. *Wireless Personal Communications*, 61(3):567–586.

[Albrecht et al. 2014] Albrecht, M. R., Driessen, B., Kavun, E. B., Leander, G., Paar, C., and Yalçin, T. (2014). Block ciphers - focus on the linear layer (feat. PRIDE). In *CRYPTO (1)*, volume 8616 of *LNCS*, pages 57–76. Springer.

[Almeida et al. 2016] Almeida, J. B., Barbosa, M., Barthe, G., Dupressoir, F., and Emmi, M. (2016). Verifying constant-time implementations. In *USENIX Security Symposium*, pages 53–70. USENIX Association.

[Aranha et al. 2014] Aranha, D. F., Karam, M. M., Miranda, A., and Scarel, F. (2014). *Software vulnerabilities in the Brazilian voting machine*, pages 149–175. IGI Global.

[Arias-Cabarcos et al. 2015] Arias-Cabarcos, P., Almenárez, F., Trapero, R., Díaz-Sánchez, D., and Marín, A. (2015). Blended identity: Pervasive idm for continuous authentication. *IEEE Security Privacy*, 13(3):32–39.

[Ashton 2009] Ashton, K. (2009). That 'Internet of Things' Thing. *RFiD Journal*, 22:97–114.

[Atzori et al. 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805.

[Aumasson and Bernstein 2012] Aumasson, J. and Bernstein, D. J. (2012). Siphash: A fast short-input PRF. In *INDOCRYPT*, volume 7668 of *LNCS*, pages 489–508. Springer.

[Aumasson et al. 2013] Aumasson, J., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. (2013). BLAKE2: simpler, smaller, fast as MD5. In *ACNS*, volume 7954 of *LNCS*, pages 119–135. Springer.

[Balasch et al. 2014] Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., and Standaert, F. (2014). On the cost of lazy engineering for masked software implementations. In *CARDIS*, volume 8968 of *LNCS*, pages 64–81. Springer.

[Banik et al. 2015] Banik, S., Bogdanov, A., and Regazzoni, F. (2015). Exploring energy efficiency of lightweight block ciphers. In *SAC*, volume 9566 of *LNCS*, pages 178–194. Springer.

[Barbulescu et al. 2014] Barbulescu, R., Gaudry, P., Joux, A., and Thomé, E. (2014). A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. In *EUROCRYPT 2014*, pages 1–16. Springer.

[Barker et al. 2012] Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M. (2012). Recommendation for key management part 1: General (revision 3). *NIST special publication*, 800(57):1–147.

[Beierle et al. 2016] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., and Sim, S. M. (2016). The SKINNY family of block ciphers and its low-latency variant MANTIS. In *CRYPTO (2)*, volume 9815 of *LNCS*, pages 123–153. Springer.

[Bennett and Brassard 1984] Bennett, C. H. and Brassard, G. (1984). Quantum Cryptography: Public Key Distribution and Coin Tossing. In *Proceedings of IEEE ICCSSP'84*, pages 175–179, New York. IEEE Press.

[Bernstein 2006] Bernstein, D. J. (2006). Curve25519: New diffie-hellman speed records. In *Public Key Cryptography*, volume 3958 of *LNCS*, pages 207–228. Springer.

[Bernstein et al. 2012a] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., and Yang, B. (2012a). High-speed high-security signatures. *J. Cryptographic Engineering*, 2(2):77–89.

[Bernstein et al. 2015] Bernstein, D. J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., and Wilcox-O'Hearn, Z. (2015). *SPHINCS: Practical Stateless Hash-Based Signatures*, pages 368–397. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bernstein et al. 2012b] Bernstein, D. J., Lange, T., and Schwabe, P. (2012b). The security impact of a new cryptographic library. In *LATINCRYPT*, volume 7533 of *Lecture Notes in Computer Science*, pages 159–176. Springer.

[Bhargav-Spantzel et al. 2007] Bhargav-Spantzel, A., Camenisch, J., Gross, T., and Sommer, D. (2007). User centricity: a taxonomy and open issues. *Journal of Computer Security*, 15(5):493–527.

[Biham and Shamir 1997] Biham, E. and Shamir, A. (1997). Differential fault analysis of secret key cryptosystems. In *CRYPTO*, volume 1294 of *LNCS*, pages 513–525. Springer.

[Biryukov et al. 2016] Biryukov, A., Dinu, D., and Khovratovich, D. (2016). Argon2: New generation of memory-hard functions for password hashing and other applications. In *EuroS&P*, pages 292–302. IEEE.

[Bogdanov et al. 2007] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. (2007). PRESENT: an ultra-lightweight block cipher. In *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer.

[Boldyreva et al. 2012] Boldyreva, A., Goyal, V., and Kumar, V. (2012). Identity-based encryption with efficient revocation. *IACR Cryptology ePrint Archive*, 2012:52.

[Boneh and Franklin 2003] Boneh, D. and Franklin, M. K. (2003). Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615.

[Borges 2016a] Borges, F. (2016a). *Introdução à Privacidade: Uma Abordagem Computacional*, pages 1–43. SBC.

[Borges 2016b] Borges, F. (2016b). *Privacy-Preserving Data Aggregation in Smart Metering Systems*. Energy Engineering Series. Institution of Engineering & Technology.

[Borges et al. 2014] Borges, F., Demirel, D., Bock, L., Buchmann, J. A., and Mühlhäuser, M. (2014). A privacy-enhancing protocol that provides in-network data aggregation and verifiable smart meter billing. In *ISCC*, pages 1–6. IEEE.

[Borges de Oliveira 2017a] Borges de Oliveira, F. (2017a). *Background and Models*, pages 13–23. Springer International Publishing, Cham.

[Borges de Oliveira 2017b] Borges de Oliveira, F. (2017b). *Introduction*, pages 3–12. Springer International Publishing, Cham.

[Borges de Oliveira 2017c] Borges de Oliveira, F. (2017c). *Quantifying the Aggregation Size*, pages 49–60. Springer International Publishing, Cham.

[Borges de Oliveira 2017d] Borges de Oliveira, F. (2017d). *Reasons to Measure Frequently and Their Requirements*, pages 39–47. Springer International Publishing, Cham.

[Borges de Oliveira 2017e] Borges de Oliveira, F. (2017e). *Selected Privacy-Preserving Protocols*, pages 61–100. Springer International Publishing, Cham.

[Borges de Oliveira 2017f] Borges de Oliveira, F. (2017f). *A Selective Review*, pages 25–36. Springer International Publishing, Cham.

[Brainard et al. 2006] Brainard, J., Juels, A., Rivest, R. L., Szydlo, M., and Yung, M. (2006). Fourth-factor authentication: somebody you know. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 168–178. ACM.

[Bronevetsky 2009] Bronevetsky, G. (2009). Communication-sensitive static dataflow for parallel message passing applications. In *CGO*, pages 1–12, Washington, DC, USA. IEEE.

[BTS 2017] BTS, U. B. o. T. S. (2017). Average age of automobiles and trucks in operation in the united states. Accessed: 2017-09-14.

[Buchmann et al. 2011] Buchmann, J., Dahmen, E., and Hülsing, A. (2011). Xmss - a practical forward secure signature scheme based on minimal security assumptions. In Yang, B.-Y., editor, *PQCrypto*, pages 117–129. Springer.

[Cadar et al. 2008] Cadar, C., Dunbar, D., and Engler, D. (2008). KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, pages 209–224. USENIX.

[Carlini et al. 2015] Carlini, N., Barresi, A., Payer, M., Wagner, D., and Gross, T. R. (2015). Control-flow bending: On the effectiveness of control-flow integrity. In *SEC*, pages 161–176, Berkeley, CA, USA. USENIX.

[Committee et al. 2012] Committee, O. S. S. T. et al. (2012). Security assertion markup language (saml) 2.0.

[Conti 2006] Conti, J. P. (2006). The internet of things. *Communications Engineer*, 4(6):20–25.

[Coppa et al. 2012] Coppa, E., Demetrescu, C., and Finocchi, I. (2012). Input-sensitive profiling. In *PLDI*, pages 89–98, New York, NY, USA. ACM.

[Costello and Longa 2015] Costello, C. and Longa, P. (2015). Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the mersenne prime. In *ASIACRYPT (1)*, volume 9452 of *LNCS*, pages 214–235. Springer.

[Cousot and Cousot 1977] Cousot, P. and Cousot, R. (1977). Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, New York, NY, USA. ACM.

[Cousot et al. 2011] Cousot, P., Cousot, R., and Logozzo, F. (2011). A parametric segmentation functor for fully automatic and scalable array content analysis. In *POPL*, pages 105–118, New York, NY, USA. ACM.

[Cremonezi et al. 2017] Cremonezi, B. M., Vieira, A. B., Nacif, J. A. M., and Nogueira, M. (2017). A dynamic channel allocation protocol for medical environment under multiple base stations. In *IEEE Wireless Communications and Networking Conference, WCNC*, pages 1–6.

[Daemen and Rijmen 2002] Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer.

[Dagenais and Hendren 2008] Dagenais, B. and Hendren, L. (2008). Enabling static analysis for partial java programs. In *OOPSLA*, pages 313–328, New York, NY, USA. ACM.

[Dai et al. 2009] Dai, C., Ghinita, G., Bertino, E., Byun, J.-W., and Li, N. (2009). Tiamat: A tool for interactive analysis of microdata anonymization techniques. *Proc. VLDB Endow.*, 2(2):1618–1621.

[De Souza et al. 2008] De Souza, L. M. S., Spiess, P., Guinard, D., Köhler, M., Karnouskos, S., and Savio, D. (2008). Socrades: A web service based shop floor integration infrastructure. In *The internet of things*, pages 50–67. Springer.

[Delic 2016] Delic, K. A. (2016). On resilience of iot systems: The internet of things (ubiquity symposium). *Ubiquity*, 2016(February):1–7.

[Diffie and Hellman 2006] Diffie, W. and Hellman, M. (2006). New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654.

[Dinu et al. 2015] Dinu, D., Corre, Y. L., Khovratovich, D., Perrin, L., Großschädl, J., and Biryukov, A. (2015). Triathlon of Lightweight Block Ciphers for the Internet of Things. NIST Workshop on Lightweight Cryptography.

[Dinu et al. 2016] Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., and Biryukov, A. (2016). Design strategies for ARX with provable bounds: Sparx and LAX. In *ASIACRYPT (1)*, volume 10031 of *LNCS*, pages 484–513.

[Domenech et al. 2016] Domenech, M. C., Boukerche, A., and Wangham, M. S. (2016). An authentication and authorization infrastructure for the web of things. In *Proceedings of the 12th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, Q2SWinet '16, pages 39–46, New York, NY, USA. ACM.

[DoT 2013] DoT, U. D. T. (2013). IEEE 1609 - Family of Standards for Wireless Access in Vehicular Environments WAVE.

[Ellison et al. 1997] Ellison, R., Fisher, D., Linger, R., Lipson, H., Longstaff, T., and Mead, N. (1997). Survivable network systems: an emerging discipline – CMU/SEI-97-TR-013. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[Estrin et al. 1999] Estrin, D., Govindan, R., Heidemann, J. S., and Kumar, S. (1999). Next century challenges: Scalable coordination in sensor networks. In *MobiCom'99*, pages 263–270.

[Faz-Hernández et al. 2015] Faz-Hernández, A., Cabral, R., Aranha, D. F., and López, J. (2015). Implementação Eficiente e Segura de Algoritmos Criptográficos. In *Minicursos do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSEG)*, pages 93–140. Sociedade Brasileira de Computação.

[Fedrecheski et al. 2016] Fedrecheski, G., Costa, L. C. P., and Zuffo, M. K. (2016). Elixir programming language evaluation for iot. In *ISCE*, page Online, Washington, DC, USA. IEEE.

[Foundation 2014] Foundation, T. O. (2014). Openid connect core 1.0. `http://openid.net/specs/openid-connect-core-1\_0.html`.

[Fremantle et al. 2014] Fremantle, P., Aziz, B., Kopecký, J., and Scott, P. (2014). Federated identity and access management for the internet of things. In *2014 International Workshop on Secure Internet of Things*, pages 10–17.

[Furr and Foster 2008] Furr, M. and Foster, J. S. (2008). Checking type safety of foreign function calls. *ACM Trans. Program. Lang. Syst.*, 30(4):18:1–18:63.

[Garcia-Morchon et al. 2017] Garcia-Morchon, O., Kumar, S., and Sethi, M. (2017). State-of-the-Art and Challenges for the Internet of Things Security. Internet-Draft draft-irtf-t2trg-iot-seccons-04, Internet Engineering Task Force. Work in Progress.

[Gardel et al. 2013] Gardel, T., Andrade, N., Farias, F., and Prazeres, C. (2013). Autenticação e autorização para acesso a aplicações em um barramento de serviços para a web das coisas. In *Anais do*. 13 Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg), 2013, SBC.

[Ge et al. 2016] Ge, Y., Deng, B., Sun, Y., Tang, L., Sheng, D., Zhao, Y., Xie, G., and Salamatian, K. (2016). A comprehensive investigation of user privacy leakage to android applications. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6.

[Gentry 2009] Gentry, C. (2009). *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA.

[Godefroid 2014] Godefroid, P. (2014). Micro execution. In *ICSE*, pages 539–549, New York, NY, USA. ACM.

[Godefroid et al. 2005] Godefroid, P., Klarlund, N., and Sen, K. (2005). Dart: directed automated random testing. In *PLDI*, pages 213–223, New York, NY, USA. ACM.

[Graham et al. 1982] Graham, S. L., Kessler, P. B., and McKusick, M. K. (1982). gprof: a call graph execution profiler (with retrospective). In *Best of PLDI*, pages 49–57, New York, NY, USA. ACM.

[Grosso et al. 2014] Grosso, V., Leurent, G., Standaert, F., and Varici, K. (2014). Ls-designs: Bitslice encryption for efficient masked software implementations. In *FSE*, volume 8540 of *LNCS*, pages 18–37. Springer.

[Großschädl et al. 2009] Großschädl, J., Oswald, E., Page, D., and Tunstall, M. (2009). Side-channel analysis of cryptographic software via early-terminating multiplications. In *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 176–192. Springer.

[Grover 1996] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *Proceedings of ACM STOC 1996*, pages 212–219, New York, NY, USA. ACM.

[Gu et al. 2016] Gu, Y., Yao, Y., Liu, W., and Song, J. (2016). We know where you are: Home location identification in location-based social networks. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9.

[Guinard et al. 2010] Guinard, D., Fischer, M., and Trifa, V. (2010). Sharing using social networks in a composable web of things. In *Proceedings...*, pages 702–707. 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010.

[Gusmeroli et al. 2013] Gusmeroli, S., Piccione, S., and Rotondi, D. (2013). A capability-based security approach to manage access control in the internet of things. *Mathematical and Computer Modelling*, 58(5–6):1189 – 1205.

[Han and Li 2012] Han, Q. and Li, J. (2012). An authorization management approach in the internet of things. *Journal of Information & Computational Science*, 9(6):1705–1713.

[Hanumanthappa and Singh 2012] Hanumanthappa, P. and Singh, S. (2012). Privacy preserving and ownership authentication in ubiquitous computing devices using secure three way authentication. In *Proceedings*, pages 107–112. International Conference on Innovations in Information Technology (IIT).

[Hardt 2012a] Hardt, D. (2012a). The oauth 2.0 authorization framework. RFC 6749, RFC Editor. `http://www.rfc-editor.org/rfc/rfc6749.txt`.

[Hardt 2012b] Hardt, E. D. (2012b). The oauth 2.0 authorization framework.

[Holvast 2009] Holvast, J. (2009). History of privacy. In Matyáš, V., Fischer-Hübner, S., Cvrček, D., and Švenda, P., editors, *The Future of Identity in the Information Society*, volume 298 of *IFIP Advances in Information and Communication Technology*, pages 13–42. Springer Berlin Heidelberg.

[Hu et al. 2016] Hu, J., Lin, C., and Li, X. (2016). Relationship privacy leakage in network traffics. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9.

[Hu et al. 2003] Hu, Y., Johnson, D., and Perrig, A. (2003). SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks. *Journal Ad Hoc Networks*, 01:175–192.

[Ishai et al. 2003] Ishai, Y., Sahai, A., and Wagner, D. (2003). Private circuits: Securing hardware against probing attacks. In *CRYPTO*, volume 2729 of *LNCS*, pages 463–481. Springer.

[ITU 2009] ITU (2009). Ngn identity management framework. Recommendation Y.2720.

[Jacobsson et al. 2016] Jacobsson, A., Boldt, M., and Carlsson, B. (2016). A risk analysis of a smart home automation system. *Future Generation Computer Systems*, 56(Supplement C):719 – 733.

[Jindou et al. 2012] Jindou, J., Xiaofeng, Q., and Cheng, C. (2012). Access control method for web of things based on role and sns. In *Proceedings...*, pages 316–321. IEEE 12th International Conference on Computer and Information Technology (CIT), 2012.

[Kaufmann et al. 2016] Kaufmann, T., Pelletier, H., Vaudenay, S., and Villegas, K. (2016). When constant-time source yields variable-time binary: Exploiting curve25519-donna built with MSVC 2015. In *CANS*, volume 10052 of *Lecture Notes in Computer Science*, pages 573–582.

[Kim and Barbulescu 2016] Kim, T. and Barbulescu, R. (2016). Extended tower number field sieve: A new complexity for the medium prime case. In *CRYPTO (1)*, volume 9814 of *LNCS*, pages 543–571. Springer.

[Kim et al. 2014] Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J., Lee, D., Wilkerson, C., Lai, K., and Mutlu, O. (2014). Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ISCA*, pages 361–372. IEEE Computer Society.

[Kim et al. 2015] Kim, Y.-P., Yoo, S., and Yoo, C. (2015). Daot: Dynamic and energy-aware authentication for smart home appliances in internet of things. In *Consumer Electronics (ICCE), 2015 IEEE International Conference on*, pages 196–197.

[Knill 2010] Knill, E. (2010). Physics: quantum computing. *Nature*, 463(7280):441–443.

[Koblitz 1987] Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209.

[Koblitz 1988] Koblitz, N. (1988). A family of jacobians suitable for discrete log cryptosystems. In *CRYPTO*, volume 403 of *LNCS*, pages 94–99. Springer.

[Kocher 1996] Kocher, P. C. (1996). Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, volume 1109 of *LNCS*, pages 104–113. Springer.

[Kocher et al. 1999] Kocher, P. C., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *CRYPTO*, volume 1666 of *LNCS*, pages 388–397. Springer.

[Kölbl et al. 2016] Kölbl, S., Lauridsen, M. M., Mendel, F., and Rechberger, C. (2016). Haraka v2 - efficient short-input hashing for post-quantum applications. *IACR Trans. Symmetric Cryptol.*, 2016(2):1–29.

[Langley 2010] Langley, A. (2010). ImperialViolet: Checking that functions are constant time with Valgrind. `https://www.imperialviolet.org/2010/04/01/ctgrind.html`.

[Laprie et al. 2004] Laprie, J.-C., Randell, B., Avizienis, A., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transaction Dependable Security Computer*, 1(1):11–33.

[Lee 2006] Lee, E. A. (2006). Cyber-physical systems-are computing foundations adequate. In *NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, volume 2. Citeseer.

[Leroy 2009] Leroy, X. (2009). Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115.

[Li et al. 2007] Li, N., Li, T., and Venkatasubramanian, S. (2007). t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115.

[Lima et al. 2009] Lima, M. N., dos Santos, A. L., and Pujolle, G. (2009). A survey of survivability in mobile ad hoc networks. *IEEE Communications Surveys Tutorials*, 11(1):66–77.

[Linger et al. 1998] Linger, R. C., Mead, N. R., and Lipson, H. F. (1998). Requirements definition for survivable network systems. In *Proceedings of the 3rd International Conference on Requirements Engineering*, page 0014, Washington, DC, USA. IEEE Computer Society.

[Lipa et al. 2015] Lipa, N., Mannes, E., Santos, A., and Nogueira, M. (2015). Firefly-inspired and robust time synchronization for cognitive radio ad hoc networks. *Computer Communications*, 66:36–44.

[Liu et al. 2012] Liu, J., Xiao, Y., and Chen, C. P. (2012). Authentication and access control in the internet of things. In *Proceedings...*, pages 588–592. 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW), 2012.

[Liu et al. 2014] Liu, W., Park, E. K., and Zhu, S. S. (2014). e-health pst (privacy, security and trust) mobile networking infrastructure. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6.

[Liu and Wong 2016] Liu, Z. and Wong, D. S. (2016). Practical attribute-based encryption: Traitor tracing, revocation and large universe. *Comput. J.*, 59(7):983–1004.

[López et al. 2015] López, H. A., Marques, E. R. B., Martins, F., Ng, N., Santos, C., Vasconcelos, V. T., and Yoshida, N. (2015). Protocol-based verification of message-passing parallel programs. In *OOPSLA*, pages 280–298, New York, NY, USA. ACM.

[Lopez et al. 2004] Lopez, J., Oppliger, R., and Pernul, G. (2004). Authentication and authorization infrastructures (aais): a comparative survey. *Computers & Security*, 23(7):578–590.

[Lou et al. 2004] Lou, W., Liu, W., and Fang, Y. (2004). SPREAD: enhancing data confidentiality in mobile ad hoc networks. In *Proceedings of IEEE INFOCOM*, volume 4, pages 2404–2413, Washington, DC, USA. IEEE Computer Society.

[Luk et al. 2005] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: Building customized program analysis tools with dynamic instrumentation. In *PLDI*, pages 190–200, New York, NY, USA. ACM.

[Lyytinen and Yoo 2002] Lyytinen, K. and Yoo, Y. (2002). Ubiquitous computing. *Communications of the ACM*, 45(12):63–96.

[Maas et al. 2016] Maas, A. J., Nazaré, H., and Liblit, B. (2016). Array length inference for c library bindings. In *ASE*, pages 461–471, New York, NY, USA. ACM.

[Macedo et al. 2016] Macedo, R., de Castro, R., Santos, A., Ghamri-Doudane, Y., and Nogueira, M. (2016). Self-organized SDN controller cluster conformations against DDoS attacks effects. In *2016 IEEE Global Communications Conference, GLOBECOM 2016, Washington, DC, USA, December 4-8, 2016*, pages 1–6.

[Mahalle et al. 2013] Mahalle, P. N., Anggorojati, B., Prasad, N. R., and Prasad, R. (2013). Identity authentication and capability based access control (iacac) for the internet of things. *Journal of Cyber Security and Mobility*, 1(4):309–348.

[Maler and Reed 2008] Maler, E. and Reed, D. (2008). The venn of identity: Options and issues in federated identity management. *IEEE Security Privacy*, 6(2):16–23.

[Mann 1997] Mann, S. (1997). Wearable computing: A first step toward personal imaging. *Computer*, 30(2):25–32.

[Manna and Waldinger 1971] Manna, Z. and Waldinger, R. J. (1971). Toward automatic program synthesis. *Commun. ACM*, 14(3):151–165.

[Markmann et al. 2015] Markmann, T., Schmidt, T. C., and Wählisch, M. (2015). Federated end-to-end authentication for the constrained internet of things using ibc and ecc. *SIGCOMM Comput. Commun. Rev.*, 45(4):603–604.

[Marti et al. 2000] Marti, S., Giuli, T. J., Lai, K., and Baker, M. (2000). Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 255–265, New York, NY, USA. ACM Press.

[Martin and Healey 2007] Martin, T. and Healey, J. (2007). 2006's wearable computing advances and fashions. *IEEE Pervasive Computing*, 6(1).

[Maurer et al. 2016] Maurer, M., Gerdes, J. C., Lenz, B., and Winner, H. (2016). *Autonomous driving: technical, legal and social aspects*. Springer Publishing Company, Incorporated.

[McEliece 1978] McEliece, R. J. (1978). A public-key cryptosystem based on algebraic coding theory. *Deep Space Network*, 44:114–116.

[McGrew et al. 2016] McGrew, D., Kampanakis, P., Fluhrer, S., Gazdag, S.-L., Butin, D., and Buchmann, J. (2016). State management for hash based signatures. *IACR Cryptology ePrint Archive*, 2016/357.

[McGrew et al. 2017] McGrew, D. A., Curcio, M., and Fluhrer, S. (2017). Hash-Based Signatures. Internet-draft, IETF.

[McGrew and Viega 2004] McGrew, D. A. and Viega, J. (2004). The security and performance of the galois/counter mode (GCM) of operation. In *INDOCRYPT*, volume 3348 of *LNCS*, pages 343–355. Springer.

[McMillan 1993] McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA.

[Merkle 1979] Merkle, R. C. (1979). *Secrecy, authentication and public key systems / A certified digital signature*. PhD thesis, Stanford.

[Miller 1985] Miller, V. S. (1985). Use of elliptic curves in cryptography. In *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer.

[Mouha 2015] Mouha, N. (2015). The design space of lightweight cryptography. *IACR Cryptology ePrint Archive*, 2015:303.

[Nazaré et al. 2014] Nazaré, H., Maffra, I., Santos, W., Barbosa, L., Gonnord, L., and Quintão Pereira, F. M. (2014). Validation of memory accesses through symbolic analyses. In *OOPSLA*, New York, NY, USA. ACM.

[Ndibanje et al. 2014] Ndibanje, B., Lee, H.-J., and Lee, S.-G. (2014). Security analysis and improvements of authentication and access control in the internet of things. *Sensors*, 14(8):14786–14805.

[Nethercote and Seward 2007] Nethercote, N. and Seward, J. (2007). Valgrind: a framework for heavyweight dynamic binary instrumentation. In *PLDI*, pages 89–100, New York, NY, USA. ACM.

[Neto et al. 2016] Neto, A. L. M., Souza, A. L. F., Cunha, Í. S., Nogueira, M., Nunes, I. O., Cotta, L., Gentille, N., Loureiro, A. A. F., Aranha, D. F., Patil, H. K., and Oliveira, L. B. (2016). Aot: Authentication and access control for the entire iot device life-cycle. In *SenSys*, pages 1–15. ACM.

[Nguyen et al. 2010] Nguyen, T., Al-Saffar, A., and Huh, E. (2010). A dynamic id-based authentication scheme. In *Proceedings...*, pages 248–253. Sixth International Conference on Networked Computing and Advanced Information Management (NCM), 2010.

[NIST 2006] NIST (2006). Sp 800-56A recommendation for pair-wise key establishment schemes using discrete logarithm cryptography.

[NIST 2013] NIST (2013). FIPS PUB 186: Digital signature standard (DSS).

[NIST 2016] NIST (2016). Post-quantum crypto project.

[NIST 2017] NIST (2017). Digital Identity Guidelines. *NIST Special Publication 800-63-3*. `https://doi.org/10.6028/NIST.SP.800-63-3`.

[Nogueira 2009] Nogueira, M. (2009). *SAMNAR: A survivable architecture for wireless self-organizing networks*. PhD thesis, Université Pierre et Marie Curie - LIP6.

[OASIS 2005] OASIS (2005). Authentication context for the oasis security assertion markup language (saml) v2.0.

[OASIS 2013] OASIS (2013). extensible access control markup language (xacml) version 3.0.

[Oliveira et al. 2017] Oliveira, L. B., ao Pereira, F. M. Q., Misoczki, R., Aranha, D. F., Borges, F., and Liu, J. (2017). The computer for the 21st century: Security & privacy challenges after 25 years. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*.

[Oliveira et al. 2011] Oliveira, L. B., Aranha, D. F., Gouvêa, C. P. L., Scott, M., Câmara, D. F., López, J., and Dahab, R. (2011). Tinypbc: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, 34(3):485–493.

[Paci et al. 2009] Paci, F., Ferrini, R., Musci, A., Jr., K. S., and Bertino, E. (2009). An interoperable approach to multifactor identity verification. *Computer*, 42(5):50–57.

[Paisante et al. 2016] Paisante, V., Maalej, M., Barbosa, L., Gonnord, L., and Quintão Pereira, F. M. (2016). Symbolic range analysis of pointers. In *CGO*, pages 171–181, New York, NY, USA. ACM.

[Papadimitratos and Haas 2003] Papadimitratos, P. and Haas, Z. J. (2003). Secure data transmission in mobile ad hoc networks. In *Proceedings of the ACM Workshop on Wireless Security (WiSe)*, pages 41–50, New York, NY, USA. ACM Press.

[Patel 2015] Patel, N. (2015). 90% of startups fail: Here is what you need to know about the 10%.

[Pierce 2002] Pierce, B. C. (2002). *Types and Programming Languages*. The MIT Press, 1st edition.

[Poovendran 2010] Poovendran, R. (2010). Cyber-physical systems: Close encounters between two parallel worlds [point of view]. *Proceedings of the IEEE*, 98(8):1363–1366.

[Pornin 2017] Pornin, T. (Accessed in July 2017). BearSSL - Constant-Time Mul. `https://www.bearssl.org/ctmul.html`.

[Pottie and Kaiser 2000] Pottie, G. J. and Kaiser, W. J. (2000). Wireless Integrated Network Sensors. *Communications ACM*, 43(5):51–58.

[Poulis et al. 2015] Poulis, G., Gkoulalas-Divanis, A., Loukides, G., Skiadopoulos, S., and Tryfonopoulos, C. (2015). *SECRETA: A Tool for Anonymizing Relational, Transaction and RT-Datasets*, pages 83–109. Springer International Publishing, Cham.

[Prasser and Kohlmayer 2015] Prasser, F. and Kohlmayer, F. (2015). *Putting Statistical Disclosure Control into Practice: The ARX Data Anonymization Tool*, pages 111–148. Springer International Publishing, Cham.

[Rajkumar et al. 2010] Rajkumar, R. R., Lee, I., Sha, L., and Stankovic, J. (2010). Cyber-physical systems: the next computing revolution. In *47th Design Automation Conference*. ACM.

[Refaei et al. 2005] Refaei, M. T., Srivastava, V., DaSilva, L., and Eltoweissy, M. (2005). A reputation-based mechanism for isolating selfish nodes in ad hoc networks. In *Proceedings of the Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS)*, pages 3–11, Washington, DC, USA. IEEE Computer Society.

[Regev 2005] Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of ACM STOC '05*, STOC '05, pages 84–93, New York, NY, USA. ACM.

[Reis et al. 2017] Reis, T. B. S., Aranha, D. F., and López, J. (2017). Present runs fast: Efficient and secure implementation in software. In *CHES*. Springer. To appear.

[Rellermeyer et al. 2008] Rellermeyer, J. S., Duller, M., Gilmer, K., Maragkos, D., Papageorgiou, D., and Alonso, G. (2008). The software fabric for the internet of things. In *IOT*, pages 87–104, Berlin, Heidelberg. Springer-Verlag.

[Reparaz et al. 2017] Reparaz, O., Balasch, J., and Verbauwhede, I. (2017). Dude, is my code constant time? In *DATE*, pages 1697–1702. IEEE.

[Rice 1953] Rice, H. G. (1953). Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74(1):358–366.

[Rimsa et al. 2011] Rimsa, A. A., D'Amorim, M., and Pereira, F. M. Q. (2011). Tainted flow analysis on e-SSA-form programs. In *CC*, pages 124–143. Springer.

[Rinaldi et al. 2001] Rinaldi, S. M., Peerenboom, J., and Kelly, T. (2001). Identifying, understanding, and analyzing critical infrastructure interdependencies. *IEEE Control Systems*, 21(6):11–25.

[Rivest et al. 1978] Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

[Rodrigues et al. 2016] Rodrigues, B., Pereira, F. M. Q., and Aranha, D. F. (2016). Sparse representation of implicit flows with applications to side-channel detection. In Zaks, A. and Hermenegildo, M. V., editors, *Proceedings of the 25th International Conference on Compiler Construction, CC 2016, Barcelona, Spain, March 12-18, 2016*, pages 110–120. ACM.

[Rotondi et al. 2011] Rotondi, D., Seccia, C., and Piccione, S. (2011). Access control & iot: Capability based authorization access control system. In *Proceedings...* 1st IoT International Forum.

[Russo and Sabelfeld 2010] Russo, A. and Sabelfeld, A. (2010). Dynamic vs. static flow-sensitive security analysis. In *CSF*, pages 186–199, Washington, DC, USA. IEEE.

[Sadeghi et al. 2015] Sadeghi, A. R., Wachsmann, C., and Waidner, M. (2015). Security and privacy challenges in industrial internet of things. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6.

[Salem and Hubaux 2006] Salem, N. B. and Hubaux, J.-P. (2006). Securing wireless mesh networks. *IEEE Wireless Communications*, 13(2):50–55.

[Santos et al. 2017] Santos, A. A., Nogueira, M., and Moura, J. M. F. (2017). A stochastic adaptive model to explore mobile botnet dynamics. *IEEE Communications Letters*, 21(4):753–756.

[Seitz et al. 2013] Seitz, L., Selander, G., and Gehrmann, C. (2013). Authorization framework for the internet-of-things. In *Proceedings...*, pages 1–6. IEEE 14th International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM).

[Serebryany et al. 2012] Serebryany, K., Bruening, D., Potapenko, A., and Vyukov, D. (2012). Addresssanitizer: a fast address sanity checker. In *ATC*, pages 28–28. USENIX.

[Shor 1997] Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509.

[Silva et al. 2015] Silva, E. F., Fernandes, N. C., and Muchaluat-Saade, D. (2015). Modelagem do across: Um arcabouço de aa baseado em políticas e atributos para organizações virtuais. In *Workshop de Gestão de Identidade (WGID), Anais do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg2015)*, pages 1–12. Sociedade Brasileira de Computação.

[Simon 1994] Simon, D. R. (1994). On the power of quantum computation. In *Symposium on Foundations of Computer Science (SFCS 94)*, pages 116–123, Washington, DC, USA. IEEE Computer Society.

[Soares et al. 2007] Soares, L. F. G., Rodrigues, R. F., and Moreno, M. F. (2007). Ginga-NCL: the declarative environment of the brazilian digital tv system. *J. Braz. Comp. Soc*, 12(4):1–10.

[Soto and Nogueira 2015] Soto, J. and Nogueira, M. (2015). A framework for resilient and secure spectrum sensing on cognitive radio networks. *Computer Networks*, 79:313–322.

[Souza et al. 2017] Souza, A., Cunha, Í., and B Oliveira, L. (2017). NomadiKey: User Authentication for Smart Devices based on Nomadic Keys. *International Journal of Network Management*, pages e1998–n/a.

[Souza and Wangham 2015] Souza, M. C. and Wangham, M. S. (2015). Estudo sobre interoperabilidade entre sistemas de gestao de identidades federadas em ambientes de pesquisas colaborativas. In *Workshop de Gestão de Identidade (WGID), Anais do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg2015)*, pages 632–643. Sociedade Brasileira de Computação.

[Stajano 2002] Stajano, F. (2002). *Security for Ubiquitous Computing*. John Wiley and Sons.

[Stevens et al. 2016] Stevens, M., Karpman, P., and Peyrin, T. (2016). Freestart collision for full SHA-1. In *EUROCRYPT (1)*, volume 9665 of *LNCS*, pages 459–483. Springer.

[Teixeira et al. 2015] Teixeira, F. A., Machado, G. V., Pereira, F. M. Q., Wong, H. C., Nogueira, J. M. S., and Oliveira, L. B. (2015). Siot: Securing the internet of things through distributed system analysis. In *IPSN*, pages 310–321, New York, NY, USA. ACM.

[Tekeoglu and Tosun 2015] Tekeoglu, A. and Tosun, A. S. (2015). Investigating security and privacy of a cloud-based wireless ip camera: Netcam. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6.

[Wangham et al. 2010] Wangham, M. S., de Mello, E. R., da Silva Böger, D., Gueiros, M., and da Silva Fraga, J. (2010). *Minicursos do X Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais -SBSeg 2010*, chapter Gerenciamento de Identidades Federadas, pages 1–52. Sociedade Brasileira de Computação.

[Wangham et al. 2013] Wangham, M. S., Domenech, M., and de Mello, E. R. (2013). *Minicursos do XIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2013*, chapter Infraestruturas de Autenticação e de Autorização para Internet das Coisas. Sociedade Brasileira de Computação.

[Waters 2011] Waters, B. (2011). Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography*, volume 6571 of *LNCS*, pages 53–70. Springer.

[Weiser 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific american*, 265(3):94–104.

[Weiser 1993] Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84.

[Wilson and Lam 1995] Wilson, R. P. and Lam, M. S. (1995). Efficient context-sensitive pointer analysis for c programs. In *PLDI*, pages 1–12, New York, NY, USA. ACM.

[Wu et al. 2017] Wu, M., Pereira, F. M. Q., Liu, J., Ramos, H. S., Alvim, M. S., and Oliveira, L. B. (2017). Proof-Carrying Sensing: Towards a Real-World Authentication Scheme to Cyber-Physical-Human Systems. In *Conference on Embedded Networked Sensor Systems (SenSys)*.

[Yang et al. 2004] Yang, H., Luo, H., Ye, F., Lu, S., and Zhang, L. (2004). Security in mobile ad hoc networks: challenges and solutions. *IEEE Wireless Communications*, pages 38–47.

[Yi et al. 2001] Yi, S., Naldurg, P., and Kravets, R. (2001). Security-aware ad hoc routing for wireless networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 299–302, New York, NY, USA. ACM Press.

[Zapata 2002] Zapata, M. G. (2002). Secure ad hoc on-demand distance vector routing. *ACM SIGMOBILE*, 6(3):106–107.

[Zeng et al. 2011] Zeng, D., Guo, S., and Cheng, Z. (2011). The web of things: A survey. *Journal of Communications*, 6(6).

[Zhang et al. 2008] Zhang, C., Song, Y., and Fang, Y. (2008). Modeling secure connectivity of self-organized wireless ad hoc networks. In *IEEE INFOCOM*, pages 251–255.

[Zhang and Liu 2011] Zhang, G. and Liu, J. (2011). A model of workflow-oriented attributed based access control. *International Journal of Computer Network and Information Security (IJCNIS)*, 3(1):47–53.

Capítulo 2

Mecanismos de segurança baseados em *hardware*: uma introdução à arquitetura Intel SGX

Newton C. Will, Rafael C. R. Condé, Carlos A. Maziero
Programa de Pós-Graduação em Informática - Universidade Federal do Paraná (UFPR)

Abstract

Data confidentiality is becoming more and more important to the corporate and domestic computer users. In addition, it is extremely important to ensure security in the execution of applications that manipulate such data, which must have their confidentiality and integrity guaranteed. In this way, there are several solutions that aim to maintain the data confidentiality and integrity, as well as security in the execution of applications. This chapter presents a review of the most relevant hardware-based security mechanisms and their evolution, culminating with the presentation of the Intel SGX architecture.

Resumo

A confidencialidade dos dados está se mostrando cada vez mais importante para os usuários de computadores, seja em um ambiente corporativo ou mesmo em um ambiente doméstico. Além disso, é extremamente importante garantir a segurança na execução das aplicações que manipulam tais dados, cuja confidencialidade e integridade devem ser garantidas. Neste sentido, existem várias soluções que se propõem a manter a confidencialidade e integridade dos dados e também a segurança na execução de aplicações. Este capítulo visa apresentar uma revisão dos mecanismos de segurança baseados em hardware mais relevantes e sua evolução, culminando na apresentação da arquitetura Intel SGX.

2.1. Introdução

Atualmente a tecnologia está presente na vida de muitas pessoas, que entregam seus dados a dispositivos como computadores, *smartphones*, *tablets*, e até mesmo a serviços de armazenamento *online*. Os dados e informações presentes nesses dispositivos, ou meios de armazenamento, são dos mais variados tipos, passando desde fotos de família e senhas, aos mais diversos serviços, agendas pessoais, dados médicos, bancários e mesmo informações sobre o trabalho, que podem ser extremamente importantes para as empresas. Em virtude desta expansão no uso de dispositivos digitais, pessoas e empresas têm ficado cada vez mais dependentes de recursos computacionais e da Internet, para exercer desde atividades simples do dia a dia até complexas atividades industriais.

A McAfee Labs estima que, devido à convergência da Internet das Coisas (IoT), 200 bilhões de dispositivos estejam conectados à Internet em 2020. Ainda segundo a McAfee Labs, em 2006 as empresas enfrentavam, em média, 25 novas ameaças por dia, enquanto hoje 500.000 ameaças são vistas diariamente [Weafer 2016]. O FBI estima que crimes cibernéticos geram 67,2 bilhões de dólares de prejuízo por ano [FBI 2005]. Desta forma, não causa estranheza que os ataques a recursos computacionais estejam ficando cada vez mais sofisticados e difíceis de serem detectados e contidos.

Brechas de segurança em dados e aplicações são alvos constantes de ataques, que atingem até mesmo grande empresas, como o ocorrido com a Apple, que supostamente estaria sendo chantageada por uma equipe de *hackers* que teria conseguido acesso a 300 milhões de contas de *e-mail* de usuários [Cox 2017]. Ataques como este causam grandes prejuízos às empresas, podendo até levá-las à falência, em casos extremos.

Conforme descrito por [Hoekstra et al. 2013], as práticas para o desenvolvimento de aplicações seguras são padronizadas em muitas empresas, tendo o teste de segurança como parte chave na validação do *software*. Ainda assim a confidencialidade e integridade dos dados dependerá da correta utilização de outros *softwares* que estarão sendo utilizados no mesmo ambiente, alguns dos quais podem ter privilégios para inspecionar a memória ou outros meios de armazenamento no dispositivo. Além disso, cabe ao usuário final o bom senso no uso das aplicações, o qual deverá seguir as boas práticas e adotar uma política cautelosa para a execução de diversas aplicações, seja no ambiente de trabalho ou para uso pessoal.

Neste contexto, grandes esforços têm sido empenhados no sentido de garantir a segurança de dados e execução de aplicações sensíveis, seja através da exploração de tecnologias existentes, ou através da pesquisa e desenvolvimento de novas tecnologias. No intuito de expandir cada vez mais o nível de proteção, tecnologias baseadas na implementação em *hardware* têm ganho grande relevância.

Este capítulo apresenta uma revisão das principais tecnologias de segurança baseadas em *hardware*, focando principalmente na recentemente proposta arquitetura Intel *Software Guard Extensions* (SGX), descrevendo as suas funcionalidades e usos, incluindo teoria e prática. Este capítulo foi dividido em sete seções, com a primeira delas trazendo esta introdução.

A Seção 2.2 faz uma revisão das diversas técnicas e mecanismos de segurança que foram ou são utilizados ao longo dos anos para manter a segurança das aplicações e

confidencialidade dos dados, com o objetivo de apresentar uma evolução destas técnicas. São abordados desde os primeiros *hardwares* criptográficos, passando pela especificação do TPM (*Trusted Platform Module*), implementações como o AMD *Secure Processor* e a Intel *Trusted Execution Technology*, a arquitetura ARM *TrustZone*, que permite uma comunicação segura entre um processador ARM e seus demais dispositivos compatíveis, uma breve introdução à tecnologia Intel SGX e, por fim, um comparativo entre esses mecanismos.

Na Seção 2.3 é detalhada a arquitetura Intel *Software Guard Extensions* (SGX), que foi incluída na família de processadores *Skylake* no final de 2015, e permite que uma aplicação seja encapsulada dentro de um enclave, que é gerenciado pelo próprio processador.

Na Seção 2.4 são apresentados alguns trabalhos que já fazem uso da arquitetura SGX, seja em caráter de produção ou experimental; a Seção 2.5 lista as limitações da arquitetura Intel SGX e alguns tópicos de interesse em pesquisa, além de um resumo das principais vulnerabilidades conhecidas do SGX.

A Seção 2.6 apresenta os recursos disponíveis no *Software Development Kit* (SDK) do SGX para a construção de aplicações, trazendo, inclusive, exemplos práticos. Por fim, a Seção 2.7 traz as considerações finais acerca dos tópicos que são apresentados neste capítulo.

2.2. Mecanismos de Segurança Baseados em *Hardware*

Esta seção visa apresentar um histórico da evolução dos principais mecanismos de segurança baseados em *hardware* que, de alguma maneira, pavimentaram a criação da arquitetura Intel SGX.

2.2.1. Primeiros *Hardwares* Criptográficos

[Anderson et al. 2006] estabelece que um cripto-processador típico é um processador embarcado dedicado que executa um conjunto pré-definido de operações criptográficas usando chaves internas, protegidas do ambiente externo, que se comunicam com o computador principal. Os cripto-processadores devem ser resistentes a ataques físicos e por *software*. Operações criptográficas são naturalmente custosas, e a utilização de um *hardware* dedicado para esta tarefa contribui para reduzir os impactos negativos no desempenho. Processadores deste tipo são utilizados há muito tempo em aplicações específicas, como em sistemas bancários e aplicações militares. No entanto, com a crescente necessidade da proteção de dados sensíveis nas mais diversas aplicações, cripto-processadores têm ganho maior relevância e diferentes implementações têm sido empregadas.

Dentre estas implementações, [Bossuet et al. 2013] destaca os seguintes mecanismos:

- **GPP Customizado:** Utiliza processadores de propósito geral customizados para implementação de algoritmos criptográficos eficientes. Eles absorvem operações criptográficas específicas, como o *Data Encryption Standard* (DES) ou o *Advanced Encryption Standard* (AES). Estas soluções melhoram o desempenho, mas não a segurança, uma vez que as chaves criptográficas são armazenados em memória e

tratadas como dados ordinários da aplicação, podendo sofrer ataques por software. Outra dificuldade é a seleção do conjunto de instruções, visto que algoritmos de chave simétrica e assimétrica têm necessidades distintas.

- **Cripto-coprocessador:** Implementação em *hardware* customizado, altamente eficiente para funções criptográficas específicas. Eles podem conter um ou mais núcleos de processamento, mas não são programáveis e sim controlados, configurados ou parametrizados usando o processador a que estão acoplados. São usados para acelerar cálculos criptográficos. Seu uso é indicado para aplicações com necessidade de alta vazão.

- **Cripto-processador:** É um módulo de hardware que se difere do coprocessador por ser programável, com um conjunto de instruções dedicadas para obter eficiência em funções criptográficas. Cripto-processadores contam com uma ou mais unidades lógico-aritméticas (ULAs) projetadas especificamente para cálculos criptográficos; por isso, para realizar operações convencionais é necessário o uso de um processador de propósito geral. Esta arquitetura é mais enxuta que as duas anteriores, podendo ser considerada relativamente flexível, uma vez que as ULAs podem ser reconfiguradas. Outra grande diferença é que as chaves de criptografia são armazenadas internamente pelo cripto-processador, e não mais na memória principal, como nos casos anteriores. Apesar de executar somente cálculos criptográficos, cripto-processadores são autônomos e representam um ótimo custo-benefício entre desempenho e capacidade de processamento. São boas opções para inclusão em *Multi-Processor-System-on-a-Chip* (MP-SoC).

- ***Cripto array*:** É um acelerador criptográfico que, assim como os cripto-processadores, precisa ser acoplado a processadores de propósito geral. Esta arquitetura usa vetores de núcleos de processamento para realizar as operações criptográficas. Normalmente os núcleos são reconfiguráveis para obtenção de uma maior flexibilidade.

2.2.2. *Trusted Platform Module* (TPM)

O *Trusted Computing Group* [TCG 2008] é uma organização internacional de normas que conta com cerca de 140 empresas, sendo responsável pela especificação de uma série de padrões de segurança, tais como IPSEC (*Internet Protocol Security Protocol*), IKE (*Internet Key Exchange*), VPN (*Virtual Private Network*), PKI (*Public Key Infrastructure*), S/MIME *(Secure Multi-purpose Internet Mail Extensions*), SSL (*Secure Socket Layer*), SET (*Secure Electronic Transaction*), dentre outros [Amin et al. 2008]. Uma das frentes de trabalho atuais do TCG é a especificação dos padrões para o *Trusted Computing Module*.

TPM (*Trusted Platform Module*) é um *chipset* capaz de armazenar artefatos usados para autenticar uma plataforma computacional, o que inclui senhas, certificados e chaves criptográficas. Um TPM também pode ser utilizado para armazenar as medidas da plataforma[1], para garantir que ela continua confiável. As especificações destacam

[1]Medidas (*measurements*) são informações previamente colhidas da plataforma em uma situação sabidamente segura, como *hashes* criptográficos, que podem ser usadas posteriormente para atestar sua integridade.

ainda que os dois elementos fundamentais para o estabelecimento de um ambiente seguro são: a) garantir que a plataforma pode provar que é quem ela diz ser (autenticação); e b) provar a entidades externas que a plataforma é confiável e não foi adulterada (atestação) [TCG 2008]. O conceito de TPM é aplicável a outros equipamentos além de PCs, tais como *smartphohes* ou equipamentos de rede.

Conforme [TCG 2016], o TPM é um componente do sistema que tem um estado separado do sistema hospedeiro. A interação entre o sistema hospedeiro e o TPM é feita somente através de uma interface definida nas especificações do TPM. O TPM é construído em recursos físicos dedicados exclusivamente a ele. Normalmente são implementados em um único chip acoplado ao sistema (tipicamente um PC) e são compostos de processador, RAM, ROM e memória *flash*, e sua única interface é um barramento LPC. A Figura 2.1 mostra a composição básica de um TPM. O sistema hospedeiro não pode alterar valores diretamente na memória do TPM, isto só pode ser feito através do *buffer* de E/S da interface, dedicado para este fim.

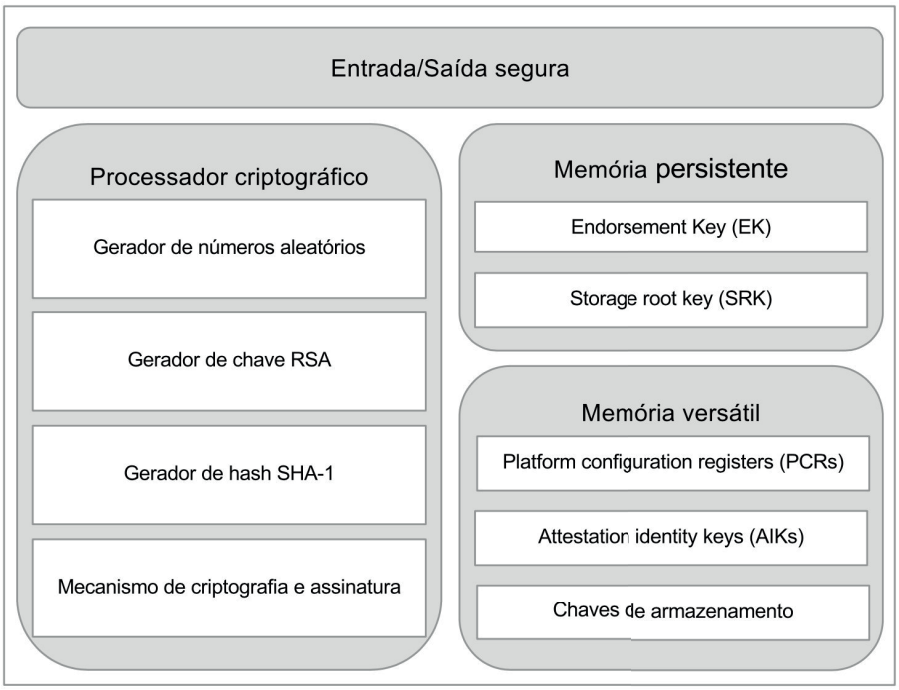


Figura 2.1. Arquitetura de um TPM (adaptado de [Vacca 2016]).

Ainda segundo as especificações [TCG 2016], outra implementação possível é executar o código seguro no processador do hospedeiro enquanto o mesmo se encontra em um modo de execução especial. Nesse caso, a memória do sistema é particionada pelo *hardware* de modo que a parte dedicada ao TPM fique inacessível, exceto quando o processador se encontra no modo especial. Quando o processador alterna entre os modos, ele sempre inicia a execução em pontos de entrada específicos. Existem diversos esquemas de implementação possíveis para isto, tais como *System Management Mode*, *TrustZone* e virtualização.

Dentro da construção de uma plataforma confiável existem alguns elementos cujas falhas não podem ser detectadas, e portanto precisam ser considerados confiáveis. Assim,

estes elementos devem ser projetados de modo a garantir que seu comportamento será sempre aquele desejado. [TCG 2016] define estes elementos como *Root of Trust*. Este conjunto de elementos deve ser o mínimo para prover as funcionalidades necessárias a uma plataforma confiável. Foram estabelecidos três *Roots of Trust* necessários ao TPM:

- ***Root of Trust for Measurement* (RTM):** Responsável por gerar informações relevantes à integridade da plataforma (suas medidas) e armazenar no RTS (próximo elemento). Normalmente este elemento é a CPU controlada pelo *Core Root of Trust for Measurement* (CRTM), que é o primeiro conjunto de instruções executadas quando uma nova cadeia de confiança é estabelecida, por exemplo, quando o sistema é reiniciado. O CRTM envia valores que indicam sua identidade ao RTS.
- ***Root of Trust for Storage* (RTS):** Usado para armazenar elementos como as medidas da plataforma, a *Endorsement Key* (EK)[2] e a *Storage Root Key* (SRK)[3]. A memória do TPM tem seu acesso bloqueado a qualquer entidade, exceto o próprio TPM. Se, no entanto, o elemento de memória contém informações não sensíveis, como no caso de algum *Platform Configuration Register* (PCR), o TPM permite a leitura.
- ***Root of Trust for Reporting* (RTR):** Responsável por reportar acerca do conteúdo do RTS para determinadas finalidades. Normalmente o *report* é um resumo criptográfico assinado digitalmente do conteúdo de valores selecionados do TPM. A interação entre o RTR e o RTS é crítica, sua implementação deve prevenir todas as formas de ataques físicos ou por *software* e prover os resumos criptográficos com precisão.

Estes três *Roots of Trust* usam certificação e atestação para fornecer evidências da confiabilidade da informação. Por fim, a plataforma mantém o *log* das suas mudanças de estado internas, para validação posterior.

2.2.3. Intel *Trusted Execution Technology*

Intel *Trusted Execution Technology* (Intel TXT) é uma tecnologia da Intel que utiliza componentes de *hardware* para criar ambientes de execução seguros contra ameaças físicas e virtuais, de modo a complementar proteções em tempo de execução, como *softwares* de anti-virus.

Conforme [Greene 2012], o Intel TXT necessita de alguns componentes fundamentais: extensões integradas ao Intel Xeon e *chipset* Intel, módulos de código autenticados (*Authenticated Code Modules* - ACMs), ferramentas de LCP (*Launch Control Policy*), um módulo TPM, BIOS e hipervisor ou sistema operacional habilitado para o Intel TXT. Se qualquer destes componentes faltar ou falhar, a plataforma irá ser carregada no seu estado não-confiável.

A execução segura é obtida com inicializações verificadas através de um ambiente de inicialização medido (*Measured Launch Environment* - MLE) que permite que os elementos críticos da inicialização possam ser confrontados contra as medições destes elementos vindas de uma fonte confiável. A medição consiste na criação de um identificador

[2] A EK é criada durante a fabricação e não pode ser alterada; ela é usada no processo de autenticação.

[3] A SRK é usada para armazenamento cifrado, sendo gerada em tempo de execução.

criptográfico único para cada componente aprovado na inicialização. O provisionamento destas medições confiáveis é feito através de um microcódigo autenticado (ACM) embarcado; uma vez realizadas as medições, elas são gravadas e travadas dentro do TPM. A Intel provê, ainda, um mecanismo baseado em *hardware* para bloquear a inicialização de código que não confere com aqueles previamente aprovados.

Existem dois métodos de estabelecimento de confiança através de medição (RTM): *estático* (S-RTM) e *dinâmico* (D-RTM). O S-RTM começa a medição em um evento de *reset* da plataforma com a medição da BIOS, passando pela medição do hipervisor (ou sistema operacional) e seus componentes. As medições realizadas são confrontadas com aquelas armazenadas no TPM. Se as medições forem equivalentes àquelas consideradas seguras, o sistema é carregado com o indicativo de confiável. Caso contrário, o sistema Intel fornecerá um indicativo de inicialização não-confiável. Apesar de ser mais simples, a S-RTM resulta em uma TCB (*Trusted Computing Base*[4]) grande e difícil de gerenciar. Uma vez estabelecida a confiança, qualquer mudança ou atualização da plataforma ocasionará na necessidade de migração ou atualização dos segredos. Já no D-RTM, as propriedades de confiança dos componentes podem ser ignoradas até a chamada de um evento seguro. Os componentes anteriores ao evento seguro serão excluídos da TCB e não poderão ser executados depois que a confiança do sistema for estabelecida. Isto resultará em uma TCB menor, o que é desejável.

Além da inicialização verificada, o Intel TXT dispõe de um mecanismo de políticas para a criação e implementação de listas de códigos executaveis aprovados ou sabidamente confiáveis. Este mecanismo é chamado de *Launch Control Policy* (LCP). Um mecanismo de proteção de segredos também é implementado através de métodos auxiliados por *hardware*, que removem dados residuais em reinicializações impróprias (ataques por *reset*). Por fim, o Intel TXT também tem a habilidade de produzir credenciais com as medições da plataforma para realizar atestação para usuários, ou sistemas locais ou remotos, de forma a completar o processo de verificação de confiança e suportar atividades de conformidade e auditoria.

Quando utilizado em ambiente virtualizado, o Intel TXT também faz uso extensivo da Tecnologia de Virtualização Intel (Intel VT) para prover proteção contra DMAs não autorizadas e garantir o isolamento de dados no sistema.

2.2.4. ARM *TrustZone*

A abordagem da arquitetura ARM para atingir as metas propostas pelo TCG também parte do conceito de uma plataforma confiável, a ARM *TrustZone*, que é uma arquitetura de *hardware* que estende o quesito de segurança a todo o *design* do sistema, permitindo que qualquer parte do sistema seja protegida. A tecnologia *TrustZone*, que está disponível nos controladores Cortex-A, Cortex-M23 e Cortex-M33, fornece uma infraestrutura básica que permite aos projetistas de SoCs (*System-on-Chip*) escolher uma gama de componentes que podem auxiliar em funções específicas dentro de um ambiente seguro. O objetivo

[4]Conjunto de todos os componentes de *hardware*, *firmware* e/ou *software* que são críticos para a segurança do sistema, no sentido de que os erros ou vulnerabilidades que ocorrem dentro do TCB podem prejudicar as propriedades de segurança do sistema inteiro. Em contrapartida, partes de um sistema fora do TCB não podem se comportar de forma a que obtenham mais privilégios do que os concedidos de acordo com a política de segurança.

principal da arquitetura é habilitar a construção de um ambiente programável que assegure a confidencialidade e integridade de quase todos os ativos a serem protegidos de ataques específicos, podendo ser utilizada para construir um conjunto de soluções de segurança que não são possíveis com os métodos tradicionais [ARM 2009].

Com a arquitetura ARM *TrustZone* o sistema pode ser isolado em dois estados lógicos: um estado seguro e um estado inseguro (Figura 2.2). Estes estados também são sinalizados para todos os dispositivos periféricos, através do barramento do sistema, permitindo-lhes tomar decisões de controle de acesso com base no estado atual do sistema. O mecanismos responsável pela troca de contexto entre os dois estados é chamado de *monitor*.

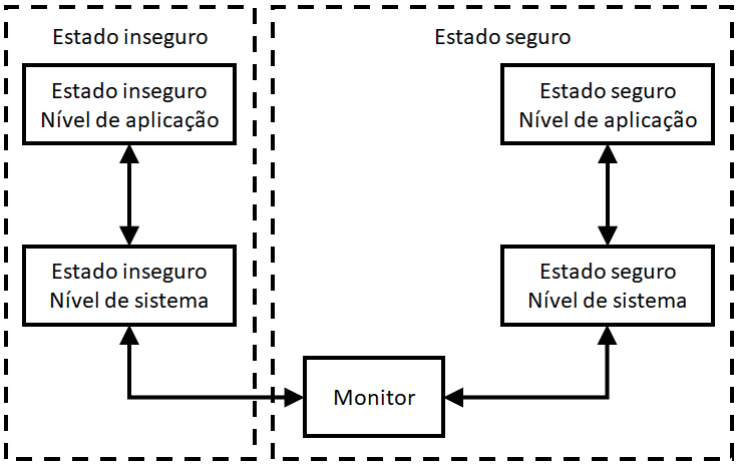


Figura 2.2. Modos de execução na arquitetura ARM *TrustZone* (adaptado de [ARM 2009]).

Quando uma aplicação está sendo executada em um estado seguro, ela pode isolar partes da memória para seu próprio uso, impedindo que aplicações sendo executadas em um estado inseguro acessem esses locais. Isto é garantido pelo controlador de memória que utiliza as premissas da arquitetura *TrustZone*, que fornece controle de acesso para regiões de memória com base no estado atual do sistema através do *TrustZone Address Space Controller*. Este particionamento de memória pode ser definido de forma fixa, ou programável em tempo de execução.

Uma aplicação em estado seguro também pode forçar que certas interrupções ou exceções sejam capturadas somente em um estado seguro, sendo que este controle também fica a cargo do controlador de interrupções. Além disso, o sistema também pode bloquear o acesso a determinados dispositivos para aplicações que não estejam sendo executadas em um estado seguro, garantindo a exclusividade de uso desses dispositivos somente a aplicações seguras [Lesjak et al. 2015].

Os sistemas que suportam *TrustZone* vêm, ainda, com *hardware* complementar para suporte a chaves seguras não-voláteis, *Real Time Clock* seguro e aceleradores criptográficos.

Para a consolidação de um ambiente seguro, um sistema operacional e um mecanismo de *boot* seguros devem ser desenvolvidos, utilizando os recursos do *TrustZone*. A combinação do isolamento por *hardware TrustZone*, *boot* seguro e sistema operacional seguro constituem um ambiente de execução confiável (*Trusted Execution Environment*

- TEE), com as propriedades de segurança do TEE suportando a confidencialidade e a integridade de múltiplas aplicações confiáveis [ARM 2009].

2.2.5. AMD *Secure Processor*

Também conhecido como *Platform Security Processor* (PSP), o AMD *Secure Processor* é um subsistema de segurança em *hardware* dedicado integrado ao chip do processador, que executa de modo independente dos núcleos do processador principal. Esse subsistema proporciona um ambiente para tratamento de dados sensíveis isolado do sistema principal.

Conforme [Arthur and Challener 2015], para ser uma uma base confiável em *hardware*, podendo ser usada para o estabelecimento da cadeia de confiança, o PSP faz uso de um microcontrolador ARM *TrustZone* de 32 *bits* mas, apesar da utilização da arquitetura *TrustZone*, o PSP não é um núcleo virtual, mas um processador real, fisicamente separado, integrado ao SoC, que dispõe de uma SRAM dedicada e acesso direto ao coprocessador criptográfico. Além disso, o PSP dispõe de acesso aos recursos de memória do sistema e uma DRAM criptografada, com isolamento implementado em *hardware*.

O PSP também contém um coprocessador criptográfico composto de um gerador de números aleatórios, mecanismos para processamento de algoritmos criptográficos (AES, RSA e outros) e um bloco para armazenamento de chaves. Este bloco é composto de duas áreas de armazenamento: uma usada por *software* privilegiado, cuja leitura não é possível; e outra onde chaves podem ser carregadas, utilizadas e descartadas, em operações convencionais tanto de *software* executando no PSP ou no sistema operacional convencional. Também há uma lógica implementada em *hardware* para inicialização segura do núcleo da CPU e uma chave única da plataforma, que é distribuída para o bloco de armazenamento do coprocessador criptográfico durante o processo de inicialização.

Para garantir uma inicialização confiável, o PSP implementa o *Hardware Validated Boot* (HVB), que é uma forma de *boot* seguro não-modificável, implementado na ROM do SoC, que verifica a integridade da BIOS. A ROM faz a validação de uma chave de inicialização e então usa essa chave para validar o *firmware* do PSP que, por sua vez, é carregado e inicia a execução da aplicação do sistema.

2.2.6. Intel *Software Guard Extensions*

O *Intel Software Guard Extensions*, ou *Intel SGX*, é uma extensão ao conjunto de instruções da arquitetura *x86* que permite que aplicações possam ser executadas em uma área protegida de memória, chamada de *enclave*, que irá conter o código e os dados da aplicação. Um enclave é uma área protegida no espaço de endereçamento da aplicação, conforme mostrado na Figura 2.3, que garante a confidencialidade e integridade dos dados, evitando que esses dados sejam acessados por *malwares* que estejam em execução, e até mesmo por outros *softwares* com alto privilégio de execução, como monitores de máquinas virtuais, BIOS, e o próprio sistema operacional [McKeen et al. 2013, Jain et al. 2016].

Para garantir a confidencialidade e integridade dos dados, a arquitetura SGX adicionou novas instruções, uma nova arquitetura de processador e um novo modelo de execução, que incluem o carregamento do enclave em uma área protegida de memória, o acesso de recursos via mapeamento de tabelas de páginas e o escalonamento de aplicações dentro de enclaves. Após a aplicação ser carregada dentro de um enclave, ela fica protegida de todo

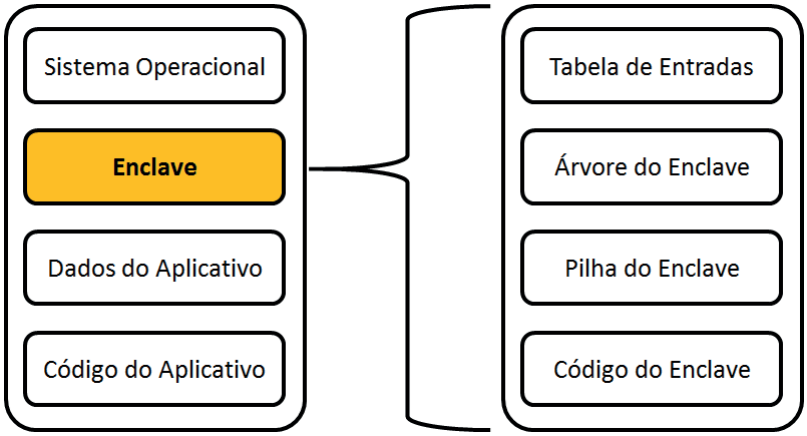


Figura 2.3. Enclave dentro do Espaço de Endereçamento Virtual da Aplicação (adaptado de [McKeen et al. 2013]).

e qualquer acesso externo ao enclave, inclusive de outras aplicações que estejam em outros enclaves. As tentativas de alterações não autorizadas de conteúdo dentro de um enclave são detectadas e impedidas, ou sua execução é abordada. Enquanto os dados do enclave estão tramitando entre os registradores e os outros blocos do processador, o acesso não autorizado é evitado utilizando os mecanismos de controle de acesso internos do próprio processador. Quando os dados são escritos na memória, eles são automaticamente cifrados e a integridade é mantida, evitando sondagens de memória ou outras técnicas para visualizar, modificar ou substituir os dados contidos em um enclave [Costan and Devadas 2016].

A cifragem de memória é feita utilizando os algoritmos padrão de criptografia, contendo proteções contra ataques de repetição. O fato de conectar os módulos de memória DRAM em outro sistema apenas dará acesso aos dados em sua forma cifrada; além disso, a chave de criptografia é armazenada em registradores dentro da CPU, não sendo acessível a componentes externos à mesma, e é alterada aleatoriamente em cada evento de hibernação ou reinício do sistema [Intel 2016b]. Maiores detalhes sobre a arquitetura, funcionalidades e implementação do SGX serão abordados na Seção 2.3.

2.2.7. Comparativo entre os mecanismos apresentados

Esta seção apresentou de forma sucinta uma coletânea dos principais mecanismos de segurança baseados em *hardware* disponíveis para sistemas computacionais modernos. A Tabela 2.1 apresenta um comparativo entre as principais características dos mecanismos abordados neste texto. Nota-se que a tecnologia Intel SGX abrange as principais características necessárias para o desenvolvimento de aplicações seguras. Por outro lado, o AMD *Secure Processor* consegue as mesmas características através da utilização do *TrustZone* e do TPM em um SoC.

2.3. Intel *Software Guard Extensions* (SGX)

O Intel SGX, introduzido na Seção 2.2.6, traz um novo conceito na construção de aplicações seguras, possibilitando o isolamento da parte sensível da aplicação dentro de uma estrutura denominada *enclave*. O *Intel Software Guard Extensions Developer Guide* [Intel 2016a] descreve o enclave como uma entidade de *software* monolítica que reduz a

Tabela 2.1. Comparativo entre os mecanismos de proteção baseados em *hardware*.

Tecnologia	Armazenamento seguro	Atestação	Isolamento de memória	Acelerador criptográfico
Cripto-processadores				✓
TPM	✓	✓		✓
Intel TXT	✓	✓	✓	
ARM TrustZone			✓	✓
AMD Secure Processor	✓	✓	✓	✓
Intel SGX	✓	✓	✓	✓

Trusted Computing Base (TCB) de um aplicativo para um sistema de tempo de execução confiável, com o domínio não confiável controlando a ordem em que as funções dentro do enclave são chamadas.

Em uma análise em alto nível, a arquitetura SGX oferece um modelo de programação muito semelhante ao utilizado para o desenvolvimento de aplicações regulares nos diversos sistemas operacionais, com o uso de *Dynamic Linked Library*, *Shared Objects*, ou similares, apenas com algumas diferenças em como a aplicação é projetada e desenvolvida para fazer uso dos recursos providos pelo SGX, já que a utilização indevida desses recursos pode levar a vulnerabilidades que podem ser exploradas posteriormente.

O código-fonte de um enclave não tem muitas diferenças quando comparado com o código-fonte de uma aplicação regular. No entanto, o código do enclave é carregado de uma maneira especial, de forma que uma vez que o enclave foi inicializado, o seu código e o restante da aplicação não podem ler diretamente os dados que residem no ambiente protegido ou alterar o comportamento do código dentro do enclave sem detecção. Por esta razão, é essencial que o desenvolvedor identifique quais componentes e recursos da aplicação devem ser protegidos, para que somente estes residam dentro de enclaves.

Normalmente, uma biblioteca compartilhada contém seções de dados e código correspondentes às variáveis e/ou objetos e funções e/ou métodos implementados, com o sistema operacional alocando um *heap* quando o processo que usa a biblioteca compartilhada é carregado, e uma pilha para cada *thread* gerada dentro do processo. Da mesma forma, uma biblioteca de enclaves contém seções de código e dados que serão carregadas na memória protegida (*Enclave Page Cache* - EPC) quando o enclave for criado. Existem também os metadados, que não são carregados no EPC, pois são utilizados pela aplicação que irá inicializar o enclave para determinar como carregá-lo corretamente no EPC. Os metadados definem um número de contextos de *threads* confiáveis, que inclui a pilha confiável e um *heap* criado na inicialização do enclave, sendo que ambos são necessários para suportar um ambiente de execução confiável. Além disso, os metadados compõem, juntamente com a assinatura do enclave, um certificado essencial para assegurar a autenticidade e a integridade do mesmo.

Mesmo que um enclave possa ser fornecido como um arquivo de biblioteca compartilhada, definir quais porções de código e dados serão colocadas dentro do enclave e quais permanecerão fora deste, na porção não protegida da aplicação, é um aspecto chave do desenvolvimento do enclave, sendo este o primeiro passo para projetar um aplicativo habili-

tado para fazer uso da arquitetura SGX. Esta é uma tarefa que fica a cargo do desenvolvedor da aplicação, já que este é quem melhor conhece o aplicativo a ser desenvolvido.

2.3.1. Organização de Memória do Enclave

O código e os dados do enclave são armazenados em uma parte reservada da memória, chamada *Processor Reserved Memory* (PRM), que é um subconjunto da memória principal que não pode ser acessado diretamente por outras aplicações. O controlador de memória da CPU também rejeita transferências via DMA para o PRM, de forma a protegê-lo do acesso através de dispositivos periféricos.

O PRM é uma área contígua de memória, tendo o seu tamanho e seu endereço de início como um valor inteiro e potência de dois, o que permite que seus endereços possam ser verificados por um *hardware* simples e barato. O PRM é um detalhe da microarquitetura que pode mudar em implementações futuras do SGX.

O conteúdo dos enclaves e as estruturas de dados associadas a estes são armazenados em um subconjunto do PRM, chamado de *Enclave Page Cache* (EPC). O EPC, por sua vez, é subdividido em páginas de 4 KBytes, que podem ser atribuídas a diferentes enclaves. Isso permite que se tenha múltiplos enclaves em execução no sistema ao mesmo tempo, o que é uma necessidade para ambientes multiprocessados.

O EPC é gerenciado pelo mesmo *software* que gerencia o restante da memória física do computador, que pode ser um hipervisor ou um *kernel* do sistema operacional. O gerenciador de memória física utiliza as instruções do SGX para alocar páginas não utilizadas por enclaves e para liberar páginas previamente alocadas [Costan and Devadas 2016].

Como o EPC está contido dentro do PRM, quaisquer *softwares* que não utilizem enclaves não podem ter acesso a este, o que garante um isolamento do enclave, mas também cria alguns obstáculos para um software carregar trechos de código e dados iniciais para um enclave recém criado. Esses obstáculos são contornados pelo uso de instruções do próprio SGX que permitem carregar conteúdo de outras páginas de memória para as páginas do EPC. Assim, quem é responsável por carregar as páginas de memória para o EPC é o *software* não confiável. Tendo isso em vista, este processo é validado pelo SGX, que pode recusar qualquer operação que comprometa as garantias de segurança dos enclaves, como, por exemplo, a tentativa de atribuir a mesma página no EPC para dois enclaves diferentes.

O armazenamento do EPC na memória principal é protegido cifrando os dados, de forma a possibilitar uma defesa contra ataques à memória, tanto por *software* quanto por *hardware*. Para isso, uma unidade de *hardware* chamada *Memory Encryption Engine* (MEE) cuida da criptografia e da integridade dos dados quando estes estão sendo transferidos entre a memória e o processador, sendo que a região de memória onde um MEE opera é chamado de *Região MEE*. Um processador que suporta a arquitetura SGX e implementa o EPC em uma região de memória cifrada também dará suporte para a BIOS reservar um intervalo de memória chamado *Processor Reserved Memory* (PRM), que poderá ser protegido por uma ou mais regiões MEE.

O processador bloqueia qualquer acesso ao PRM vindo de agentes externos, tratando esses acessos como referências não existentes na memória. Já o acesso a páginas de memória dentro de um enclave, utilizando instruções de memória como MOV, é verificado

pelo hardware seguindo o fluxograma descrito na Figura 2.4. Inicialmente verifica-se se o processo está sendo executado dentro de um enclave e se o mesmo pertence ao enclave que ele está querendo acessar; em caso afirmativo, o acesso é liberado; caso contrário, o acesso é bloqueado, tratando-o como uma referência a uma posição de memória inexistente. Da mesma forma, as tentativas de acesso às páginas de memória dentro de um enclave por parte de um processo que não está sendo executado em um enclave, também são tratadas como referências inexistentes [McKeen et al. 2013, Intel 2014, Costan and Devadas 2016].

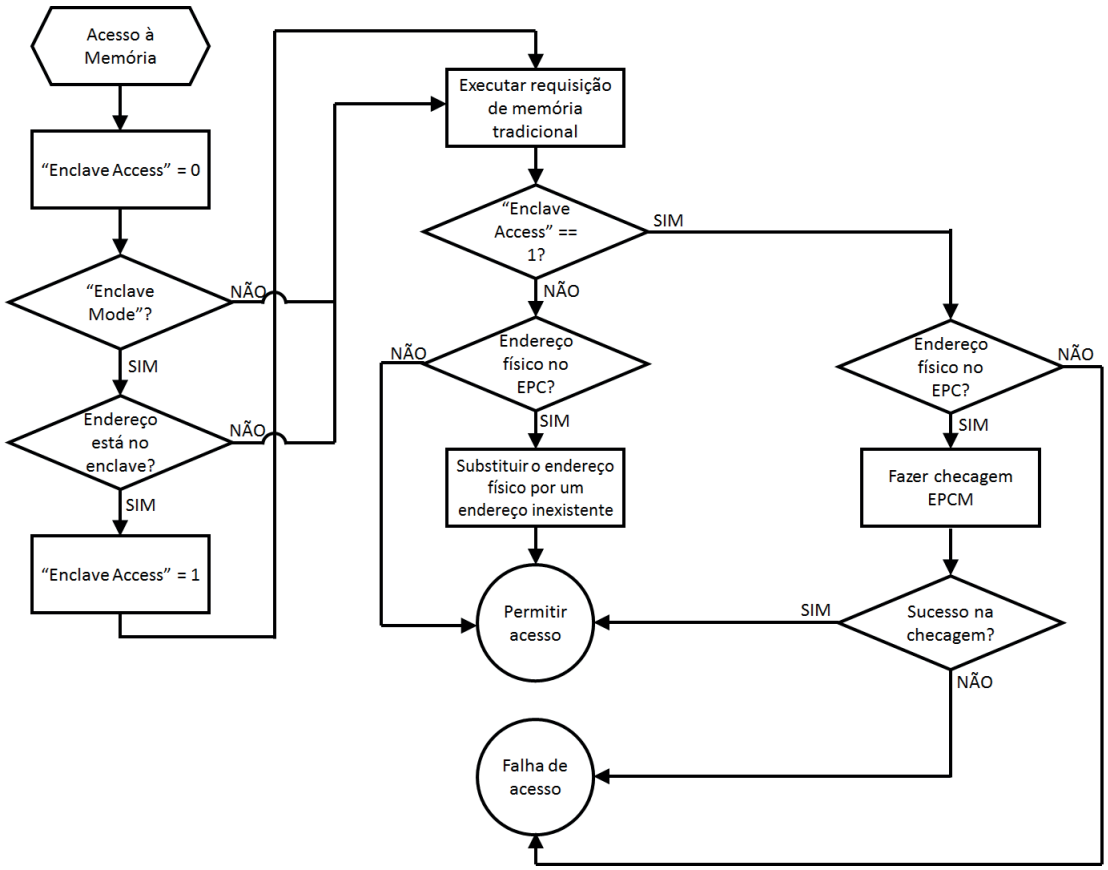


Figura 2.4. Fluxograma de acesso à memória (adaptado de [McKeen et al. 2013]).

Para realizar suas verificações de segurança, o SGX grava algumas informações sobre as decisões de alocação para cada página EPC no EPCM *(Enclave Page Cache Map)*, que é uma matriz com uma entrada para cada página EPC. Ela permite que a computação do endereço da entrada EPCM de uma página seja efetuada com apenas uma operação de troca *bit* a *bit* e uma adição. O conteúdo do EPCM é utilizado apenas pelas verificações de segurança da SGX; em condições normais de funcionamento, o EPCM não gera qualquer comportamento visível ao *software*, sendo que a sua utilização é transparente para os desenvolvedores. A organização do PRM é mostrada na Figura 2.5.

De acordo com [Costan and Devadas 2016], o EPCM basicamente armazena três informações para identificar o proprietário de cada página EPC:

- Um campo de um único *bit* (*VALID*) indica se a página EPC foi alocada (*bit* 1)

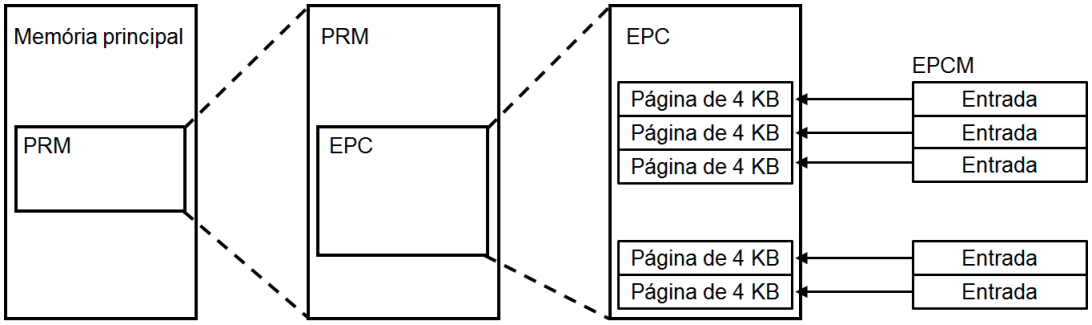


Figura 2.5. Os dados do enclave são armazenados no EPC, que é um subconjunto do PRM, que por sua vez é uma área contígua de memória que não pode ser acessada por outros softwares ou por dispositivos via DMA (adaptado de [Costan and Devadas 2016]).

ou está livre (*bit* 0), o que evita que dados sejam sobrescritos em páginas EPC já alocadas.

- Um segundo campo (*ENCLAVESECS*) identifica a qual enclave a página EPC pertence, de forma a impedir que um enclave acesse informações confidenciais de outro enclave. Isso impede que os enclaves se comuniquem via memória compartilhada utilizando páginas EPC, já que cada página EPC irá pertencer a um único enclave. Por outro lado, existe a possibilidade de os enclaves compartilharem dados residentes em uma memória não confiável, que está em uma região fora do EPC.

- Por fim, a instrução utilizada para alocar uma página EPC também determina o seu uso pretendido, que é registrado no campo de tipo de página (*PT*) da entrada EPCM correspondente. As páginas que armazenam o código e dados de um enclave são consideradas como tendo um tipo regular (*PT_REG*). As páginas dedicadas ao armazenamento das estruturas de dados de suporte da arquitetura SGX são marcadas com tipos especiais.

Um exemplo de página EPC com um tipo especial é o *SGX Enclave Control Structure* (SECS), marcada com o tipo PT_SECS, que armazena metadados do enclave. As páginas PT_SECS não são mapeadas no espaço de endereçamento dos enclaves, e são utilizadas exclusivamente pela CPU.

O SECS pode ser considerado como um sinônimo da identidade de um enclave, já que o primeiro passo para se criar um enclave é alocar uma página no EPC para armazenar seu SECS, e a última etapa para a destruição de um enclave é desalocar a página no EPC que contém o seu SECS. A entrada no EPCM que identifica um enclave possui uma página EPC que aponta para o SECS do enclave, sendo que o endereço virtual do SECS é utilizado para identificar o enclave quando este invoca uma instrução SGX.

Como as instruções do SGX utilizam endereços SECS para identificar os enclaves, o *software* não confiável deve criar entradas na sua tabela de páginas apontando para os SECS dos enclaves que gerencia. Apesar disso, o *software* não confiável não pode acessar as páginas SECS, já que estas são armazenadas no PRM, e tampouco os enclaves podem acessá-las, visto que o SGX impede explicitamente que o código do enclave acesse as

páginas SECS. Esta limitação de acesso às páginas SECS existe para que a arquitetura SGX possa armazenar informações confidenciais no SECS e possa assumir que nenhum *software* potencialmente mal-intencionado irá acessar essas informações. Um exemplo de informação confidencial armazenada no SECS é a medição do enclave (descrita na Seção 2.3.3): se o *software* fosse capaz de alterar a medição do enclave, o esquema de atestação da arquitetura SGX (descrito na Seção 2.3.6) não proporcionaria garantias de segurança.

2.3.2. Ciclo de Vida do Enclave

O processo de criação de um enclave consiste em várias etapas: inicialização da estrutura de controle do enclave; alocação das páginas de memória no EPC (*Enclave Page Cache*) e carregamento do conteúdo do enclave para essas páginas; medição do conteúdo do enclave; e criação de um identificador para o enclave. Antes de iniciar a criação do enclave, o processo já estará residindo na memória principal, estando livre para qualquer inspeção e análise e, após ele ser carregado para dentro do enclave, seus dados e código estarão protegidos de quaisquer acessos externos. O ciclo de vida do enclave é apresentado na Figura 2.6, que mostra também as instruções responsáveis pelo gerenciamento do enclave.

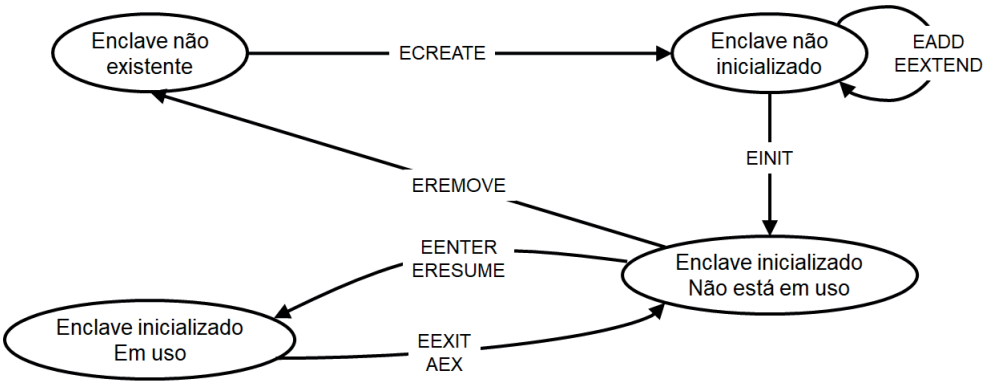


Figura 2.6. Instruções de gerenciamento do ciclo de vida do enclave e diagrama de transição de estados (adaptado de [Costan and Devadas 2016]).

A instrução *ECREATE* inicia a criação do enclave e inicializa o *SGX Enclave Control Structure* (SECS), que irá conter informações globais sobre o enclave. Cada página de memória é adicionada ao enclave utilizando a instrução *EADD* e a instrução *EEXTEND* adiciona a medição criptográfica do conteúdo do enclave. Finalmente, a instrução *EINIT* completa o processo de criação do enclave e cria a sua identidade, permitindo então que ele seja utilizado.

Após esse processo, uma questão importante é o controle da transferência de execução ao entrar e sair do enclave. No procedimento de entrada no enclave, é necessário limpar quaisquer valores em cache que possam se sobrepor à região protegida pelo enclave, além de checar todos os endereços de memória protegidos pelo enclave, identificar a instrução dentro do enclave para a qual o processador deve transferir a execução e habilitar o modo de execução em enclave. Já ao sair de um enclave, novamente deve-se limpar os valores em cache que se referem a endereços de memória protegidos pelo enclave, de forma a evitar que outro *software* possa acessar essas informações. Para entradas e saídas efetuadas programaticamente no enclave, o SGX oferece as instruções *EENTER* e

EEXIT, respectivamente. Se a saída do enclave ocorrer devido a algum evento ou falha, o processador executará uma rotina chamada *Asynchronous Exit* (AEX), que irá salvar o estado do enclave, limpar os registradores e armazenar o endereço da instrução que gerou a falha, permitindo que a execução seja posteriormente retomada, invocando a instrução *ERESUME*.

Por fim, o enclave é destruído através da instrução *EREMOVE*, que irá liberar todas as páginas EPC utilizadas pelo enclave, garantindo que nenhum processador lógico está executando instruções dentro das páginas EPC a serem removidas. O enclave é completamente destruído quando a página EPC que contém a sua estrutura *SECS* é liberada [McKeen et al. 2013, Intel 2014, Costan and Devadas 2016].

2.3.3. Medição e Assinatura do Enclave

O Intel *Software Guard Extensions Developer Guide* [Intel 2016a] define três principais atividades para estabelecer a confiança em um enclave:

- **Medição:** Um enclave é instanciado em um ambiente confiável e é feita uma gravação precisa e protegida de sua identidade.
- **Atestação:** Demonstra a outras entidades que o enclave foi instanciado de maneira correta.
- **Selagem:** Permite que os dados pertencentes a um ambiente confiável sejam vinculados a ele de tal forma que só podem ser restaurados quando o ambiente confiável é restaurado.

Nesta seção, será abordada a medição do enclave, sendo que a selagem e a atestação serão tratados nas seções seguintes.

Cada enclave possui um certificado auto-assinado pelo autor do enclave, também conhecido como *Enclave Signature* (SIGSTRUCT). A assinatura do enclave contém informações que permitem que a arquitetura Intel SGX detecte se alguma parte do enclave foi adulterada, de forma a permitir que um enclave possa provar que ele foi corretamente carregado no EPC e pode ser confiável. No entanto, o *hardware* apenas verifica a medição do enclave quando este é carregado. Assim, qualquer pessoa pode modificar um enclave e assiná-lo com sua própria chave. Para evitar esse tipo de ataque, a assinatura do enclave também identifica o autor do enclave, contendo vários campos essenciais para que o enclave seja atestado por entidades externas:

- ***Enclave Measurement*:** Um *hash* de 256 *bits* único, que identifica o código e os dados iniciais a serem colocados dentro do enclave, bem como a ordem e a posição na qual eles devem ser colocados, e as propriedades de segurança dessas páginas. Uma alteração em qualquer uma dessas variáveis resultará em uma medida diferente. Quando as páginas de código/dados do enclave são carregadas para o EPC, a CPU efetua a medição do enclave e armazena esse valor no registrador MRENCLAVE. Em seguida, a CPU compara o conteúdo do MRENCLAVE com o valor contido no SIGSTRUCT desse enclave e, somente se eles forem idênticos, a CPU permitirá que o enclave seja inicializado.

- **Chave Pública do Autor do Enclave:** Após um enclave ser inicializado com êxito, a CPU registra um *hash* da chave pública do autor do enclave no registrador MRSIGNER. O conteúdo do MRSIGNER servirá como identidade do autor do enclave. Assim, os enclaves que foram autenticados com a mesma chave devem ter o mesmo valor colocado no registo MRSIGNER.

- ***Security Version Number* do Enclave (ISVSVN):** O autor do enclave atribui um SVN (*Security Version Number*) a cada versão de um enclave. O SVN reflete o nível da propriedade de segurança do enclave, e deve monotonicamente aumentar com melhorias das propriedades de segurança. Depois que um enclave é inicializado com sucesso, a CPU registra o SVN, que pode ser usado durante a atestação. Diferentes versões de um enclave com as mesmas propriedades de segurança devem ter o mesmo SVN. Por exemplo, uma nova versão de um enclave com correções de *bugs* não relacionados à segurança deve ter o mesmo SVN que a versão anterior.

- ***Product ID* do Enclave (ISVPRODID):** Permite que o autor marque seus enclaves com identificadores de produto com a mesma identidade do autor. Depois que um enclave é inicializado com êxito, o *Product ID* é registrado pela CPU, que pode ser usado durante a atestação.

O desenvolvedor deve fornecer o SVN e o *Product ID* de um enclave, bem como um par de chaves para gerar a assinatura de enclave. A CPU deriva a identidade do autor do enclave a partir de sua chave pública, já a chave privada é utilizada para assinar o enclave. O cálculo da medida do enclave deve ser realizado com base no código e nos dados iniciais a serem colocados dentro do enclave, a ordem esperada e a posição em que eles devem ser colocados e as propriedades de segurança dessas páginas. O código e os dados iniciais a serem colocados no interior do enclave, bem como as propriedades de segurança dessas páginas são gerados pelo compilador, enquanto a sua colocação no enclave é controlada pelo carregador de enclave. Assim, o cálculo de medição deve seguir o comportamento esperado do carregador de enclave no que diz respeito à forma de colocar o código e os dados iniciais dentro do enclave.

A chave de assinatura do desenvolvedor é parte da identidade do enclave e é fundamental para proteger seus segredos. Um intruso que comprometa a chave de assinatura privada de um ISV (*Independent Software Vendor*) poderá ser capaz de escrever um enclave malicioso que ateste com êxito a identidade dos enclaves legítimos e/ou escrever um *malware* que usa o enclave malicioso para comprometer dados selados em plataformas individuais [Intel 2016a].

2.3.4. Interface do Enclave: Chamadas ECALL e OCALL

A arquitetura SGX requer que toda a funcionalidade dentro de um enclave seja ligada estaticamente, em tempo de compilação. Isso cria um *trade-off* de desempenho/tamanho que os desenvolvedores devem analisar cuidadosamente, pois impacta no tamanho do TCB. Ao usar a funcionalidade de biblioteca estática, os desenvolvedores têm duas opções: fornecer uma camada para executar funções fora do enclave, adicionando uma sobrecarga de desempenho na aplicação; ou incluir a implementação da biblioteca como parte do enclave, provocando um aumento no tamanho do TCB. A utilização da primeira opção é

representada pela Figura 2.7, onde tem-se a divisão entre os componentes confiável e não confiável da aplicação, e ambos tem uma camada chamada *edge routines*, ou rotinas de borda, que são funções que podem ser executadas dentro ou fora de um enclave, com a função de ligar uma chamada do aplicativo com uma função dentro do enclave ou uma chamada do enclave com uma função no aplicativo.

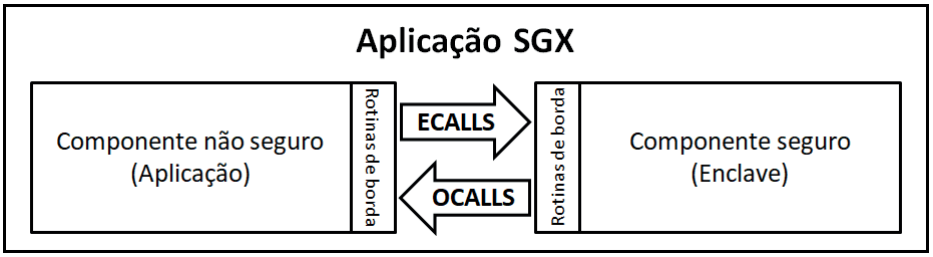


Figura 2.7. Divisão da aplicação em dois componentes, confiável e não confiável, e a comunicação entre os dois componentes (adaptado de [Intel 2016a]).

Particionar um aplicativo em dois componentes, confiável e não confiável, tem implicações adicionais do ponto de vista da segurança. Pode-se dizer que uma porção de código e dados menor normalmente implica uma menor probabilidade de ter defeitos no produto final, e também resulta em uma análise de segurança mais simples e um *software* mais seguro. Outro ponto a ser considerado é que, quanto menor for o tamanho do enclave, mais rápido este será capaz de efetuar o *backup* dos dados fora do enclave quando ocorrer um evento de hibernação do sistema, além de também carregar o enclave mais rapidamente quando o sistema é restaurado. Assim, embora seja possível mover a maior parte do código do aplicativo para um enclave, na maioria dos casos isso não seria desejável, visto que o tamanho de TCB deve ser um fator a considerar ao projetar o que será colocado dentro de um enclave, com o desenvolvedor tendo o objetivo de minimizar o tamanho deste [Intel 2016a].

2.3.4.1. Chamadas para o Enclave (ECALLs)

Após definir os componentes confiável (enclave) e não confiável (aplicativo) de uma aplicação habilitada para a arquitetura SGX, o desenvolvedor deve definir cuidadosamente a interface entre esses componentes. O código confiável é executado nos seguintes cenários:

- Quando o componente não confiável faz uma chamada explícita para uma função pertencente ao enclave, através de um ECALL. Este procedimento é o mesmo que um aplicativo regular chamar uma função em uma biblioteca compartilhada.

- Após o retorno de uma chamada feita a partir do enclave para o aplicativo externo (OCALL). O retorno de um OCALL é semelhante ao que acontece quando uma biblioteca compartilhada envia o retorno de uma chamada efetuada por outra biblioteca compartilhada, por exemplo, o retorno de uma biblioteca *Standard C* para executar uma operação de E/S. Quando um OCALL retorna, a função confiável que fez a chamada para fora do enclave continua a sua execução.

- A arquitetura SGX suporta interrupções durante a execução do enclave, no entanto, garante que, ao retornar a para dentro do enclave, sua execução continuará como se a interrupção nunca tivesse ocorrido, da mesma forma que deve ocorrer quando ocorre uma interrupção durante a execução de uma função em uma biblioteca regular.

Um enclave deve disponibilizar um conjunto de métodos e/ou funções que estarão disponíveis para que o aplicativo faça uso (ECALLs) e descrever quais serviços o aplicativo deve fornecer (OCALLs), sendo que essas funções devem ser definidas pelo desenvolvedor.

As funções ECALLs expõem a interface do enclave que o aplicativo não confiável pode utilizar e, por isso, deve-se limitar o número de ECALLs para reduzir a superfície de ataque ao enclave. O desenvolvedor deve estar ciente de que o enclave não tem controle sobre quais ECALLs são executadas ou em que ordem estas são chamadas. Desta forma, um enclave não deve depender de ECALLs sendo executadas em uma determinada ordem.

Em contrapartida, as funções ECALL e OCALL somente podem ser chamadas após a inicialização do enclave, significando que as alocações de endereço necessárias foram executadas com êxito, todos os dados globais (incluídos os relativos à segurança) foram inicializados com êxito, o *trusted thread context* foi inicializado com todos os seus dados de segurança, e as funções de inicialização foram executadas de forma completa.

As entradas e saídas de dados no enclave podem ser observadas e modificadas por aplicações ou códigos executados em um ambiente não confiável. Desta forma, o desenvolvedor do enclave nunca deve confiar em qualquer dado proveniente de um domínio de *software* não confiável, e deve sempre verificar os parâmetros de entrada das funções ECALLs e os valores retornados pelas funções OCALLs. Após os dados terem sido verificados e ter sido identificada a sua origem e/ou destino (entidade remota, usuários, etc.), deve-se decidir pela aplicação ou não de criptografia para proteger os dados que em algum momento estarão expostos ao domínio de *software* não confiável.

Deve-se ter também um cuidado especial para tratar os ponteiros e as passagens de parâmetros por referência a funções ECALL, já que o aplicativo não confiável pode passar um ponteiro referenciando um endereço de memória dentro do limite do enclave, o que pode fazer com que o enclave sobrescreva seus dados ou código de forma indevida, podendo até mesmo ocasionar o vazamento de informações confidenciais do enclave.

2.3.4.2. Chamadas Externas ao Enclave (OCALLs)

Os enclaves não podem acessar diretamente os serviços fornecidos pelo sistema operacional (*system calls*). Para isso, um enclave deve executar uma chamada OCALL para uma rotina no aplicativo não confiável, o que acrescenta uma sobrecarga de desempenho, mas não afeta a confidencialidade dos dados. Porém, a comunicação com o sistema operacional pode requerer a divulgação de dados ou a importação de dados vindos do aplicativo não confiável que precisam ser tratados adequadamente. Além disso, as chamadas OCALLs, necessárias para operações como sincronização de *threads* e operações de E/S, são expostas ao domínio não confiável da aplicação e, portanto, podem ser associadas a alguns riscos de segurança. Desta forma, um enclave deve ser projetado de tal forma que evite um vazamento de

informações que permitiriam a um invasor, que está examinando as funções não confiáveis que são chamadas pelo enclave, obter informações sobre os dados confidenciais do mesmo.

O enclave também deve tratar as situações em que uma chamada de função OCALL não é executada por completo, para evitar situações em que, por exemplo, um invasor intercepte uma solicitação do enclave para gravar dados em disco e emita uma resposta informando que a operação foi concluída com êxito, quando na verdade a operação não foi realizada. O retorno de uma chamada OCALL ocorre no mesmo contexto que estava sendo executado antes da função OCALL ser chamada, mantendo, assim, o fluxo de execução do enclave.

Ao executar uma chamada OCALL, o domínio não confiável da aplicação também pode efetuar chamadas ECALL, retornando a execução para o enclave durante a execução dessa chamada. Uma vez fora do enclave, um intruso pode tentar encontrar vulnerabilidades invocando funções ECALL recursivamente para o enclave, assim, o desenvolvedor deve limitar quais chamadas ECALL são permitidas durante a execução de um OCALL, ou até mesmo impedindo que as chamadas ECALL ocorram.

Uma chamada OCALL expõe somente os dados passados a ela por parâmetro. O valor de retorno da função e os parâmetros de retorno passados a ela por referência pertencem ao domínio confiável da aplicação e, por isso, não estão acessíveis fora desse domínio. Se o valor de retorno de uma função OCALL é um ponteiro, apenas a referência deste estará dentro do enclave; os valores referenciados por este ponteiro devem ser previamente verificados para serem utilizados no domínio confiável da aplicação.

2.3.5. Selagem de Dados

Quando um enclave é instanciado, o desenvolvedor deve identificar quais dados e/ou estados devem ser protegidos e que devam ser preservados (“selados”) após o enclave ser destruído. O enclave pode ser encerrado quando a aplicação completa a sua execução e encerra o enclave, quando a aplicação é encerrada (de maneira forçada ou não), e quando o computador entra em hibernação ou é desligado.

Em geral, os dados fornecidos a um enclave são perdidos quando o enclave é destruído mas, se estes dados precisam ser preservados durante um desses eventos para uso futuro, estes devem ser armazenados fora dos limites do enclave antes de destruí-lo. Para proteger e preservar os dados, existe um mecanismo que permite ao enclave recuperar uma chave exclusiva para esse enclave. Essa chave só pode ser gerada por esse enclave naquela plataforma específica. O enclave usa essa chave para cifrar dados para a plataforma ou para decifrar dados já existentes na plataforma. Estas operações de cifragem e decifragem são chamadas de *sealing* e *unsealing*, respectivamente[Intel 2016a].

Ao selar dados, o enclave precisa especificar as condições que precisam ser atendidas quando os dados devem ser recuperados e, para isto, existem duas opções disponíveis. A primeira opção envolve a criação de um selo para o enclave corrente, utilizando a medida do enclave (MRENCLAVE) para a criação do selo. Assim, somente um enclave com a mesma medida será capaz de acessar os dados que forem selados desta maneira. Isso significa que, se o código do enclave sofrer alterações, isso refletirá em uma mudança na medição do enclave e a chave de criptografia irá ser alterada também, impedindo a

recuperação dos dados previamente selados.

A segunda opção é utilizar a assinatura do autor do enclave (MRSIGNER), juntamente com o *Product ID* para gerar o selo do enclave. Assim, somente um enclave com o mesmo valor no registrador MRSIGNER e o mesmo *Product ID* será capaz de acessar os dados que foram selados dessa maneira. Desta forma, permite-se que um enclave seja atualizado pelo autor, mas não requer um processo complexo de atualização para desbloquear dados selados na versão anterior do enclave (que terá uma valor de MRENCLAVE diferente) e também permite que enclaves do mesmo autor compartilhem dados selados.

Os desenvolvedores também podem definir o *Security Version Number* (SVN) quando codificam o enclave, que também é armazenado na CPU quando o enclave é instanciado. Um enclave deve fornecer o SVN em sua solicitação para obter a chave de criptografia da CPU, não sendo possível especificar um SVN superior ao especificado na assinatura do enclave (ISVSVN). No entanto, o enclave pode especificar um SVN anterior ao ISVSVN do enclave, dando-lhe a capacidade de acessar dados selados por uma versão anterior do enclave, o que facilitaria as atualizações de *software*, por exemplo.

Além disso, as aplicações que executarão em enclaves não devem ser distribuídas com os dados confidenciais, mas sim, após a instalação, contatar um provedor de serviços para solicitar esses dados. Este processo é descrito na Figura 2.8 [Anati et al. 2013].

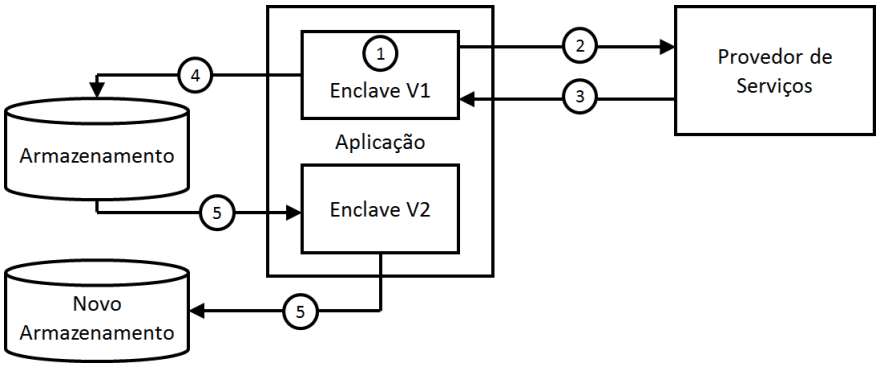


Figura 2.8. Fluxograma referente à integridade no armazenamento de dados (adaptado de [Anati et al. 2013]).

No passo 1, o *software* inicia a criação do enclave para que possa ser executado de forma segura. No passo 2, o enclave contata o provedor de serviços para efetuar o *download* dos dados; nesta etapa, o provedor irá gerar uma chave que identificará a máquina e o enclave que está executando a aplicação. Posteriormente, no passo 3, o provedor estabelece uma linha de comunicação segura com a aplicação e envia os dados para o enclave. Já no passo 4, o enclave utiliza uma chave de criptografia baseada em *hardware* para codificar e armazenar os dados confidenciais em um sistema de armazenamento permanente, garantindo que os dados serão acessados somente quando o enclave for restaurado. No passo 5 tem-se a situação de uma atualização de *software*, sendo que neste caso o enclave irá acessar os dados e gerar uma nova chave de criptografia, de forma que somente a nova versão do *software* consiga acessar os dados novamente, impedindo que versões anteriores da aplicação tenham acesso a esses dados.

Todo esse processo tem como finalidade proteger os dados confidenciais até mesmo

de versões anteriores da aplicação que possam conter brechas que possibilitem o acesso indevido aos mesmos. Desta forma, quando essas brechas forem detectadas, a empresa responsável pelo desenvolvimento do *software* pode corrigi-las e disponibilizar uma atualização, podendo impor que apenas os usuários que disponham da última versão do *software* possam acessar os dados e o provedor [Intel 2014].

2.3.6. Atestação

Atestação é o processo de comprovar que um *software* foi corretamente estabelecido em uma plataforma. No caso da Intel SGX, é o mecanismo pelo qual uma terceira entidade estabelece que uma determinada entidade de *software* está em execução em uma plataforma habilitada para Intel SGX e protegida dentro de um enclave, antes de provisionar esse *software* com segredos e dados protegidos. A atestação depende da capacidade de uma plataforma produzir uma credencial que reflita com precisão a assinatura de um enclave, que inclui informações sobre as propriedades de segurança do enclave. A arquitetura Intel SGX fornece os mecanismos para suportar duas formas de atestação. Existe um mecanismo para criar uma afirmação básica entre enclaves em execução na mesma plataforma, que suporta atestação local ou intra-plataforma, e outro mecanismo que fornece a base para a atestação remota, com os enclaves executando em dispositivos diferentes.

A atestação intra-plataforma é importante, pois os desenvolvedores podem escrever enclaves que podem cooperar uns com os outros para executar alguma função de nível mais alto. Para isso, os desenvolvedores precisam de um mecanismo que permita a um enclave provar sua identidade e autenticidade para outra parte dentro da plataforma local.

Um enclave pode pedir ao *hardware* para gerar uma credencial, também conhecida como relatório ou *EREPORT*, que inclui uma prova criptográfica de que o enclave existe na plataforma. Esse relatório pode ser fornecido a outro enclave para verificar se este foi gerado na mesma plataforma. O mecanismo de atestação utilizado para o certificado de enclave intra-plataforma utiliza um sistema de chaves simétricas, onde apenas o enclave que verifica a estrutura de relatório e o *hardware* de enclave que cria o relatório conhecem a chave. O processo de atestação é efetuado em três etapas, como mostra a Figura 2.9 [Anati et al. 2013].

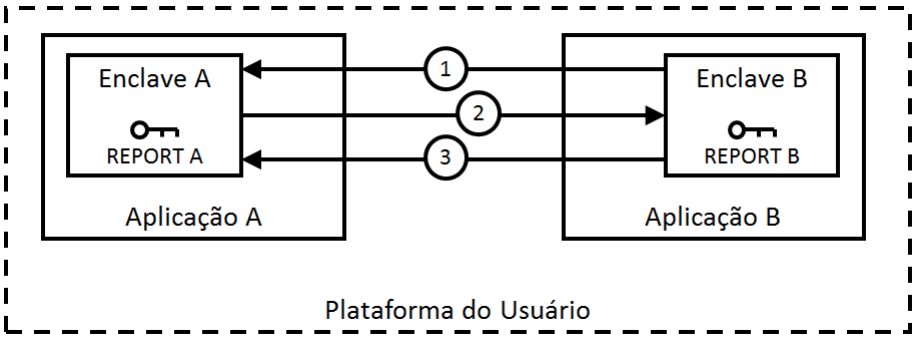


Figura 2.9. Autenticação intra-plataforma (adaptado de [Anati et al. 2013]).

No primeiro passo, o enclave A obtém a identificação do enclave B, utilizando uma comunicação aberta. No passo 2 o enclave A cria a estrutura *REPORT* utilizando a identificação do enclave B e envia esta estrutura ao enclave B, ainda utilizando uma

comunicação aberta. No terceiro passo, o enclave B utiliza os dados contidos no *REPORT* enviado por A de forma a verificar que o enclave A está sendo executado no mesmo dispositivo que B e, confirmando estes dados, o enclave B também pode criar uma estrutura *REPORT* e enviar esta ao enclave A. Assim, os dois enclaves estão autenticados e podem trocar mensagens de maneira segura.

Já a autenticação inter-plataforma, ou remota, requer o uso de criptografia assimétrica e de um enclave especial, chamado de *Quoting Enclave*, que irá verificar as estruturas *REPORT* dos outros enclaves da máquina utilizando a autenticação intra-plataforma e então substituir o *message authentication code* (MAC) dessas estruturas com uma assinatura criada através de uma chave assimétrica, utilizando o *Intel Enhanced Privacy ID* (EPID) e tendo como saída uma outra estrutura chamada *QUOTE*. O processo de autenticação remota é descrito na Figura 2.10 [Anati et al. 2013].

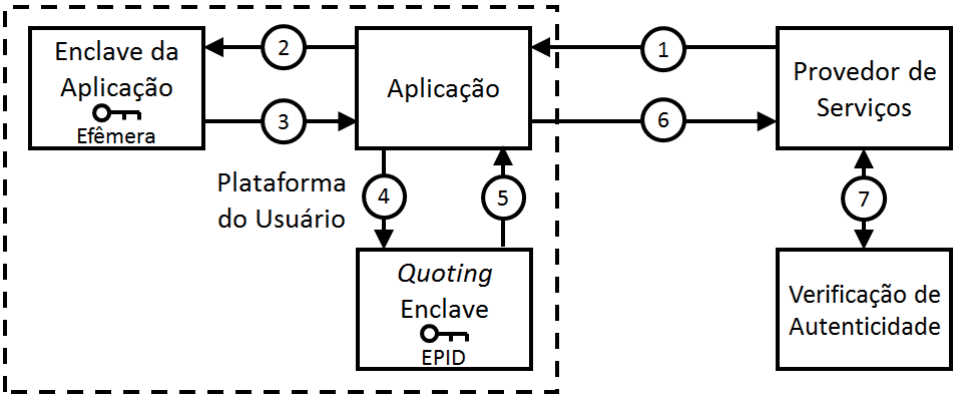


Figura 2.10. Autenticação remota (adaptado de [Anati et al. 2013]).

Primeiramente, o provedor de serviços solicita que a aplicação prove que está executando os componentes necessários dentro de um ou mais enclaves. No segundo passo, a aplicação encaminha a solicitação do provedor de serviços, juntamente com a identidade do *Quoting Enclave*, para o enclave da aplicação. No passo 3, o enclave da aplicação gera uma chave pública efêmera que deverá ser utilizada pelo provedor de serviços para as próximas comunicações e então gera a estrutura *REPORT*, enviando esta para a aplicação. No passo 4, a aplicação encaminha a estrutura *REPORT* para que o *Quoting Enclave* a assine. No passo 5 o *Quoting Enclave* verifica a estrutura *REPORT*, cria a estrutura *QUOTE*, assinando-a com a sua chave EPID, e a devolve à aplicação. No passo seguinte (6), a aplicação envia a estrutura *QUOTE* para o provedor de serviços. Finalmente, o provedor de serviços pode utilizar um verificador de autenticação para analisar a resposta recebida (7) e verificar se a aplicação satisfaz os requisitos.

A arquitetura Intel SGX também provê um enclave especial conhecido como o *Quoting Enclave*, que verifica os relatórios que foram criados para o MRENCLAVE e, em seguida, converte e assina esses relatórios usando uma chave assimétrica específica do dispositivo, a chave *Intel EPID*. A saída desse processo é chamada uma *quote*, e pode ser verificada fora da plataforma. Somente o *Quoting Enclave* instanciado tem acesso à chave Intel EPID, assim a chave da CPU nunca é exposta fora da plataforma.

O Intel EPID é um esquema de assinatura de grupo, que permite que as plataformas

assinem criptograficamente objetos ao mesmo tempo em que preservam a privacidade do assinante, com cada assinante de um grupo tendo sua própria chave privada para assinatura, e os verificadores utilizando a mesma chave pública do grupo para verificar assinaturas individuais, impedindo que os usuários possam ser identificados de forma exclusiva [Intel 2016a].

Uma entidade remota pode fornecer dados confidenciais para um enclave após a atestação remota ter sido realizada, com a transferência dos dados sendo conduzida através de um canal seguro. O estabelecimento de canal seguro deve estar vinculado ao processo de atestação remota, para evitar que o servidor remoto forneça dados confidenciais para uma entidade diferente do enclave que foi atestado.

Existem diversas maneiras de estabelecer um canal confiável para a comunicação remota entre em enclave e um provedor de serviços. Uma dessas maneiras é o enclave autenticar o servidor para garantir que receberá os dados de uma entidade confiável e, após isso, o enclave pode gerar um par de chaves público/privada para aquela comunicação, enviando a chave pública ao servidor. Após o servidor validar o enclave que enviou a chave, o próprio servidor pode gerar uma chave de criptografia simétrica E, cifrá-la com a chave pública P e enviar $P(E)$ através do canal de comunicação para a aplicação remota. Após receber os dados, o enclave pode decifrar $P(E)$ utilizando a sua chave privada e, assim, tanto o enclave quanto o servidor remoto irão possuir a mesma chave de criptografia E. O canal de comunicação não precisa ser protegido, já que os dados que trafegarão neste canal estarão cifrados.

O servidor remoto também deve verificar a identidade do assinante do enclave (MRSIGNER) antes de encaminhar a este quaisquer dados confidenciais, já que na instanciação do enclave apenas é verificado o registrador MRENCLAVE. Dessa forma, qualquer pessoa poderia modificar um enclave e assiná-lo novamente. Além disso, o provedor de serviços também deve verificar os outros atributos do enclave, para evitar que sejam divulgados dados confidenciais a enclaves de teste ou que estão em modo de depuração, por exemplo.

Após estabelecer um canal seguro para a comunicação, os dados confidenciais podem ser provisionados para o enclave, cifrando os dados seguros S com uma chave E e enviando, assim, $E(S)$ para o enclave, que utilizará a mesma chave E para efetuar a decifragem de $E(S)$, obtendo novamente os dados S. No entanto, pode ser inconveniente exigir que o enclave se conecte a uma entidade remota toda vez que ele necessitar os mesmos dados e, para evitar este tipo de situação, o enclave pode armazenar os dados obtidos em uma memória não volátil, selando estes dados (conforme descrito na Seção 2.3.5) de forma que ficarão acessíveis somente ao enclave que os selou, e somente na plataforma em que eles foram selados [Intel 2016a].

2.4. Casos de Uso da Arquitetura SGX

Apesar da arquitetura SGX ser relativamente nova, já que foi de fato lançada no mercado em outubro de 2015, juntamente com os processadores Intel Core de 6^a geração (*Skylake*), desde o seu anúncio vários trabalhos foram sendo desenvolvidos de forma a validar a proposta. Um desses trabalhos, descrito por [Jain et al. 2016], apresenta a proposta de emulação das instruções adicionadas à SGX, utilizando um emulador de processador *open-source*, o QEMU. O emulador desenvolvido não se restringe apenas à simulação de

hardware, mas também de componentes do sistema operacional, além de fornecer uma biblioteca que permite a depuração e o monitoramento do desempenho das aplicações.

Em [Hoekstra et al. 2013] são apresentados três protótipos de aplicações que se valem dos recursos fornecidos pela SGX. O primeiro protótipo é de uma aplicação do tipo OTP (*One-Time Password*), que tem por finalidade prover uma chave de acesso que poderá ser utilizada apenas uma vez, por exemplo em transações financeiras. O segundo protótipo apresenta um ERM (*Enterprise Rights Management*), que tem como finalidade manter a integridade e controlar o acesso a documentos sigilosos, sendo este protótipo desenvolvido para o Departamento de Segurança Interna dos EUA. O terceiro protótipo desenvolvido se refere a um sistema de videoconferência, que faz uso da SGX e do PAVP (*Protected Audio and Video Path*), de forma a manter o sigilo dos dados de áudio e vídeo em trânsito.

Outros trabalhos também fazem uso da arquitetura SGX para prover segurança a aplicações que são executadas em ambientes de nuvem pública. Um exemplo é apresentado em [Baumann et al. 2015], que descreve um protótipo que permite encapsular aplicações legadas dentro de um enclave, sem a necessidade de reescrevê-las. Esta abordagem pode ser utilizada tanto para aplicações simples, como também para máquinas virtuais executadas em ambientes *IaaS*. [Brenner et al. 2017] também apresenta o uso do SGX para encapsular a execução de micro serviços em ambientes na nuvem. Outros trabalhos que fazem uso da arquitetura SGX em ambientes distribuídos são descritos em [Schuster et al. 2015] e [Shih et al. 2016].

[Richter et al. 2016] utilizam a arquitetura SGX para isolar módulos do *kernel* do Linux em enclaves, evitando que uma falha em um módulo do *kernel* se propague ao restante do sistema. Para isso, partes do módulo são migradas para o *user space*, visto que os enclaves não podem ser executados em modo núcleo. Já [Tsai et al. 2017] e [Tian et al. 2017] apresentam um sistema operacional, constituído por uma biblioteca (ou seja, um *Library OS*), que é executado dentro de um enclave.

Os trabalhos [Brekalo et al. 2016] e [Mofrad et al. 2017] propõem o uso do SGX para auxiliar na segurança de bancos de senhas e na criação de uma biblioteca de criptografia, respectivamente. O ponto em comum nos dois trabalhos é a utilização de enclaves para a geração e manutenção das chaves de criptografia a serem utilizadas. Por sua vez, [Karande et al. 2017] faz uso do SGX para manter a integridade e confidencialidade dos dados em *logs* gerados pelo sistema Linux.

Por fim, [Lind et al. 2017] apresentam um *framework* que utiliza diretivas de pré-compilação para particionar automaticamente aplicações escritas em linguagem C em seus componentes confiável e não confiável.

2.5. Limitações, Questões em Aberto e Vulnerabilidades do SGX

Apesar da arquitetura SGX prover mecanismos eficientes para garantir a segurança dos dados de uma aplicação, ainda há algumas questões a serem consideradas com maior atenção.

Uma das limitações da arquitetura SGX diz respeito ao uso da CPU para determinar a chave utilizada para a selagem dos dados. Isso implica que, no caso de substituição da CPU, seja por opção do usuário ou por problemas técnicos, o enclave não estará mais apto

a abrir os dados previamente selados por ele. Essa é uma questão também importante quando se trata de questões de balanceamento de carga em servidores, seja na utilização de múltiplos servidores ou de servidores com múltiplos processadores. Em virtude desta limitação, ainda não há disponíveis no mercado processadores multi-*socket* com arquitetura SGX. O Intel *Software Guard Extensions Developer Guide* [Intel 2016a] também coloca que, na necessidade de efetuar a migração de dados selados para outro computador, isso deverá ser efetuado utilizando o processo de atestação remota entre ambos os enclaves.

Outro ponto, descrito por [Hoekstra et al. 2013], diz respeito à entrada de dados das aplicações. Os usuários, inevitavelmente, terão que informar dados em algum momento para as aplicações que estão protegidas em seus enclaves, mas os dispositivos de entrada de dados (como teclados, câmeras de vídeo, microfones, entre outros) não estão preparados para fornecer esses dados ao sistema de forma segura, o que possibilita sua interceptação antes que eles cheguem à segurança do enclave. Vale ressaltar também que, assim como os dispositivos de entrada, tem-se também os dispositivos de saída que, da mesma forma, não estão preparados para fornecer a segurança adequada no tratamento dos dados provenientes dos enclaves.

[Davenport and Ford 2014] também apontam uma preocupação quanto ao uso do SGX como um aliado à execução de *malwares*, visto que a Intel não fez nenhuma restrição a quais aplicações poderiam se utilizar da segurança dos enclaves. [Schwarz et al. 2017] implementaram um ataque de canal lateral ao enclave através de um *malware* que, por executar dentro de outro enclave, fica protegido pelo isolamento provido pela arquitetura SGX e não pode ser identificado por anti-virus ou analisado pelo sistema operacional. Procedimentos forenses também ficam limitados devido à proteção da memória.

O Intel SGX não considera ataques por canal lateral nem ataques de engenharia reversa em seu modelo de ameaças. O Intel *Software Guard Extensions Developer Guide* [Intel 2016a] ressalta que cabe aos desenvolvedores construir enclaves resistentes a estes tipos de ataque. O *malware* de [Schwarz et al. 2017] é um ataque de canal lateral à memória *cache* do tipo *Prime and Probe*, capaz de extrair uma chave de um processamento RSA ocorrendo dentro do enclave da vítima. Este é o tipo de ataque mais comum ao enclave SGX encontrado até o momento. [Moghimi et al. 2017] utiliza este mesmo método para extrair chaves AES de um processamento do enclave e [Brasser et al. 2017] conseguiu, além de extrair uma chave RSA, detectar sequências específicas do genoma humano durante a indexação genômica ocorrendo dentro do enclave.

Nestes ataques o enclave da vítima e o *malware* executam em paralelo na mesma máquina física. Apesar do enclave estar protegido criptograficamente na EPC, na *cache* os dados estão em claro. Além disso, é o *software* do sistema operacional que gerencia a tradução dos endereços da *cache*, que por sua vez é compartilhada com os demais *softwares* em execução no computador. Desta forma, apesar do atacante não conseguir acesso direto ao conteúdo do *cache* usado pelo enclave, ele pode aplicar a técnica conhecida como *Prime and Probe* para inferir segredos do enclave.

Para obter sucesso com a técnica *Prime and Probe*, pelo menos dois requisitos devem ser atendidos: o *malware* deve ser capaz de sobrescrever determinados endereços da *cache* usados pela vítima, e também deve ser capaz de realizar medição de tempo com resolução boa o suficiente para distinguir *hits* e *misses* na *cache*. Em linhas gerais, no

primeiro momento o atacante preenche a *cache* monitorada com seus dados. Então aguarda que a vítima execute algum código cujo acesso à memória é dependente da informação secreta (a chave por exemplo). Na sequencia o atacante faz uma requisição a dados seus e mede o tempo para recuperá-los. Este tempo determina se o dado se encontrava ou não na *cache* (*hit* ou *miss*). No caso de *miss*, sabe-se que aquele endereço foi utilizado pela vítima. Então o atacante se aproveita da dependência do acesso à memória para inferir os *bits* correspondentes ao segredo.

Em nível de *software*, no entanto, várias bibliotecas criptográficas estão sendo robustecidas especificamente para evitar ataques ao *cache*. Para cada acesso à memória dependente de segredos, o enclave pode gerar um conjunto de acessos à memória que irá se manifestar em mudanças em todos os conjuntos da *cache* monitorados, escondendo o endereço de memória efetivamente acessado do adversário. Outras abordagens buscam eliminar as dependências entre o segredo e o acesso à memória das implementações, o que também é efetivo para evitar este tipo de ataque.

[Costan and Devadas 2016] ressalta outras vulnerabilidades, como ataques por monitoramento do consumo de energia, cujas variações podem gerar informações suficientes para que o atacante infira a sequência de instruções sendo executadas, ou ainda, ataques passivos de tradução de endereçamento que podem dar informações do padrão de acesso à memória com a granularidade de páginas. A Intel menciona que o modelo de ameaças ao SGX considera que o atacante pode ter acesso à *flash* que armazena o *firmware* do computador, mas segundo [Costan and Devadas 2016], a documentação oficial não aborda as implicações decorrentes deste fato. No entanto, até o presente momento, não foram encontrados trabalhos demonstrando o uso destas vulnerabilidades para efetivamente descobrir segredos do enclave.

Por fim, além da Intel ter de ser considerada confiável, como, por exemplo, no provimento da PROVISIONKEY utilizada em enclaves que implementam a atestação, [Costan and Devadas 2016] questiona a real necessidade do *Launch Enclave* (LE), que é um enclave emitido pela Intel e responsável por aprovar todos os enclaves antes da inicialização. Segundo os autores, o LE poderia funcionar como um mecanismo de licenciamento, permitindo à Intel se posicionar como um intermediário na distribuição de todo *software* SGX.

2.6. SGX na Prática: SGX SDK

Esta seção tem o objetivo de descrever os detalhes para a construção de aplicações utilizando a arquitetura Intel SGX. Para isto, faz-se o uso do SDK (*Software Development Kit*) do SGX para o sistema operacional Linux, com implementações em linguagem C [Intel 2016b]. Os exemplos apresentados nesta seção estão disponíveis na íntegra em https://github.com/newtoncw/sbseg-sgx, já as bibliotecas necessárias para a execução dos exemplos, bem como as instruções para sua instalação, estão disponíveis em https://github.com/01org/linux-sgx.

2.6.1. A Linguagem de Definição do Enclave e a Ferramenta *Edger8r*

A definição da interface do enclave (funções ECALLs e OCALLs) é feita através da configuração dos arquivos EDL (*Enclave Definition Language*), que definem também os

tipos de dados que serão suportados pelo enclave. A Listagem 2.1 traz um exemplo de configuração de um arquivo EDL.

Listagem 2.1. Exemplo de configuração de arquivo EDL (adaptado de [Intel 2016b]).

```
1  enclave {
2      from "file1.edl" import *;
3      from "file2.edl" import foo, bar;
4
5      include "string.h"
6      include "mytypes.h"
7
8      struct mysecret {
9          int key;
10         const char* text;
11     };
12
13     enum boolean { FALSE = 0, TRUE = 1 };
14
15     trusted {
16         include "trusted.h"
17         public void set_secret([in] struct mysecret* psecret);
18         void some_private_func(enum boolean b);
19     };
20
21     untrusted {
22         include "untrusted.h"
23         void ocall_print();
24         int another_ocall([in] struct mysecret* psecret) allow(some_private_func);
25     };
26 };
```

O arquivo EDL é dividido em duas seções: a seção `trusted` define as ECALLs, enquanto a seção `unstrusted` define as OCALLs. Como descrito nas seções anteriores, uma ECALL define um ponto de entrada para o enclave, e uma OCALL define uma transferência do controle do enclave para a aplicação, para a realização de *systems calls* ou operações de E/S. As OCALLs também podem ser utilizadas para efetuar a transferência de dados entre o enclave e a aplicação, sendo que estas são opcionais. Em contrapartida, deve ser definida pelo menos uma chamada ECALL pública, já que estas representam as entradas no enclave. No exemplo apresentado na Listagem 2.1, tem-se apenas a função definida na linha 17 como ponto de entrada para o enclave. A função definida na linha 18 pode ser chamada apenas por uma função OCALL. Na seção `untrusted` estão contidas as definições das funções OCALLs, sendo que a função definida na linha 23 não pode efetuar chamadas de funções ECALL. Já a função definida na linha 24 pode invocar a função ECALL definida na linha 18.

Um arquivo EDL pode, opcionalmente, importar funções de outros arquivos EDL, fazendo isso de uma forma global ou seletiva. Para importar todas as funções de outro arquivo EDL utiliza-se a sintaxe definida na linha 2 da Listagem 2.1 e, para importar determinadas funções de outro arquivo EDL, é necessário informar a lista de funções a serem importadas, separando seus nomes por vírgula, como definido na linha 3.

Pode-se também efetuar a inclusão de arquivos de cabeçalho contendo definições de funções em C. As definições contidas nos arquivos de cabeçalho podem ser inseridas apenas para o contexto confiável da aplicação, fazendo sua inclusão na seção `trusted` (linha 15), apenas para o contexto não confiável, incluindo na seção `untrusted` (linha 21), ou para ambos os contextos, sendo incluídas na seção `enclave` (linhas 5 e 6). Além disso, é possível definir os tipos de dados que poderão ser utilizados pelo enclave, como

exemplificado nas linhas 8 a 13.

O arquivo também EDL provê suporte à definição de atributos de direção para os parâmetros: `[in]`, `[out]` ou `[user_check]`. O atributo `[in]`, quando utilizado em um ECALL, representa que o parâmetro será passado da aplicação para o enclave. Quando ele é utilizado em um OCALL, define que o parâmetro será retornado do enclave para a aplicação. Já o atributo `[out]` em uma função ECALL define que o parâmetro é retornado do enclave para a aplicação, e quando utilizado em um OCALL, ele define que o parâmetro é passado da aplicação para o enclave. Os atributos `[in]` e `[out]` podem ser utilizados de forma combinada, permitindo que o parâmetro seja transferido em ambas as direções.

Por fim, o atributo `[user_check]` é utilizado em casos onde as demandas de comunicação de dados entre o enclave e a aplicação exigem que as restrições impostas pelos parâmetros `[in]` e `[out]` sejam ignoradas. Isto pode ocorrer, por exemplo, quando um *buffer* é muito grande para residir totalmente na memória do enclave, sendo necessário fragmentá-lo em blocos menores e processá-lo através de uma série de chamadas ECALL, ou em situações onde a aplicação passa um ponteiro como parâmetro do ECALL. A utilização do atributo `[user_check]` implica em maiores riscos de segurança e, em virtude disso, devem ser adicionadas verificações adicionais no código para evitar o vazamento de informações confidenciais do enclave.

O arquivo EDL é utilizado pela ferramenta *Edger8r* para gerar a interface de comunicação entre o enclave e a parte não confiável da aplicação, ou seja, as rotinas de borda (ou *proxies*) entre o código confiável e o código confiável. A execução de *Edger8r* gera dois arquivos-fonte e dois arquivos cabeçalho para cada arquivo EDL, conforme apresentado na Figura 2.11. O arquivo-fonte `enclave_u.c` contém as rotinas que serão as portas de entrada para o enclave (funções ECALLs), efetuando as validações necessárias de acordo com os atributos de entrada e saída que foram definidos para cada um dos parâmetros. Já o arquivo-fonte `enclave_t.c` contém as rotinas que serão as portas de saída do enclave (funções OCALLs). Os arquivos de cabeçalho `enclave_u.h` e `enclave_t.h` devem ser incluídos no código não confiável da aplicação e no enclave, respectivamente, para que as rotinas de entrada e saída possam ser utilizados.

A ferramenta *Edger8r* pode ser executada como parte do processo de construção da aplicação ou de forma isolada. É importante destacar que sua execução deve ser feita em um ambiente protegido e livre de aplicações maliciosas, para garantir a integridade do código gerado. Além disso, o código contido no enclave é responsabilidade do desenvolvedor, portanto este deve revisar o código gerado pelo *Edger8r*.

2.6.2. O Arquivo de Configuração do Enclave

O *Enclave Configuration File* é um arquivo XML que faz parte do projeto, onde o programador pode definir alguns parâmetros do enclave. Este arquivo é utilizado pela ferramenta *SGX Sign* para efetuar a assinatura do enclave. Um exemplo deste arquivo de configuração é mostrado na Listagem 2.2. Todos os elementos contidos no arquivo são opcionais, caso algum dos elementos não esteja presente, ou se o arquivo não for criado, serão utilizados os valores padrões no processo de assinatura do enclave.

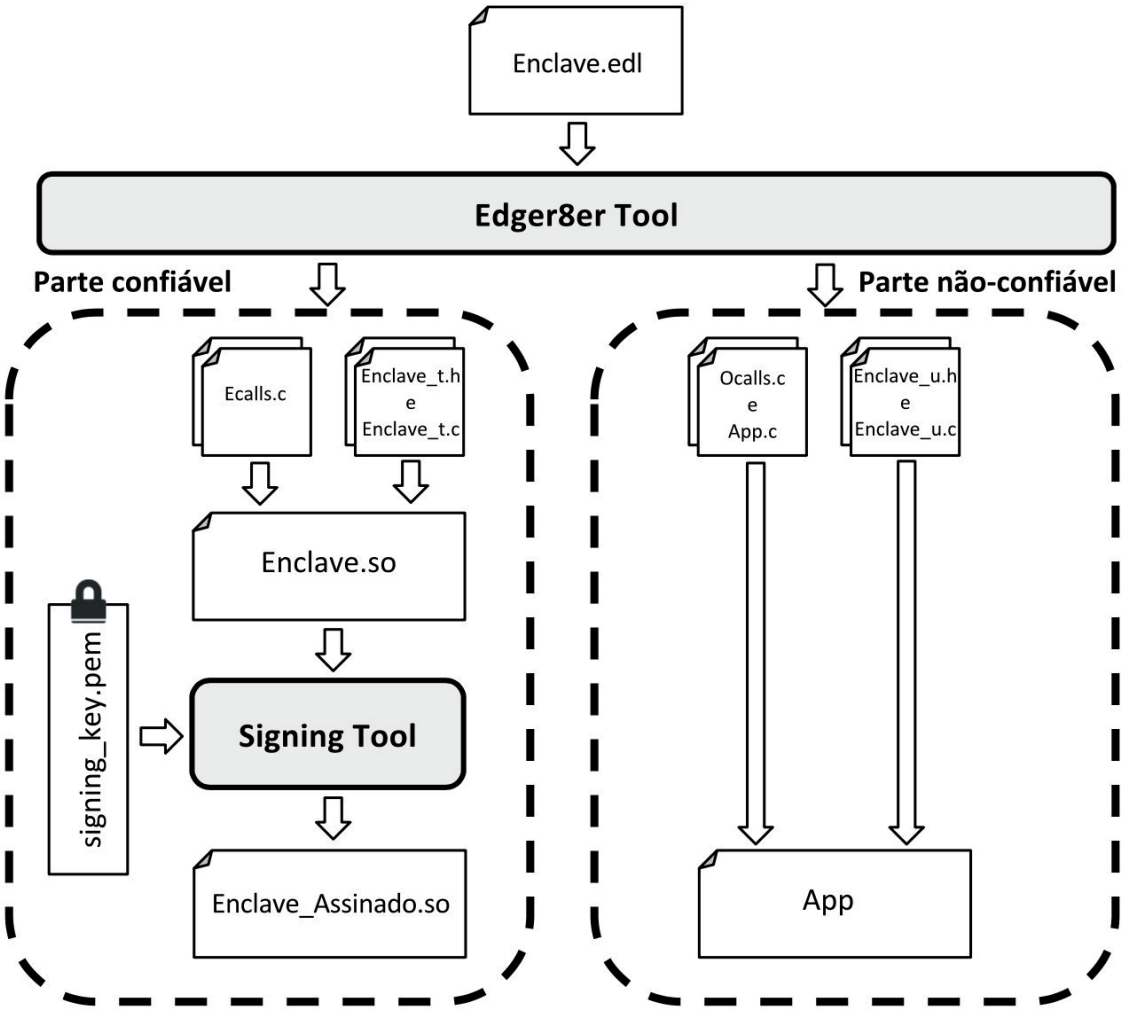


Figura 2.11. Processamento do arquivo EDL para a geração das interfaces das funções de borda do enclave.

Listagem 2.2. Arquivo de configuração do enclave (adaptado de [Intel 2016b]).

```
1  <EnclaveConfiguration>
2      <ProdID>1</ProdID>
3      <ISVSVN>1</ISVSVN>
4      <TCSNum>1</TCSNum>
5      <TCSPolicy>1</TCSPolicy>
6      <DisableDebug>0</DisableDebug>
7      <StackMaxSize>0x50000</StackMaxSize>
8      <HeapMaxSize>0x100000</HeapMaxSize>
9      <MiscSelect>0</MiscSelect>
10     <MiscMask>0xFFFFFFFF</MiscMask>
11 </EnclaveConfiguration>
```

Os elementos `ProdID` e `ISVSVN` representam, respectivamente, o ID do produto e o *Secure Version Number*, sendo atribuídos pelo desenvolvedor, e ambos têm seu valor padrão como zero. Cada *thread* em execução no enclave é associado à um *Thread Context Structures* (TCS). O elemento `TCSNum` define a quantidade de TCS , sendo que este deve ser maior que zero, e tem seu valor padrão como 1. O elemento `TCSPolicy`, quando atribuído com zero, indica que o TCS estará limitado ao contexto não confiável e, quando

atribuído com valor 1, indica que o TCS não estará limitado ao contexto não confiável, sendo este o seu valor padrão. O elemento `DisableDebug` indica a possibilidade de o enclave ser depurado; quando atribuído com zero, seu valor padrão, o enclave pode ser depurado e, caso atribuído com 1, a depuração não é permitida.

O elemento `StackMaxSize` define o tamanho máximo da pilha por *thread* e o elemento `HeapMaxSize` define o tamanho máximo do *heap* para o processo. Ambos devem ser definidos sempre com valores múltiplo de $4KB$, e seus valores padrão são $0x40000$ e $0x100000$, respectivamente. Caso não haja tamanho suficiente na pilha do enclave, uma chamada ECALL irá retornar o código de erro `SGX_ERROR_STACK_OVERRUN`, que indica que o valor do elemento `StackMaxSize` deve ser ajustado. Para evitar esse tipo de problema, os valores destes dois parâmetros podem ser ajustados utilizando o *Enclave Memory Measurement Tool*, que efetua a medição do uso real de memória por um enclave em tempo de execução, retornando o pico de uso da pilha e do *heap* de memória.

Por fim, os elementos `MiscSelect` e `MiscMask` são reservados para extensões futuras, tendo seus valores definidos com 0 e $0xFFFFFFFF$, respectivamente. Caso os valores sejam definidos de outra forma, o enclave pode não ser carregado corretamente.

2.6.3. Assinando o Enclave: A Ferramenta de Assinatura de Enclave

Conforme explicado na seção 2.3.3, assinar um enclave é um processo que envolve a criação de uma estrutura de assinatura com propriedades do enclave. A estrutura de assinatura do enclave é composta de 4 seções: cabeçalho, assinatura, corpo e *buffer* (os detalhes de cada seção podem ser encontrados em [Intel 2016b]. Esta estrutura permite à arquitetura SGX detectar quaisquer adulterações do enclave, e assim provar que o mesmo é legítimo e que foi carregado corretamente.

O *Software Development Kit* (SDK) fornecido pela Intel inclui uma ferramenta, chamada *Enclave Signing Tool*, que realiza o processo de assinatura. Esta ferramenta também avalia a imagem do enclave em busca de potenciais erros e problemas de segurança. Ao ser carregado, a assinatura é conferida para confirmar a integridade do enclave.

São admitidos dois métodos de assinatura: em passo único, usando a chave privada do usuário, ou em dois passos, usando uma ferramenta de assinatura externa. O método de dois passos protege a chave de assinatura em um ambiente separado e deve ser o método utilizado nas versões de produção da aplicação.

Para executar a ferramenta deve ser utilizado o comando com a seguinte sintaxe:

```
# sgx_sign <comando> [argumentos]
```

Os comandos possíveis são:

- `sign`: Assina o enclave em um passo único. Este comando deve receber os seguintes argumentos: `-enclave [arquivo]`, especificando o enclave a ser assinado; `-key [arquivo]`, especificando a chave de assinatura; e `-out [arquivo]`, especificando o arquivo retornado. Além destes argumentos, `-config [arquivo]`, que aponta o arquivo de configuração, também pode ser utilizado.

- `gendata`: É o primeiro passo da assinatura em dois passos. Gera o material a ser assinado pela ferramenta externa que consiste em seções de cabeçalho e corpo da estrutura de assinatura do enclave. A saída é um arquivo de 256 bytes. Este comando deve receber os argumentos `-enclave [arquivo]`, `-out [arquivo]` e, opcionalmente, `-config [arquivo]`.

- `catsig`: É o segundo passo da assinatura em dois passos. Gera o enclave assinado com a entrada de assinatura e a chave pública. A entrada de assinatura é gerada por uma ferramenta externa baseada nos dados gerados no passo anterior. Aqui são geradas as seções de assinatura e *buffer* da estrutura de assinatura do enclave.Este comando deve receber os argumentos `-enclave [arquivo]`, `-key [arquivo]`, `-out [arquivo]`, `-sig [arquivo]`, `-unisgned [arquivo]` e, opcionalmente, `-config [arquivo]`. O argumento `-sig` aponta o arquivo contendo a assinatura correspondente ao material de assinatura do enclave e o argumento `-unsigned` aponta o arquivo contendo o material gerado pelo comando `gendata`.

Quando um projeto de enclave é criado pela primeira vez, o usuário deve escolher entre usar uma chave de assinatura existente ou gerar uma chave automaticamente. Quando se optar por utilizar uma chave existente, ela deve estar no formato PEM e não deve estar cifrada. A assinatura em passo único utilizando uma chave existente é efetuada através do seguinte comando:

```
# sgx_sign sign -enclave enclave.so -config config.xml \
                -out enclave_assinado.so -key chave.pem
```

Os enclaves assinados em passo único só podem ser iniciado em modo *debug* ou *prerelease*, sendo que este processo de assinatura não pode ser utilizado para gerar enclaves em modo de produção.

Para o modo de produção, o SGX fornece um esquema de assinatura em dois passos. O primeiro passo ocorre ao final da compilação do enclave, quando a ferramenta de assinatura gera o material de assinatura do enclave. O comando para gerar o material de assinatura do enclave é o seguinte:

```
# sgx_sign gendata -enclave enclave.so -config config.xml \
                   -out enclave_hash.hex
```

O desenvolvedor deve levar o arquivo resultante (`enclave_hash.hex`) a uma plataforma de assinatura externa, onde a chave privada é armazenada, e assinar o arquivo de *hash* com o material de assinatura, e então levar o arquivo de assinatura resultante de volta para a plataforma de compilação.

No segundo passo, o desenvolvedor executa a ferramenta de assinatura com o comando `catsig` fornecendo as informações necessárias na linha de comando para adicionar o *hash* da chave pública e a assinatura à seção de metadados do enclave. Um exemplo do comando para o segundo passo de assinatura é dado a seguir:

```
# sgx_sign catsig -enclave enclave.so -config config.xml \
                  -out enclave_assinado.so -key chave_publica.pem \
                  -sig assinatura.hex -unsigned enclave_hash.hex
```

O arquivo `config.xml` pode ser omitido, sendo que, neste caso, as configurações padrão serão utilizadas.

O processo de assinatura em dois passos protege a chave de assinatura em uma plataforma separada. Este é o método de assinatura padrão, e o único método para assinar enclaves em modo de produção.

2.6.4. Bibliotecas Confiáveis e Não-Confiáveis do SDK

As bibliotecas confiáveis são bibliotecas estáticas desenvolvidas para serem ligadas ao enclave. Estas bibliotecas não podem usar instruções não suportadas pela arquitetura SGX e devem passar por um processo de revisão mais rigoroso que uma biblioteca convencional. Para desenvolver uma biblioteca confiável, deve-se gerar um arquivo `.lib` ao invés de um arquivo `.so`. Se um subconjunto da biblioteca fizer parte da interface do enclave com o domínio não-confiável, ele deverá ser declarado no arquivo EDL.

A ferramenta `sgx_edger8er` acrescenta automaticamente o nome do enclave nas funções de borda para evitar colisão dos nomes das funções, mas o desenvolvedor deve garantir a unicidade do nome das ECALLs. No entanto, quando dois enclaves importam a mesma ECALL de uma biblioteca confiável, o conjunto de funções de borda para cada enclave vão conter os mesmos nomes de funções *proxy* (de interface) não-confiáveis, o que irá gerar um erro. Para solucionar este problema, a ferramenta `sgx_edger8er` deve ser usada com a opção `-use-prefix` e acrescentar o prefixo com o nome do enclave nas ECALLs do código não-confiável.

A segurança do enclave depende na capacidade de se obter medição precisa de todo código e dados colocados dentro do enclave. Desta forma, enclaves não devem ser ligados dinamicamente em hipótese alguma. Ligações estáticas são permitidas.

A seguir serão relacionadas algumas das bibliotecas confiáveis e não-confiáveis disponibilizadas no SDK SGX, com a descrição de suas principais funcionalidades. Algumas delas são de uso obrigatório, ou seja, devem ser usadas nas aplicações envolvendo enclaves. A descrição detalhada de todas as funções disponíveis nas bibliotecas SGX e casos de uso podem ser encontrados em [Intel 2016b].

As seguintes bibliotecas são consideradas confiáveis; suas funções somente podem ser chamadas a partir de aplicações executando dentro do enclave:

- `libsgx_trts.a` (obrigatória): *Trusted Runtime System*. É uma biblioteca-chave do SDK. Fornece a lógica do ponto de entrada no enclave. Conta com funções auxiliares para determinar se um um dado endereço está, ou não, na região do enclave. Também é responsável pelo gerenciamento das interrupções e fornece um envelope para a instrução RDRAND, que retorna um número realmente aleatório gerado pelo hardware (`sgx_read_rand`).
- `libsgx_tstdc.a` (obrigatória): Biblioteca C padrão (*math*, *string*, etc). Algumas

funções não são suportadas, nos casos em que suas definições são inseguras (função `strcpy`), implicam em uso de instruções de CPU restritas, a implementação é muito grande para o enclave, precisam de informação do domínio não-confiável, ou a função não faz parte do padrão ou não é suportada por compilador específico.

- `libsgx_tstdcxx.a` (opcional): Biblioteca C++ em conformidade com o C++ 03 (incluindo STL). O suporte apresenta algumas limitações, como a impossibilidade de passar objetos pelos limites do enclave e de usar destrutores globais.

- `libsgx_tservice.a` (obrigatória): Esta biblioteca traz funcionalidades para dar suporte à manipulação segura e proteção de dados. Destacam-se as funcionalidades de criptografia (selagem e deselagem), estabelecimento de sessão Diffie-Hellman, suporte arquitetural ao enclave e funções relacionadas à atestação.

- `libsgx_tcrypto.a` (obrigatória): A biblioteca criptográfica conta com algoritmos como SHA256, Rijndael 128 CGM, CMAC 128, AES, ECC256 e ECDSA, embora com funcionalidades ainda bastante limitadas. Caso seja necessário usar outras funcionalidades, uma biblioteca criptográfica própria deve ser desenvolvida.

- `libsgx_tkey_exchange.a` (opcional): Biblioteca de troca de chaves confiável, que permite ao desenvolvedor trocar segredos entre servidores e enclaves. É usada em conjunto com a biblioteca de troca de chaves não-confiável.

As seguintes bibliotecas são consideradas não-confiáveis; suas funções somente podem ser chamadas a partir de aplicações executando fora do enclave:

- `libsgx_urts.so` (obrigatória): Fornece as funcionalidades necessárias para que as aplicações possam gerenciar os enclaves. Traz funções como `sgx_create_enclave` e `sgx_destroy_enclave`.

- `libsgx_uae_service.so` (obrigatória): Fornece acesso às funcionalidades dos enclaves arquiteturais (AEs), tanto aos enclaves quanto às aplicações não confiáveis.

- `libsgx_ukey_exchange.a` (opcional): Biblioteca de troca de chaves não-confiável.

Por fim, vale ressaltar que também existem bibliotecas como a `libsgx_trts_sim.a` e `libsgx_urts_sim.a`, que simulam as instruções SGX em software e podem ser usadas em plataformas sem suporte ao SGX, para fins de estudo e desenvolvimento.

2.6.5. Criando um Enclave

Esta seção demonstra detalhadamente como efetuar a criação de um enclave utilizando os métodos providos pelo SGX SDK, além de apresentar um exemplo simples da execução de chamadas ECALL e OCALL.

O projeto está dividido em três arquivos fonte (`app.c`, `lib_ocalls.c`, `lib_ecalls.c`), além do arquivo EDL para a geração das rotinas de borda. A Figura 2.12 relaciona as dependências dos arquivos fonte dentro da arquitetura da aplicação e esquematiza o seu fluxo de execução.

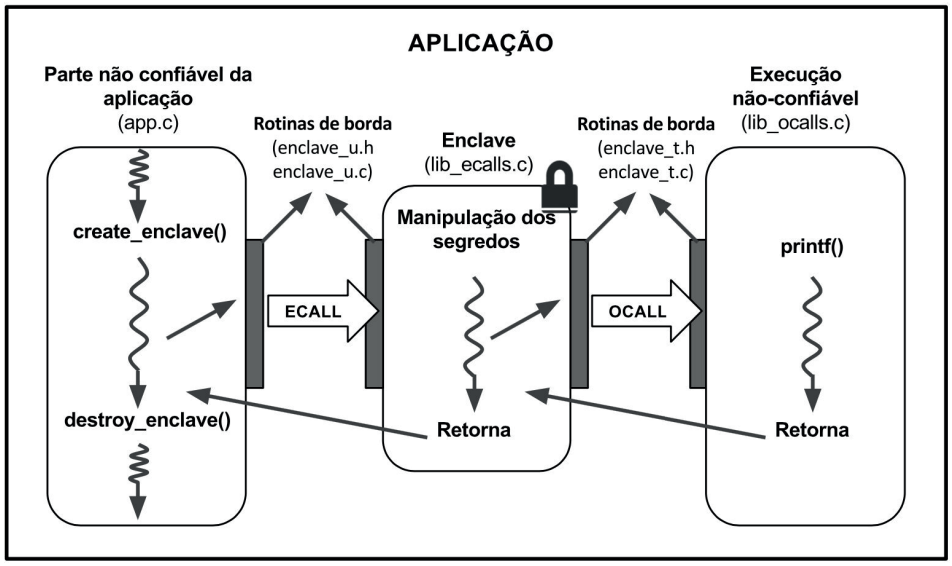


Figura 2.12. Composição e fluxo de execução da aplicação SGX.

O arquivo fonte `app.c` é apresentado na Listagem 2.3, contendo a função `main` e as operações necessárias para a criação do enclave, entrada no enclave, e destruição do mesmo.

Na linha 16 é efetuada a chamada da função `sgx_create_enclave`, responsável por carregar e inicializar o enclave. Para isto, utiliza-se a biblioteca do enclave (`ENCLAVE_FILENAME`), que deve estar assinada pela *SGX Sign Tool*, e um *token* que é criado na primeira inicialização do enclave, podendo ser armazenado para as inicializações seguintes do mesmo enclave, resultando em um ganho de desempenho. O parâmetro `SGX_DEBUG_FLAG` é provido pelo próprio SDK e pode ter seu valor atribuído no momento de compilação, sendo que o valor 0 indica que o enclave será inicializado em modo *release*, não permitindo que o mesmo seja depurado; já o valor 1 permite a depuração do enclave. O parâmetro `updated` retorna o valor 1 caso o `token` tenha sido atualizado na criação do enclave. Já o parâmetro `eid` retorna o identificador do enclave, necessário para a execução das funções ECALL.

Listagem 2.3. Arquivo fonte *app.c*.

```
1  #include "stdio.h"
2  #include "string.h"
3  #include "sgx_eid.h"
4  #include "sgx_urts.h"
5  #include "Enclave_u.h" //Gerado pelo Edger8r
6
7  #define ENCLAVE_FILENAME "enclave.signed.so"
8
9  int main(int argc, char *argv[]) {
10     char *data = "Hello World!";
11     sgx_enclave_id_t eid = 0;
12     sgx_launch_token_t token = {0};
```

```
13      sgx_status_t ret = SGX_ERROR_UNEXPECTED;
14      int updated = 0;
15
16      ret = sgx_create_enclave(ENCLAVE_FILENAME, SGX_DEBUG_FLAG, &token, &updated, &eid,
17          NULL);
18
19      if (ret != SGX_SUCCESS) {
20          printf("Erro na inicializacao do enclave ...\n");
21          return -1;
22      }
23
24      if (ecall_teste(eid, data) != SGX_SUCCESS) {
25          printf("ERRO na execucao da ecall ...\n");
26      } else {
27          printf("SUCESSO na execucao da ecall ...\n");
28      }
29
30      if(sgx_destroy_enclave(eid) != SGX_SUCCESS) {
31          printf("ERRO ao destruir o enclave.\n");
32          return -1;
33      }
34
35      return 0;
36  }
```

Na linha 24 é efetuada a chamada para a função ECALL (`ecall_teste`) passando por parâmetro o identificador do enclave. A função chamada neste ponto é gerada pela ferramenta *Edger8r*, ficando contida no arquivo fonte `Enclave_u.c` (conforme descrito no diagrama da Figura 2.11 da Seção 2.6.1) e contém as validações necessárias para efetuar a entrada no enclave. Após validados os parâmetros, a função `ecall_teste` contida no arquivo fonte `lib_ecalls.c`, e apresentada na Listagem 2.4, é chamada. Essa função recebe a mensagem passada por parâmetro e cria um *hash* de 256 *bits* a partir dessa mensagem, utilizando a função `sgx_sha256_msg` provida pelo SDK. Caso a criação do *hash* ocorra com sucesso, o mesmo é mostrado em tela, através da execução da função `ocall_print`.

Listagem 2.4. Arquivo fonte *lib_ecalls.c*.

```
1   #include "string.h"
2   #include "sgx_tcrypto.h"
3   #include "Enclave_t.h" //Gerado pelo Edger8r
4
5   void ecall_teste(char *c) {
6       sgx_status_t ret;
7       sgx_sha256_hash_t hash;
8
9       ret = sgx_sha256_msg((const uint8_t *)c, strlen(c), &hash);
10
11      if (ret == SGX_SUCCESS) {
12          ocall_print("sgx_sha256_msg SUCESSO");
13          ocall_print((char*)hash);
14      } else {
15          ocall_print("sgx_sha256_msg ERRO");
16      }
17  }
```

A função `ocall_print` está contida no arquivo fonte `lib_ocalls.c`, que é apresentado na Listagem 2.5. Essa função apenas faz a impressão na tela da mensagem recebida por parâmetro. A execução de uma função OCALL para realizar essa tarefa se faz necessária, já que o enclave não pode executar *system calls*. Da mesma forma que ocorre na execução de uma chamada ECALL, os parâmetros passados para a função também são

validados em uma função homônima contida no arquivo fonte `Enclave_t.c` gerado pela ferramenta *Edger8r*.

Listagem 2.5. Arquivo fonte *lib_ocalls.c*.

```
1 #include "stdio.h"
2 #include "Enclave_u.h" //Gerado pelo Edger8r
3
4 void ocall_print(char *c) {
5     printf("%s\n", c);
6 }
```

Quando o enclave não for mais necessário, deve-se destruí-lo executando a função `sgx_destroy_enclave`, que recebe por parâmetro o identificador do enclave a ser destruído.

Por fim, a Listagem 2.6 apresenta o conteúdo do arquivo EDL, que é utilizado pela ferramenta *Edger8r* para gerar as rotinas que irão efetuar as validações dos parâmetros passados, conforme os atributos utilizados. Nas ECALLs, o atributo `[in]` indica a entrada de parâmetro no enclave, enquanto na OCALLs ele é utilizado para indicar a entrada de parâmetro no universo não-confiável. Em ambos os casos, o atributo `[string]` indica que o parâmetro é uma cadeia de caracteres terminada com *NULL*. A rotina de borda primeiro determina o tamanho da string e, após isso, copia o seu conteúdo para o enclave (quando utilizada em uma ECALL), para inserir o terminador *NULL*. O atributo `[string]` pode ser utilizado em conjunto com o atributo `[in]` ou em uma combinação dos atributos `[in]` e `[out]`, nunca com o atributo `[out]` apenas.

Listagem 2.6. Arquivo EDL contendo as definições das funções ECALLs e OCALLs.

```
1 enclave {
2     trusted {
3         public void ecall_teste([in, string] char *c);
4     };
5
6     untrusted {
7         void ocall_print([in, string] char *c);
8     };
9 };
```

2.6.6. Selando Dados

Esta seção é mostra como pode-se efetuar a selagem de dados através de um enclave SGX, bem como a abertura posterior dos dados selados. O exemplo aqui apresentado é divido em dois arquivos fonte (`app.c` e `lib_ecalls.c`), não tendo nenhuma função OCALL disponível.

Este exemplo tem a seguinte sequência: primeiro é efetuada a criação do enclave, em seguida é realizada uma ECALL passando os dados que deverão ser selados. Os dados selados pelo enclave são retornados à aplicação, e, por sua vez, são passados para uma segunda ECALL que abrirá novamente estes dados e retornará os mesmos para a aplicação.

O arquivo fonte `app.c` é apresentado na Listagem 2.7, tendo uma estrutura semelhante ao apresentado anteriormente na Listagem 2.3 para efetuar a criação e destruição de um enclave.

Listagem 2.7. Arquivo fonte *app.c*.

```
1 #include "stdio.h"
```

```
2  #include "string.h"
3  #include "sgx_eid.h"
4  #include "sgx_urts.h"
5  #include "Enclave_u.h" // Gerado pelo Edger8r
6
7  #define ENCLAVE_FILENAME "enclave.signed.so"
8
9  int main(int argc, char *argv[])
10 {
11     char *data = "Hello World!!!";
12     uint8_t *sealed_data, *unsealed_data;
13     uint32_t data_size = strlen(data), sealed_data_size, unsealed_data_size, i;
14     sgx_enclave_id_t eid = 0;
15     sgx_launch_token_t token = {0};
16     sgx_status_t ret = SGX_ERROR_UNEXPECTED;
17     int updated = 0;
18
19     ret = sgx_create_enclave(ENCLAVE_FILENAME, SGX_DEBUG_FLAG, &token, &updated, &eid,
20         NULL);
21
22     if (ret != SGX_SUCCESS) {
23         printf("Erro na inicializacao do enclave ...\n");
24         return -1;
25     }
26
27     ret = ecall_get_sealed_data_size(eid, data_size, &sealed_data_size);
28
29     if (ret != SGX_SUCCESS) {
30         printf("Erro ao calcular o tamanho dos dados selados.\n");
31         return -1;
32     }
33
34     sealed_data = (uint8_t*) malloc(sealed_data_size);
35
36     ret = ecall_seal_data(eid, (uint8_t*)data, data_size, sealed_data,
37         sealed_data_size);
38
39     if (ret != SGX_SUCCESS) {
40         printf("Erro ao selar dados ...\n");
41         return -1;
42     }
43
44     printf("Dados selados: \n");
45     for(i = 0; i < sealed_data_size; i++)
46         printf("%02x ", sealed_data[i]);
47     printf("\n");
48
49     ret = ecall_get_unsealed_data_size(eid, sealed_data, sealed_data_size,
50         &unsealed_data_size);
51
52     if (ret != SGX_SUCCESS) {
53         printf("Erro ao calcular o tamanho dos dados abertos.\n");
54         return -1;
55     }
56
57     unsealed_data = (uint8_t*) malloc(unsealed_data_size);
58
59     ret = ecall_unseal_data(eid, sealed_data, sealed_data_size, unsealed_data,
60         unsealed_data_size);
61
62     if (ret != SGX_SUCCESS) {
63         printf("Erro ao abrir dados ...\n");
64         return -1;
65     }
66
67     printf("Dados abertos: \n%s\n", (char*)unsealed_data);
68
69     if(sgx_destroy_enclave(eid) != SGX_SUCCESS) {
70         printf("ERRO ao destruir o enclave.\n");
71         return -1;
```

```
72      }
73
74      return 0;
75  }
```

Após a criação do enclave, o primeiro passo é verificar qual será o tamanho dos dados após selados (linha 27). Isso se faz necessário pois deve-se alocar memória para que esses dados sejam armazenados. Esta operação é efetuada através de uma chamada ECALL, que recebe como parâmetro de entrada o tamanho dos dados a serem selados. Esta função, assim como as outras funções ECALL deste projeto, serão detalhadas mais adiante.

Após verificar qual será o tamanho dos dados selados, é feita a alocação de memória para armazená-los, e então efetua-se a chamada de uma segunda função ECALL (linha 36) que efetuará a selagem dos dados. Esta função recebe como parâmetros de entrada os dados a serem selados, o tamanho destes e o tamanho dos dados após selados, e como parâmetro de saída os dados selados. Após a selagem dos dados, os mesmos são impressos em tela *byte* a *byte*.

Por fim, de forma semelhante, é efetuado o processo para abrir os dados selados. Primeiramente, na linha 49, é executada uma chamada ECALL para saber qual será o tamanho dos dados após abertos (neste caso, esta etapa poderia ser omitida, pois neste exemplo o tamanho dos dados já é previamente conhecido), em seguida a memória necessária é alocada, e então uma segunda chamada ECALL é efetuada para abrir os dados (linha 59).

As funções ECALL estão definidas no arquivo fonte `lib_ecalls.c`, que é é apresentado na Listagem 2.8. A primeira ECALL, `ecall_get_sealed_data_size`, é responsável por calcular o tamanho dos dados após serem selados; para isso, recebe como parâmetro de entrada o tamanho dos dados abertos. Tal cálculo é efetuado pela função `sgx_calc_sealed_data_size`, pertencente ao SDK do SGX, que recebe dois parâmetros:

- `add_mac_txt_size`: Tamanho dos dados adicionais. Estes dados não serão cifrados, mas irão compor o cálculo do MAC (*Message Authentication Code*). Neste exemplo, é informado o valor 0 como parâmetro, pois não há dados adicionais.
- `txt_encrypt_size`: Tamanho dos dados que serão cifrados.

Listagem 2.8. Arquivo fonte *lib_ecalls.c*.

```
1  #include "sgx_tseal.h"
2  #include "Enclave_t.h" // Gerado pelo Edger8r
3
4  void ecall_get_sealed_data_size(uint32_t data_size, uint32_t *sealed_data_size) {
5      *sealed_data_size = sgx_calc_sealed_data_size(0, data_size);
6  }
7
8  void ecall_seal_data(uint8_t *data, uint32_t data_size, uint8_t *sealed_data,
9      uint32_t sealed_data_size) {
10
11     sgx_status_t ret = SGX_ERROR_UNEXPECTED;
12
13     ret = sgx_seal_data(0, NULL, data_size, data, sealed_data_size,
14         (sgx_sealed_data_t*)sealed_data);
15 }
```

```
16
17 void ecall_get_unsealed_data_size(uint8_t *sealed_data, uint32_t sealed_data_size,
18     uint32_t *unsealed_data_size) {
19
20     *unsealed_data_size = sgx_get_encrypt_txt_len((sgx_sealed_data_t*)sealed_data);
21 }
22
23 void ecall_unseal_data(uint8_t *sealed_data, uint32_t sealed_data_size,
24     uint8_t *unsealed_data, uint32_t unsealed_data_size) {
25
26     sgx_status_t ret = SGX_ERROR_UNEXPECTED;
27
28     ret = sgx_unseal_data((sgx_sealed_data_t*)sealed_data, NULL, 0, unsealed_data,
29         &unsealed_data_size);
30 }
```

A função `ecall_seal_data` é utilizada para efetuar a selagem dos dados, recebendo quatro parâmetros: os dados a serem selados, o tamanho dos dados a serem selados, os dados após a selagem (parâmetro de saída) e o tamanho dos dados após selados. Esta função faz uma chamada à função `sgx_seal_data`, disponível no SDK, passando a ela seis parâmetros:

- `additional_MACtext_length`: Tamanho dos dados adicionais que serão utilizados no cálculo do MAC. Neste exemplo não há dados adicionais, por este motivo é passado o valor 0.
- `*p_additional_MACtext`: Dados adicionais que serão utilizados no cálculo do MAC. Neste exemplo não há dados adicionais, por este motivo é passado o valor `NULL`.
- `text2encrypt_length`: Tamanho dos dados a serem selados.
- `*p_text2encrypt`: Dados a serem selados.
- `sealed_data_size`: Tamanho dos dados após a selagem.
- `*p_sealed_data`: Dados selados. Este é um parâmetro de saída da função.

A função `sgx_seal_data` sela os dados para o autor do enclave, ou seja, qualquer enclave produzido pelo mesmo autor estará apto a abrir os dados. O SDK também disponibiliza a função `sgx_seal_data_ex`, que funciona de forma semelhante, permitindo que seja escolhida entre a selagem de dados para o autor ou para o enclave, restringindo a abertura desses dados apenas para o enclave que os selou.

A função `ecall_get_unsealed_data_size` é responsável por calcular o tamanho dos dados após abertos. Para isso, recebe como parâmetros de entrada os dados selados e o tamanho destes, tendo como parâmetro de saída o tamanho dos dados após abertos. Este cálculo é feito através da função `sgx_get_encrypt_txt_len`, disponível no SDK do SGX, que recebe como único parâmetro os dados selados na forma da estrutura `sgx_sealed_data_t`.

Por fim, a função `ecall_unseal_data` efetua a abertura dos dados selados, tendo como parâmetros os dados selados, o tamanho destes, os dados abertos (parâmetro

de saída), e o tamanho dos dados após abertos. A abertura dos dados é feita através da função `sgx_unseal_data`, que requer os seguintes parâmetros:

- `*p_sealed_data`: Dados selados, na forma da estrutura `sgx_sealed_data_t`.
- `*p_additional_MACtext`: Parâmetro de saída que retorna os dados opcionais que fazem parte do cálculo do MAC. No caso do exemplo apresentado, é passado o valor `NULL` por não haver dados adicionais. Caso existam dados adicionais, deverá ser alocada memória para retornar estes dados.
- `*p_additional_MACtext_length`: Tamanho dos dados adicionais que fazem parte do cálculo do MAC. Na existência de dados adicionais, deve-se utilizar a função `sgx_get_add_mac_txt_len` para calcular o tamanho destes. Caso não existam dados adicionais, o valor passado deve ser 0.
- `*p_decrypted_text`: Parâmetro de saída que retorna os dados decifrados.
- `*p_decrypted_text_length`: Parâmetro que recebe o tamanho do *buffer* de dados depois de decifrados. Este parâmetro também retorna o tamanho real do *buffer* após decifrados.

Por fim, a Listagem 2.9 apresenta o arquivo EDL que contém as definições para as funções ECALL do enclave. Todas ECALLs são definidas com acesso público, permitindo que estas sejam chamadas pela aplicação.

Listagem 2.9. Arquivo EDL contendo as definições das funções ECALLs.

```
enclave {
    trusted {
        public void ecall_get_sealed_data_size(uint32_t data_size,
            [out] uint32_t *sealed_data_size);

        public void ecall_seal_data([in, size=data_size] uint8_t *data,
            uint32_t data_size, [out, size=sealed_data_size] uint8_t *sealed_data,
            uint32_t sealed_data_size);

        public void ecall_get_unsealed_data_size(
            [in, size=sealed_data_size] uint8_t *sealed_data,
            uint32_t sealed_data_size, [out] uint32_t *unsealed_data_size);

        public void ecall_unseal_data([in, size=sealed_data_size] uint8_t *sealed_data,
            uint32_t sealed_data_size,
            [out, size=unsealed_data_size] uint8_t *unsealed_data,
            uint32_t unsealed_data_size);
    };
};
```

Um ponto importante a ser observado é a utilização do atributo `[size]` em conjunto com os atributos `[in]` e `[out]` quando o parâmetro é um *buffer* de dados. Isso é necessário para que, ao se utilizar o atributo `[in]`, os dados sejam copiados para a área protegida de memória, para serem utilizados pelo enclave. Na utilização do atributo `[out]` os dados serão copiados para a área não protegida de memória, para que possam ser utilizados pela aplicação. Todo esse processo de cópia dos dados é executado pelas rotinas de borda criadas pela ferramenta *Edger8r*.

2.6.7. Atestação Entre Enclaves

Esta seção apresenta um exemplo de atestação local entre dois enclaves (ilustrado na Figura 2.9), permitindo a criação de um canal seguro de comunicação para efetuar o envio de uma mensagem. Este canal seguro é criado através de uma troca de chaves *Diffie-Hellman* entre os enclaves, sendo que a chave acordada é posteriormente utilizada para cifrar a mensagem a ser enviada. O processo de troca de chaves entre os enclaves se divide em 12 etapas representadas no diagrama da Figura 2.13, numeradas na ordem em que ocorrem.

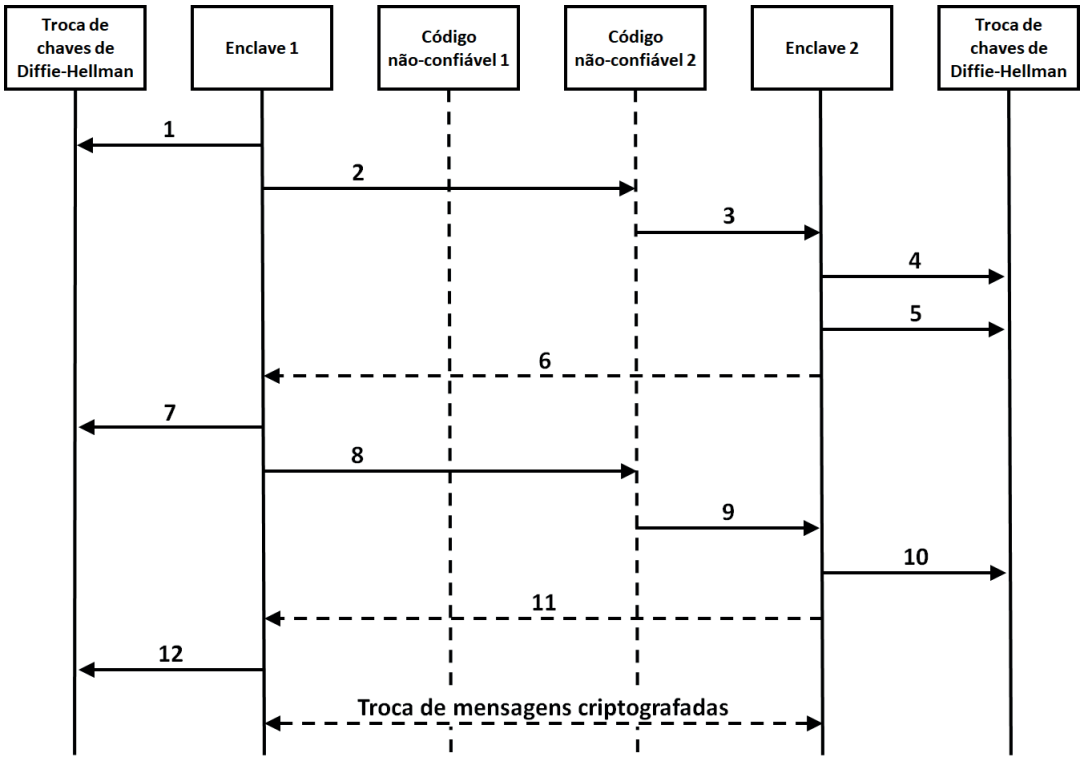


Figura 2.13. Etapas necessárias para efetuar a atestação local entre dois enclaves utilizando troca de chaves *Diffie-Hellman* (adaptado de [Intel 2016b]).

1. O enclave que deseja iniciar a atestação (identificado neste exemplo como Enclave 1) efetua uma chamada para a função `sgx_dh_init_session` de forma a iniciar a sessão para a troca de chaves, na condição de inicializador da requisição;

2. O Enclave 1 efetua uma chamada OCALL identificando a instância do enclave de destino (Enclave 2);

3. A aplicação efetua uma ECALL para o Enclave 2 para que este também inicie uma sessão para a troca de chaves;

4. O Enclave 2 inicia o processo de criar uma sessão para a troca de chaves, também utilizando a função `sgx_dh_init_session`, mas na condição de respondedor da requisição;

5. O Enclave 2 gera uma estrutura `sgx_dh_msg1_t`, através da função `sgx_dh_responder_gen_msg1`, que contém uma chave pública baseada em

uma curva elíptica NIST P-256, além da identificação do enclave destinatário da chave;

6. A estrutura `sgx_dh_msg1_t` é retornada ao Enclave 1. Nesta etapa conclui-se o passo 1 apresentado na Figura 2.9;

7. O Enclave 1 valida a estrutura `sgx_dh_msg1_t` através da função *sgx_dh_initiator_proc_msg1*, gerando também uma segunda estrutura, `sgx_dh_msg2_t`, que contém uma chave pública do Enclave 1, a sua estrutura *EREPORT* e um valor CMAC (*Cipher-based Message Authentication Code*), que será validada pelo Enclave 2 para atestar que o Enclave 1 está sendo executado na mesma plataforma;

8. O Enclave 1 executa uma chamada OCALL para enviar ao Enclave 2 a estrutura `sgx_dh_msg2_t` gerada;

9. A aplicação executa uma chamada ECALL para encaminhar as informações para o Enclave 2. Nesta etapa conclui-se o passo 2 apresentado na Figura 2.9;

10. O Enclave 2 valida a estrutura `sgx_dh_msg2_t` gerada pelo Enclave 1 através da função `sgx_dh_responder_proc_msg2` e, a partir desta, gera uma estrutura `sgx_dh_msg3_t` que irá conter a estrutura *EREPORT* do Enclave 2 e um valor CMAC calculado com base nesta estrutura. Nesta etapa também é gerada uma chave de 128 *bits*, definida como AEK, que será utilizada para cifrar as mensagens trocadas entre os enclaves;

11. A estrutura `sgx_dh_msg3_t` e a chave AEK gerada são retornadas ao Enclave 1. Nesta etapa conclui-se o passo 3 apresentado na Figura 2.9;

12. O Enclave 1 valida a estrutura `sgx_dh_msg3_t` através da função `sgx_dh_initiator_proc_msg3` para atestar que o Enclave 2 está sendo executado na mesma plataforma.

Após os enclaves confirmarem suas identidades e atestarem que ambos estão sendo executados na mesma plataforma, eles podem iniciar a troca de mensagens, cifrando-as com a chave AEK de 128 *bits* obtida no acordo de *Diffie-Hellman*. Para efetuar a cifragem das mensagens, utiliza-se a função `sgx_rijndael128GCM_encrypt`, que recebe os seguintes parâmetros:

- `*p_key`: Um ponteiro para a chave a ser utilizada no processo de criptografia. A chave deve conter 128 *bits*;
- `*p_src`: Um ponteiro para os dados que serão cifrados. Este valor pode ser nulo se houver dados adicionais (parâmetro `*p_aad`);
- `src_len`: Tamanho do *buffer* de dados que será cifrado. Este valor pode ser 0, desde que os parâmetros `*p_src` e `*p_dst` sejam nulos e o parâmetro `aad_len` maior que zero;

- `*p_dst`: Um ponteiro para um *buffer* onde os dados cifrados serão salvos. Este *buffer* deve ser previamente alocado;
- `*p_iv`: Um ponteiro para o vetor de inicialização utilizado no cálculo AES-GCM, com tamanho recomendado de 12 *bytes*;
- `iv_len`: Tamanho do ponteiro de inicialização;
- `*p_aad`: Um ponteiro para um *buffer* de dados adicional que será utilizado para efetuar o cálculo GCM MAC. Este parâmetro é opcional, podendo ser nulo, e o conteúdo deste *buffer* não será cifrado;
- `aad_len`: Tamanho do *buffer* de dados adicional. Caso o *buffer* adicional seja nulo, o valor deste parâmetro deve ser zero;
- `*p_out_mac`: Parâmetro de saída contendo o resultado do cálculo GCM MAC efetuado sobre os dados cifrados e os dados adicionais.

Para efetuar a decifragem dos dados utiliza-se a função `sgx_rijndael128GCM_decrypt`, que recebe os mesmos parâmetros que a função `sgx_rijndael128GCM_encrypt`. No processo de decifragem o parâmetro `*p_src` recebe os dados cifrados e o parâmetro `*p_dst` retorna os dados decifrados. Além disso, o parâmetro `*p_out_mac`, neste caso, recebe o valor GCM MAC calculado no processo de cifragem.

O projeto de exemplo para atestação local é divido em quatro arquivos fonte. Os arquivos `lib_ecalls1.c` e `lib_ecalls2.c` contêm as funções ECALLs dos enclaves 1 e 2, respectivamente. O arquivo fonte `lib_ocalls.c` contém as funções OCALLs de ambos os enclaves. Por fim, o arquivo fonte `app.c` contém o código da aplicação, responsável por instanciar os dois enclaves e efetuar a chamada à função ECALL do enclave 1, responsável por enviar a mensagem ao enclave 2.

A Listagem 2.10 apresenta o arquivo fonte com as funções ECALL do enclave 1. A função `Enclave1_ecall_send_message` é responsável por iniciar o processo de atestação e, após concluí-lo, enviar a mensagem cifrada ao enclave 2. Na linha 65 efetua-se a chamada à função `Enclave1_create_session`, que irá efetuar o processo de atestação. Dentro dessa função, na linha 24, é iniciado o processo de atestação, criando uma sessão para a troca de chaves (etapa 1 do diagrama da Figura 2.13). Após isso, é efetuada uma chamada OCALL (etapa 2, linha 29) que irá executar uma chamada ECALL para o enclave 2 (etapa 3), apresentada na Listagem 2.10 como `Enclave2_ecall_session_request`, que será responsável por iniciar a sessão no enclave 2 (etapa 4) e criar a estrutura `sgx_dh_msg1_t` (etapa 5) que será validada pelo Enclave 1.

Listagem 2.10. Arquivo fonte *lib_ecalls.c* do Enclave 1.

```
1 #include "sgx_eid.h"
2 #include "sgx_report.h"
3 #include "sgx_eid.h"
4 #include "sgx_ecp_types.h"
5 #include "sgx_dh.h"
6 #include "sgx_tseal.h"
```

```
7  #include "stdlib.h"
8  #include "string.h"
9  #include "Enclave1_t.h"
10
11 sgx_key_128bit_t session_dh_aek;
12 sgx_dh_session_t sgx_dh_session;
13
14 sgx_status_t Enclave1_create_session(sgx_enclave_id_t src_enclave_id,
15     sgx_enclave_id_t dest_enclave_id) {
16
17     sgx_status_t status = SGX_SUCCESS;
18     sgx_key_128bit_t dh_aek;
19     sgx_dh_msg1_t dh_msg1;  //Diffie-Hellman Message 1
20     sgx_dh_msg2_t dh_msg2;  //Diffie-Hellman Message 2
21     sgx_dh_msg3_t dh_msg3;  //Diffie-Hellman Message 3
22     sgx_dh_session_enclave_identity_t responder_identity;
23
24     status = sgx_dh_init_session(SGX_DH_SESSION_INITIATOR, &sgx_dh_session);
25     if(status != SGX_SUCCESS) {
26         return status;
27     }
28
29     status = Enclave1_ocall_session_request(dest_enclave_id, &dh_msg1);
30     if (status != SGX_SUCCESS) {
31         return status;
32     }
33
34     status = sgx_dh_initiator_proc_msg1(&dh_msg1, &dh_msg2, &sgx_dh_session);
35     if (status != SGX_SUCCESS) {
36         return status;
37     }
38
39     status = Enclave1_ocall_exchange_report(dest_enclave_id, &dh_msg2, &dh_msg3,
40         &dh_aek);
41     if (status != SGX_SUCCESS) {
42         return status;
43     }
44
45     status = sgx_dh_initiator_proc_msg3(&dh_msg3, &sgx_dh_session, &dh_aek,
46         &responder_identity);
47     if (status != SGX_SUCCESS) {
48         return status;
49     }
50
51     memcpy(&session_dh_aek, dh_aek, sizeof(sgx_key_128bit_t));
52
53     return SGX_SUCCESS;
54 }
55
56 void Enclave1_ecall_send_message(sgx_enclave_id_t src_enclave_id,
57     sgx_enclave_id_t dest_enclave_id, const char* message) {
58
59     sgx_status_t status;
60     uint32_t src_len = strlen(message);
61     uint8_t p_dest2[src_len];
62     sgx_aes_gcm_data_t* secure_message;
63     size_t message_size;
64
65     status = Enclave1_create_session(src_enclave_id, dest_enclave_id);
66     if(status != SGX_SUCCESS) {
67         ocall_print("Enclave1_create_session ERRO");
68         return;
69     }
70
71     message_size = sizeof(sgx_aes_gcm_data_t) + src_len;
72     secure_message = (sgx_aes_gcm_data_t*)malloc(message_size);
73     secure_message->payload_size = src_len;
74
75     status = sgx_rijndael128GCM_encrypt(&session_dh_aek, (uint8_t*)message, src_len,
76         secure_message->payload, secure_message->reserved,
```

```
77          sizeof(secure_message->reserved), NULL, 0, &(secure_message->payload_tag));
78      if(status != SGX_SUCCESS) {
79          ocall_print("sgx_rijndael128GCM_encrypt ERRO");
80          return;
81      }
82
83      status = Enclave1_ocall_send_request(dest_enclave_id, secure_message,
84          message_size);
85      if(status != SGX_SUCCESS) {
86          ocall_print("ERRO ao enviar mensagem");
87          return;
88      }
89  }
```

Após criada, a estrutura `sgx_dh_msg1_t` é retornada para o Enclave 1 (etapa 6) e então validada por ele (chamada na linha 34 da Listagem 2.10, correspondente à etapa 7 do diagrama da Figura 2.13). A mesma função que valida a estrutura `sgx_dh_msg1_t` gera uma segunda estrutura, `sgx_dh_msg2_t`, que contém as informações do Enclave 1, a qual é encaminhada ao Enclave 2, efetuando uma chamada OCALL (etapa 8, linha 39 da Listagem 2.10) e posteriormente uma chamada ECALL para o Enclave 2 (etapa 9). O Enclave 2 então valida esta estrutura (etapa 10, linha 43 da Listagem 2.11), gerando uma terceira estrutura, `sgx_dh_msg3_t`, e uma chave de 128 *bits* (`dh_aek`), que são retornadas para o Enclave 1 (etapa 11). Por fim, a última etapa de atestação é a validação da estrutura `sgx_dh_msg3_t` por parte do Enclave 1 (linha 45 da Listagem 2.10).

Finalizado o processo de atestação entre os enclaves, as mensagens trocadas entre eles são cifradas utilizando a chave AEK de 128 *bits* obtida durante o processo de atestação. Neste exemplo, a cifragem da mensagem é efetuada na linha 75 da Listagem 2.10, e a mensagem cifrada é encaminhada através de uma função OCALL (linha 83). Esta função OCALL é responsável por encaminhar a mensagem para o Enclave 2, efetuando uma chamada ECALL para este. O Enclave 2 recebe a mensagem e decifra ela (linha 59 da Listagem 2.11).

Listagem 2.11. Arquivo fonte *lib_ecalls.c* do Enclave 2.

```
1   #include "sgx_eid.h"
2   #include "sgx_report.h"
3   #include "sgx_eid.h"
4   #include "sgx_ecp_types.h"
5   #include "sgx_dh.h"
6   #include "sgx_tseal.h"
7   #include "string.h"
8   #include "stdlib.h"
9   #include "Enclave2_t.h"
10
11  sgx_key_128bit_t session_dh_aek;
12  sgx_dh_session_t sgx_dh_session;
13
14  void Enclave2_ecall_session_request(sgx_dh_msg1_t *dh_msg1) {
15      sgx_status_t status = SGX_SUCCESS;
16
17      if(!dh_msg1) {
18          ocall_print("dh_msg1 INVALIDA!");
19          return;
20      }
21
22      ocall_print("Enclave 2 sgx_dh_init_session");
23      status = sgx_dh_init_session(SGX_DH_SESSION_RESPONDER, &sgx_dh_session);
24      if(status != SGX_SUCCESS) {
25          ocall_print("ERRO!");
26          return;
```

```
27      }
28
29      ocall_print("Enclave 2 sgx_dh_responder_gen_msg1");
30      status = sgx_dh_responder_gen_msg1(dh_msg1, &sgx_dh_session);
31      if(status != SGX_SUCCESS) {
32          ocall_print("ERRO!");
33          return;
34      }
35  }
36
37  void Enclave2_ecall_exchange_report(sgx_dh_msg2_t *dh_msg2,
38      sgx_dh_msg3_t *dh_msg3, sgx_key_128bit_t *dh_aek) {
39
40      sgx_dh_session_enclave_identity_t initiator_identity;
41
42      ocall_print("Enclave 2 sgx_dh_responder_proc_msg2");
43      sgx_status_t status = sgx_dh_responder_proc_msg2(dh_msg2, dh_msg3, &sgx_dh_session,
44          dh_aek, &initiator_identity);
45      if(status != SGX_SUCCESS) {
46          ocall_print("ERRO!");
47          return;
48      }
49
50      memcpy(&session_dh_aek, dh_aek, sizeof(sgx_key_128bit_t));
51  }
52
53  void Enclave2_ecall_receive_message(sgx_aes_gcm_data_t* message,
54      size_t message_size) {
55
56      sgx_status_t status;
57      uint8_t p_dest[message->payload_size];
58
59      status = sgx_rijndael128GCM_decrypt(&session_dh_aek, message->payload,
60          message->payload_size, p_dest, message->reserved, sizeof(message->reserved),
61          NULL, 0, &(message->payload_tag));
62
63      if(status != SGX_SUCCESS) {
64          ocall_print("sgx_rijndael128GCM_decrypt ERRO");
65      } else {
66          ocall_print((char*)p_dest);
67      }
68  }
```

A mensagem cifrada pode ser encapsulada em uma estrutura `sgx_aes_gcm_data_t`, que irá conter o tamanho da mensagem (campo `payload_size`), um campo reservado para alinhar os dados em 16 *bytes* (campo `reserved`) , a mensagem cifrada e o conteúdo adicional autenticado, mas não cifrado (campo `payload`). Além destes, a estrutura também contém um quarto campo, `payload_tag`, que armazena o valor AES-GMAC calculado sobre os outros três campos da estrutura.

2.7. Considerações Finais

Neste trabalho foram apresentados alguns mecanismos de segurança baseados em *hardware*, que são utilizados para garantir a segurança de aplicações e a confidencialidade dos dados dos usuários, com enfoque principal na arquitetura Intel SGX – *Software Guard Extensions*, que permite a execução de aplicações em um ambiente seguro, denominado enclave, além de selagem de dados e atestação entre aplicações, de forma local ou remota.

Apesar de ser uma arquitetura relativamente nova, tendo sido disponibilizada no final do ano de 2015, já existem diversos trabalhos que fazem uso dela, em diferentes áreas

de aplicação, como descrito na Seção 2.4, além de questões ainda em aberto a respeito desta nova tecnologia, conforme abordado na Seção 2.5.

Por fim, os exemplos práticos apresentados na Seção 2.6 visam fornecer uma base de conhecimento para a utilização do SGX SDK para a construção de aplicações reais. Para priorizar a didática de apresentação, os exemplos apresentados abrem mão de algumas premissas de segurança como, por exemplo, gerar os dados a serem selados ou transferidos entre enclaves dentro do próprio enclave (visto que os dados gerados fora do enclave estão sujeitos a inspeções por outras aplicações). Para o desenvolvimento de aplicações seguras reais, essas premissas de segurança devem ser consideradas.

Referências

[Amin et al. 2008] Amin, M., Khan, S., Ali, T., and Gul, S. (2008). Trends and directions in trusted computing: Models, architectures and technologies. In *International Multiconference Of Engineers and Computer Scientist*, volume 1, pages 19–21.

[Anati et al. 2013] Anati, I., Gueron, S., Johnson, S. P., and Scarlata, V. R. (2013). Innovative technology for CPU based attestation and sealing. In *2nd Intl Workshop on Hardware and Architectural Support for Security and Privacy*, New York, NY. ACM.

[Anderson et al. 2006] Anderson, R., Bond, M., Clulow, J., and Skorobogatov, S. (2006). Cryptographic processors: A survey. *Proceedings of the IEEE*, 94(2):357–369.

[ARM 2009] ARM, A. (2009). Security technology building a secure system using TrustZone technology (white paper). *ARM Limited*.

[Arthur and Challener 2015] Arthur, W. and Challener, D. (2015). *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Apress Eds.

[Baumann et al. 2015] Baumann, A., Peinado, M., and Hunt, G. (2015). Shielding applications from an untrusted cloud with Haven. *ACM Trans. Comput. Syst.*, 33(3):8:1–8:26.

[Bossuet et al. 2013] Bossuet, L., Grand, M., Gaspar, L., Fischer, V., and Gogniat, G. (2013). Architectures of flexible symmetric key crypto engines: A survey: From hardware coprocessor to multi-crypto-processor system on chip. *ACM Compututer Survey*, 45(4):41:1–41:32.

[Brasser et al. 2017] Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., and Sadeghi, A.-R. (2017). Software grand exposure: SGX cache attacks are practical. *arXiv preprint arXiv:1702.07521*.

[Brekalo et al. 2016] Brekalo, H., Strackx, R., and Piessens, F. (2016). Mitigating password database breaches with Intel SGX. In *1st Workshop on System Software for Trusted Execution*, SysTEX '16, pages 1:1–1:6, New York, NY, USA. ACM.

[Brenner et al. 2017] Brenner, S., Hundt, T., Mazzeo, G., and Kapitza, R. (2017). *Secure Cloud Micro Services Using Intel SGX*, pages 177–191. Springer International Publishing, Neuchâtel, Switzerland.

[Costan and Devadas 2016] Costan, V. and Devadas, S. (2016). Intel SGX explained. Cryptology ePrint Archive, Report 2016/086.

[Cox 2017] Cox, J. (2017). Hackers: We will remotely wipe iPhones unless Apple pays ransom. `https://motherboard.vice.com/en_us/article/hackers-we-will-remotely-wipe-iphones-unless-apple-pays-ransom`. Acessado em 01/04/2017.

[Davenport and Ford 2014] Davenport, S. and Ford, R. (2014). SGX: the good, the bad and the downright ugly. *Virus Bulletin*.

[FBI 2005] FBI (2005). Computer crime survey. `http://www.fbi.gov/publications/ccs2005.pdf`. Acessado em 01/02/2017.

[Greene 2012] Greene, J. (2012). Intel trusted execution technology, white paper. *Online: http://www. intel. com/txt*.

[Hoekstra et al. 2013] Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., and Del Cuvillo, J. (2013). Using innovative instructions to create trustworthy software solutions. In *2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP 2013, pages 11:1–11:1, New York, NY, USA. ACM.

[Intel 2014] Intel (2014). *Intel Software Guard Extensions Programming Reference*.

[Intel 2016a] Intel (2016a). *Intel Software Guard Extensions Developer Guide*. Intel Corporation.

[Intel 2016b] Intel (2016b). *Intel Software Guard Extensions SDK for Linux OS Developer Reference*. Intel Corporation.

[Jain et al. 2016] Jain, P., Desai, S., Kim, S., Shih, M.-W., Lee, J., Choi, C., Shin, Y., Kim, T., Kang, B. B., and Han, D. (2016). OpenSGX: An open platform for SGX research. In *Network and Distributed System Security Symposium*, NDSS 2016, San Diego, CA.

[Karande et al. 2017] Karande, V., Bauman, E., Lin, Z., and Khan, L. (2017). SGX-Log: Securing system logs with SGX. In *2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 19–30, New York, NY, USA. ACM.

[Lesjak et al. 2015] Lesjak, C., Hein, D., and Winter, J. (2015). Hardware-security technologies for industrial IoT: TrustZone and security controller. In *41st Annual Conference of the IEEE Industrial Electronics Society*, IECON 2015, pages 002589–002595.

[Lind et al. 2017] Lind, J., Priebe, C., Muthukumaran, D., O'Keeffe, D., Aublin, P.-L., Kelbert, F., Reiher, T., Goltzsche, D., Eyers, D., Kapitza, R., Fetzer, C., and Pietzuch, P. (2017). Glamdring: Automatic application partitioning for Intel SGX. In *2017 USENIX Annual Technical Conference*, pages 285–298, Santa Clara, CA. USENIX Association.

[McKeen et al. 2013] McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V., and Savagaonkar, U. R. (2013). Innovative instructions and software model for isolated execution. In *2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, New York, NY, USA. ACM.

[Mofrad et al. 2017] Mofrad, M. H., Lee, A., and Gray, S. L. (2017). Leveraging Intel SGX to create a nondisclosure cryptographic library. *arXiv preprint arXiv:1705.04706*.

[Moghimi et al. 2017] Moghimi, A., Irazoqui, G., and Eisenbarth, T. (2017). Cachezoom: How SGX amplifies the power of cache attacks. *arXiv preprint arXiv:1703.06986*.

[Richter et al. 2016] Richter, L., Götzfried, J., and Müller, T. (2016). Isolating operating system components with Intel SGX. In *1st Workshop on System Software for Trusted Execution*, SysTEX '16, pages 8:1–8:6, New York, NY, USA. ACM.

[Schuster et al. 2015] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M. (2015). VC3: Trustworthy data analytics in the cloud using SGX. In *IEEE Symposium on Security and Privacy*, pages 38–54.

[Schwarz et al. 2017] Schwarz, M., Weiser, S., Gruss, D., Maurice, C., and Mangard, S. (2017). Malware guard extension: Using SGX to conceal cache attacks. *arXiv preprint arXiv:1702.08719*.

[Shih et al. 2016] Shih, M.-W., Kumar, M., Kim, T., and Gavrilovska, A. (2016). S-NFV: Securing NFV states by using SGX. In *1st ACM International Workshop on Security in SDN and NFV*, New Orleans, LA.

[TCG 2008] TCG (2008). Trusted Platform Module (TPM) summary. `https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-Summary_04292008.pdf`. Acessado em 02/04/2017.

[TCG 2016] TCG (2016). Trusted Platform Module library - part 1: Architecture. `https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf`. Acessado em 02/04/2017.

[Tian et al. 2017] Tian, H., Zhang, Y., Xing, C., and Yan, S. (2017). SGXKernel: A library operating system optimized for Intel SGX. In *Computing Frontiers Conference*, CF'17, pages 35–44, New York, NY, USA. ACM.

[Tsai et al. 2017] Tsai, C.-C., Porter, D. E., and Vij, M. (2017). Graphene-SGX: A practical library OS for unmodified applications on SGX. In *USENIX Annual Technical Conference*, pages 645–658, Santa Clara, CA. USENIX Association.

[Vacca 2016] Vacca, J. R. (2016). *Cloud Computing Security: Foundations and Challenges*. CRC Press.

[Weafer 2016] Weafer, V. (2016). Report: 2017 threats prediction. Technical report, McAfee Labs. Acessado em 29/03/2017.

Capítulo 3

Segurança de Aplicações Blockchain Além das Criptomoedas

Alexandre Melo Braga, Fundação CPqD (ambraga@cpqd.com.br)
Fernando C. Herédia Marino, Fundação CPqD (fmarino@cpqd.com.br)
Robson Romano dos Santos, Fundação CPqD (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

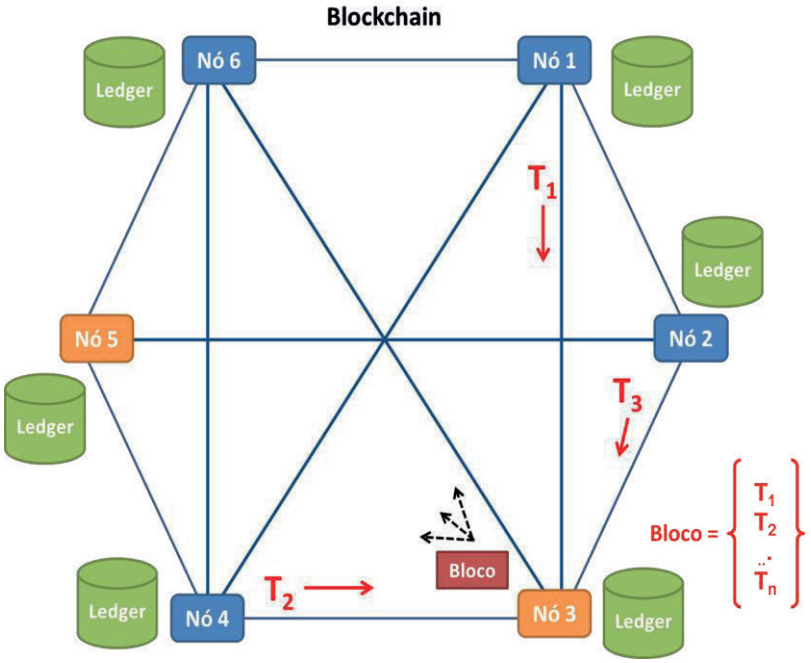


Figura 3.1. Blockchain como uma base de dados distribuída em uma rede P2P.

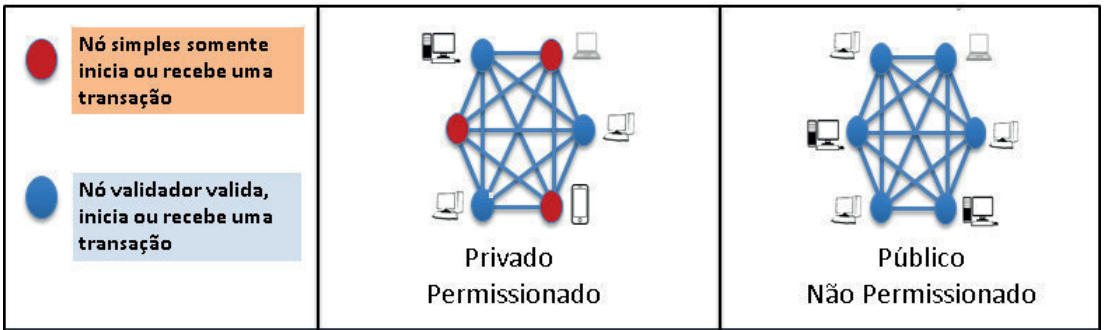
(coleção crescente de registros de transações) distribuída e blockchain para o sistema distribuído formado pela *ledger* distribuída e os nós da rede P2P.

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.



Acesso à rede	Necessita de autorização	Acesso aberto
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 divido 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

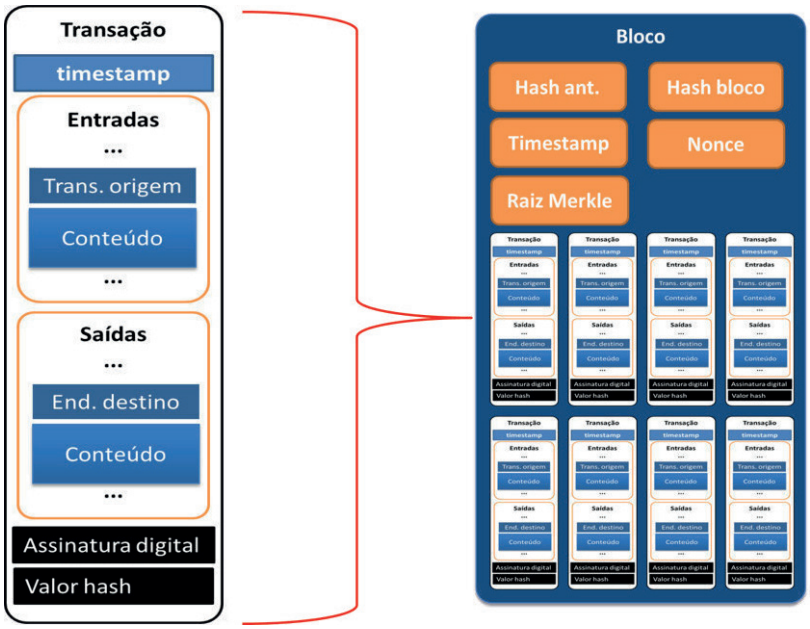


Figura 3.3. A transação e o bloco.

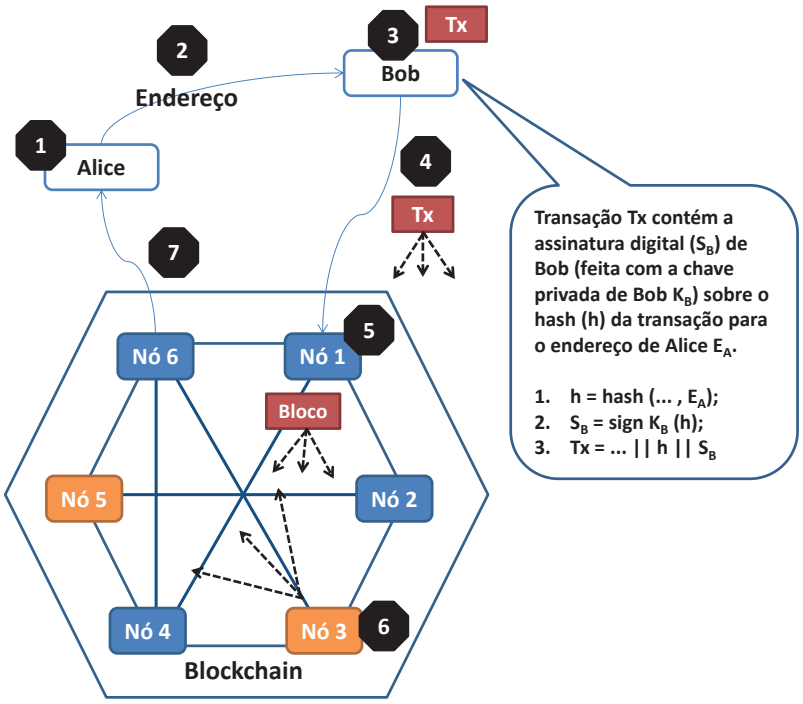


Figura 3.4. Os passos de uma transação.

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.

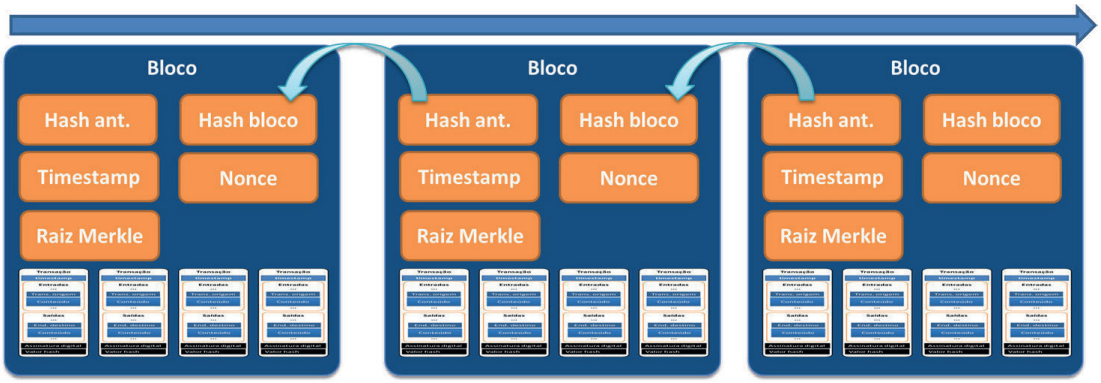


Figura 3.5. A cadeia de blocos.

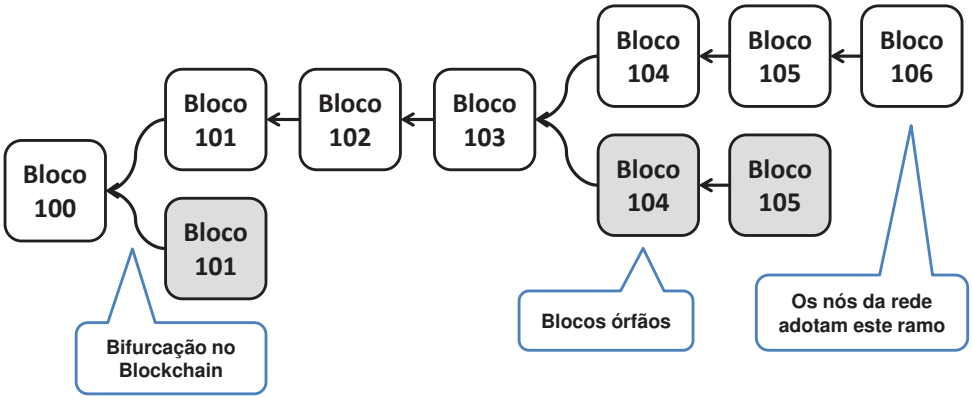


Figura 3.6. A dinâmica da cadeia de blocos.

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 arvore, 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 l*og(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.

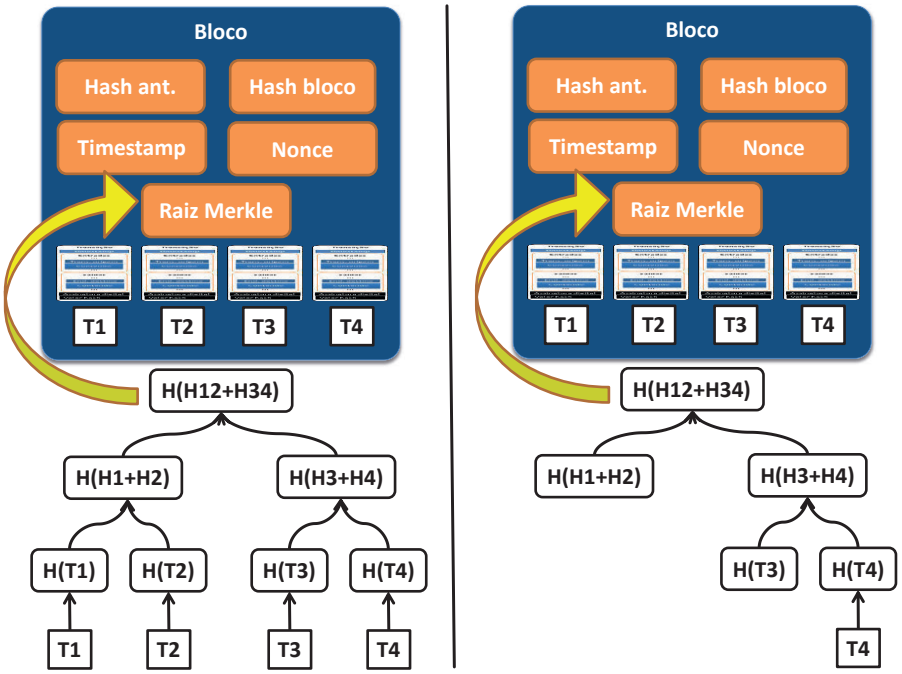


Figura 3.7. Árvore Merkle do bloco (esquerda) e verificação da transação T4 (direita).

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 presenta 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.

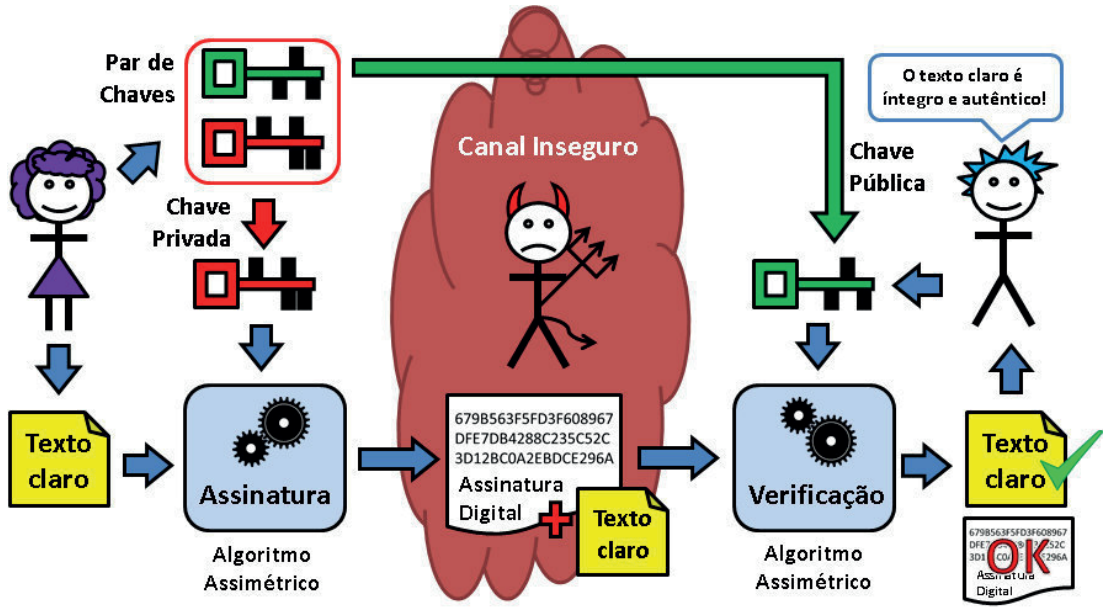


Figura 3.8. Assinatura digital (adaptada de [6]).

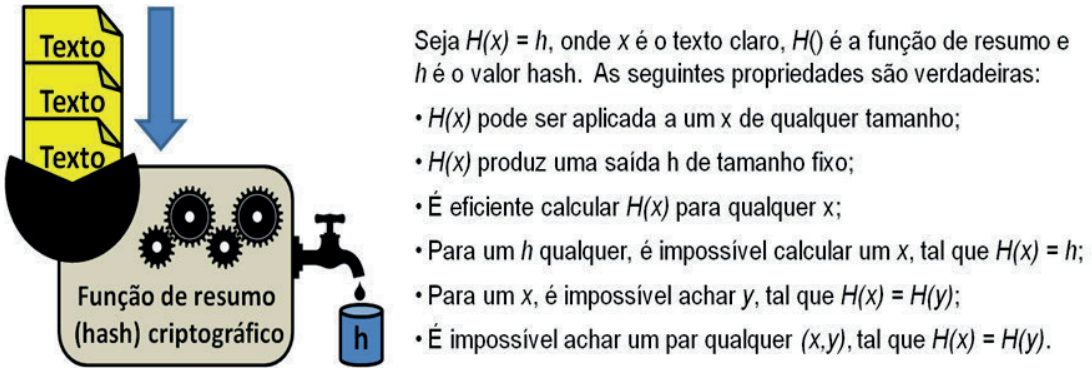


Figura 3.9. Funções de resumo (*hash*) criptográfico (adaptada de [6]**).**

3.2.3. Propriedades técnicas do blockchain

O blockchain tem uma séria 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 do 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 valido, 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.

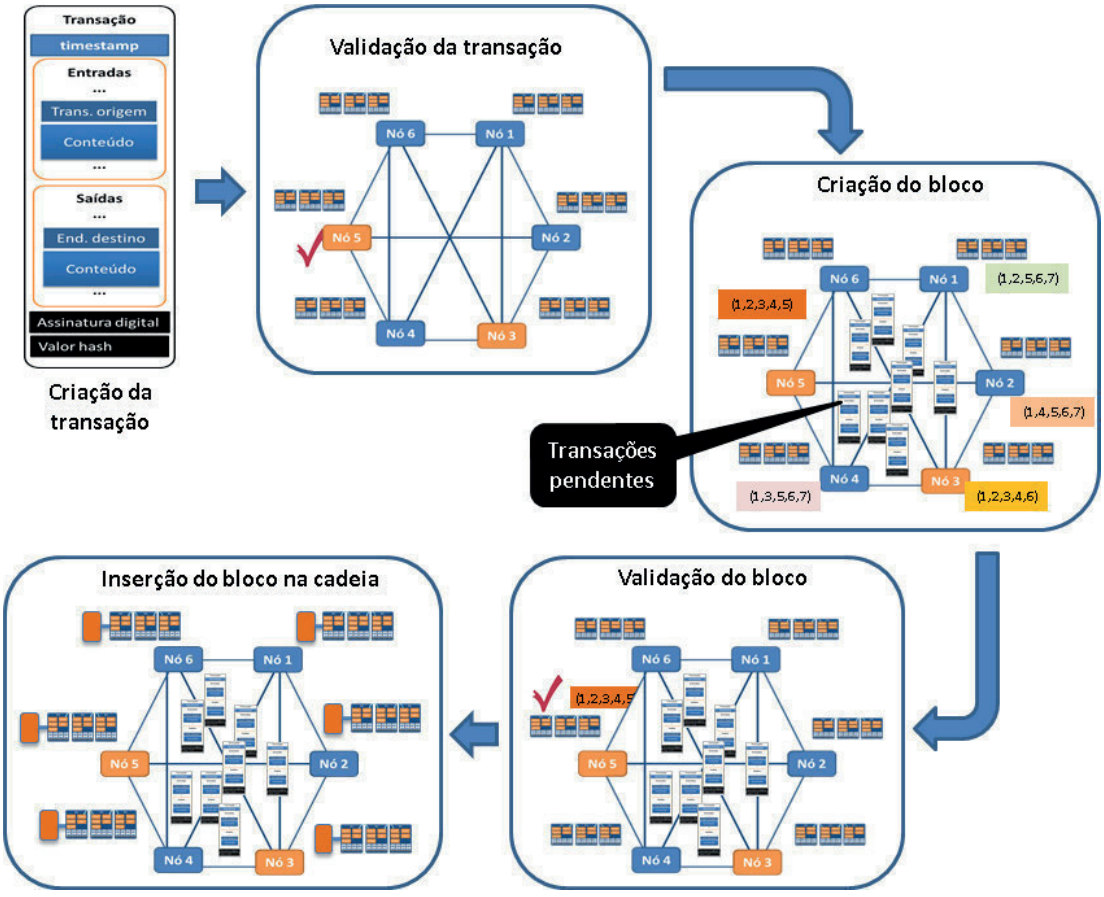


Figura 3.10. Transação, bloco, cadeia, consenso e rede de nós P2P.

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

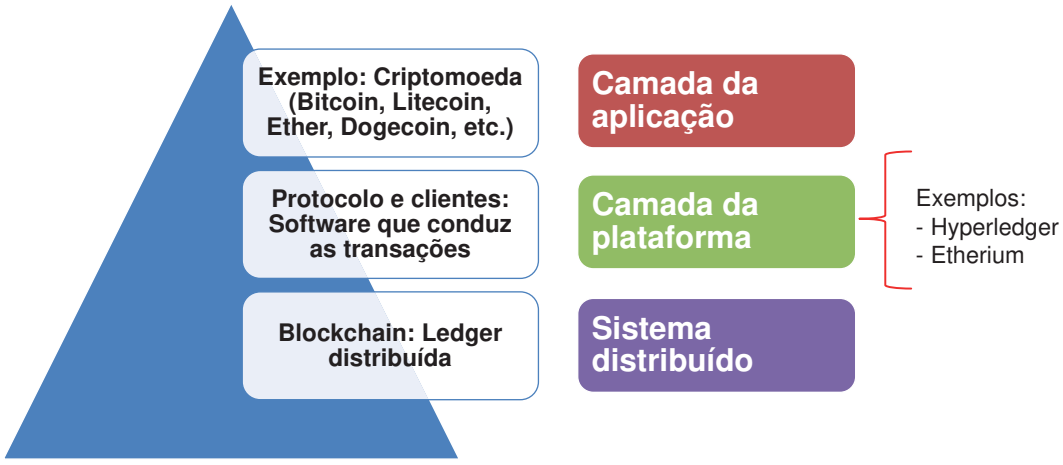


Figura 3.11. Camadas tecnológicas do blockchain.

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

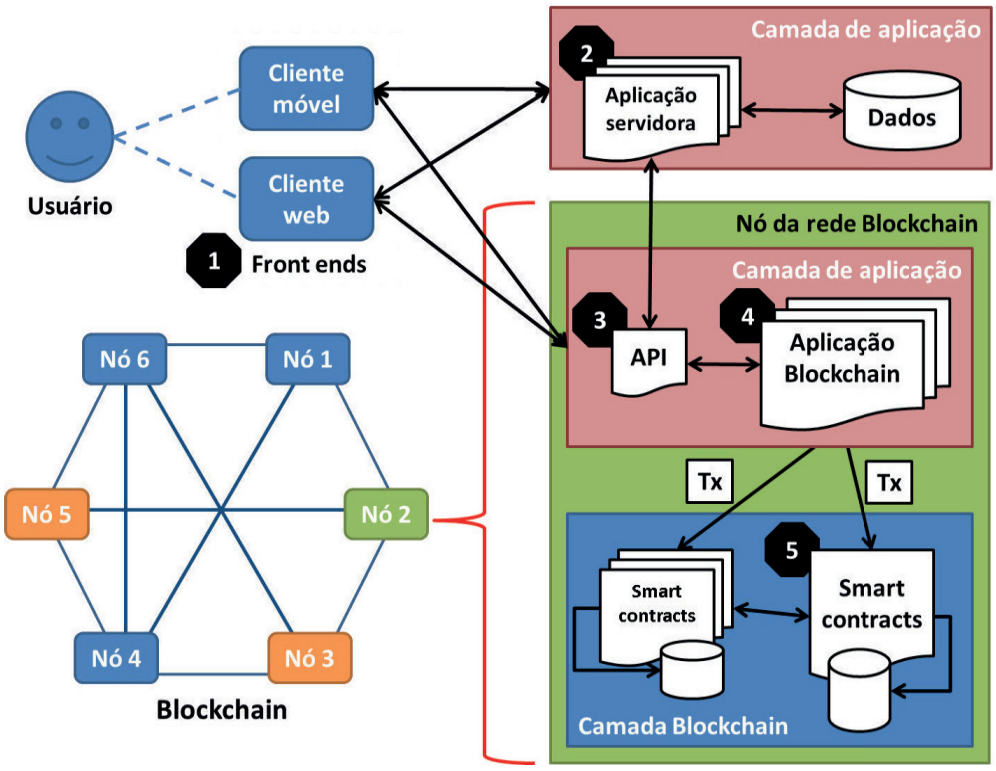


Figura 3.12. Estrutura de uma aplicação blockchain.

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.

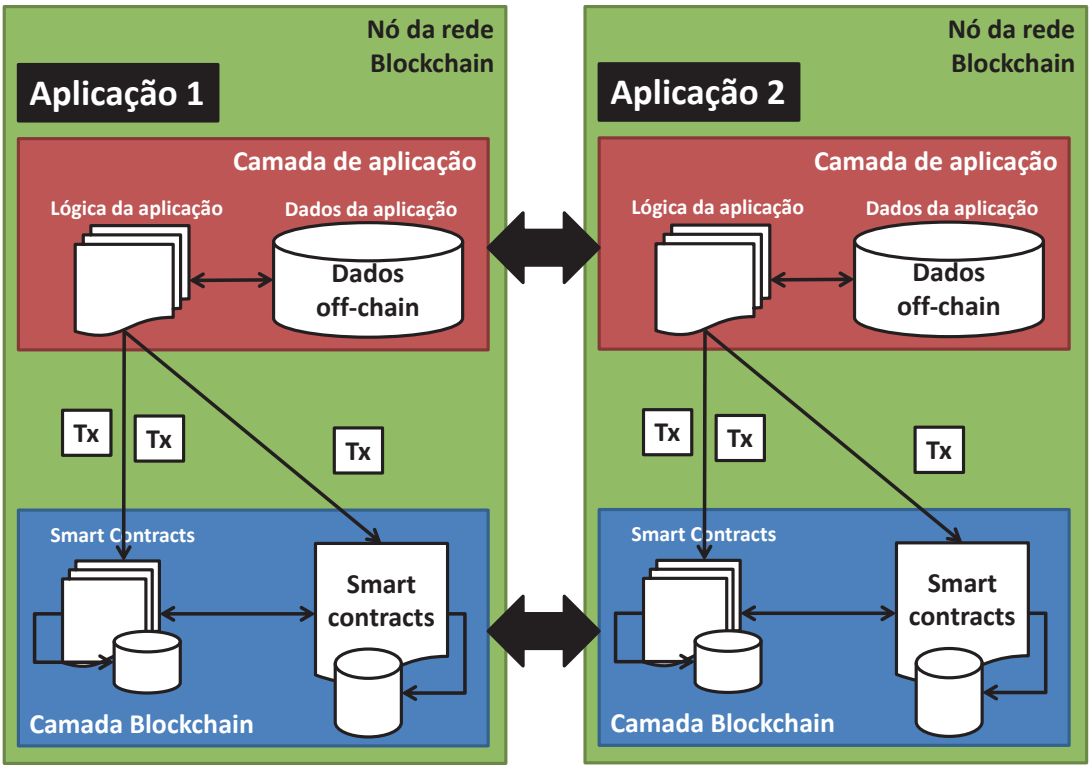


Figura 3.13. Blockchain como um conector de sistemas complexos.

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.

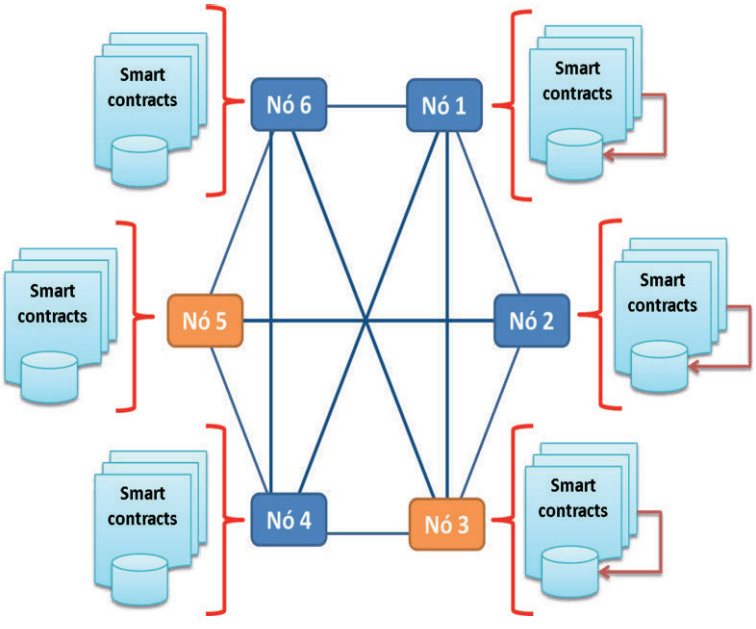


Figura 3.14. Contratos inteligentes (*smart contracts*, *chaincodes*).

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.

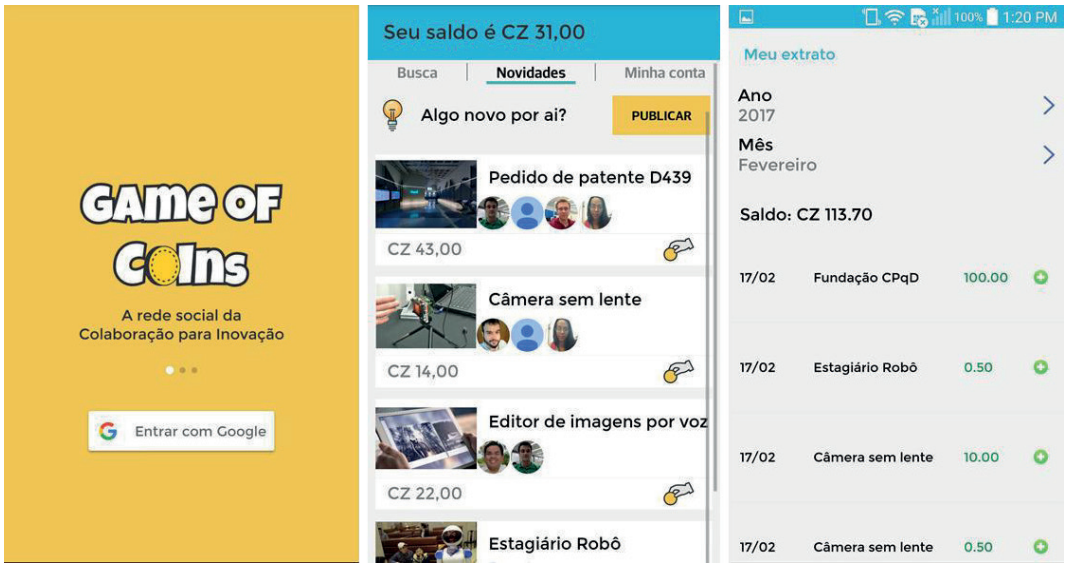


Figura 3.15. Exemplo de um software cliente blockchain (*eWallet*).

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.

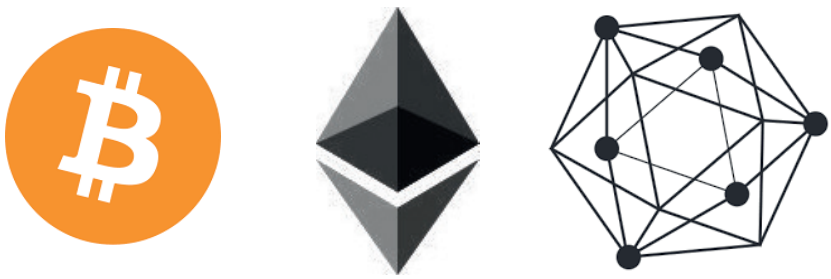


Figura 3.16. Da esquerda para a direita: Bitcoin, Ethereum e Hyperledger.

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 e 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

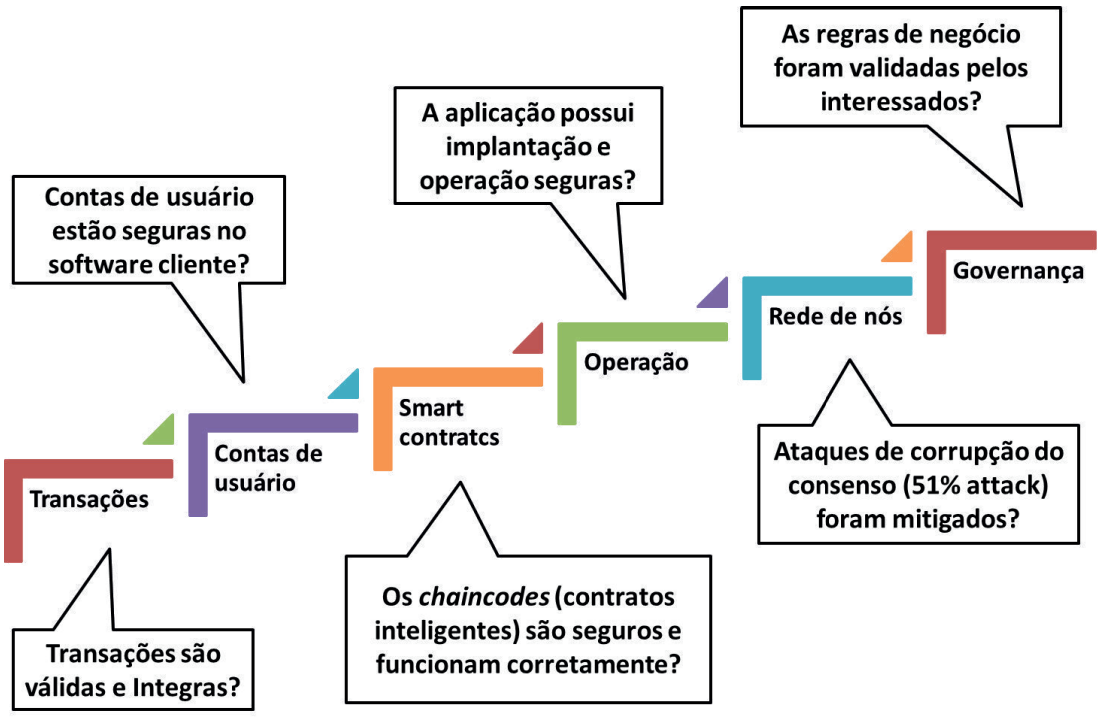


Figura 3.17. Camadas de segurança da tecnologia blockchain e suas aplicações.

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.

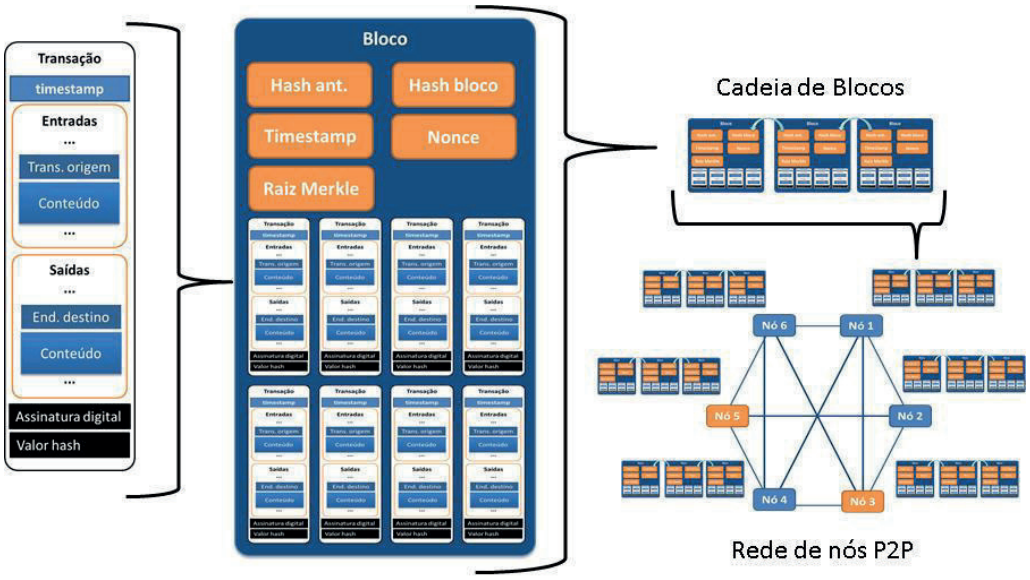


Figura 3.18. Todos os envolvidos na proteção da transação (bloco, cadeia e rede P2P).

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 ás revisões e inspeções. O terceiro, em vermelho, é voltado aos testes de segurança.

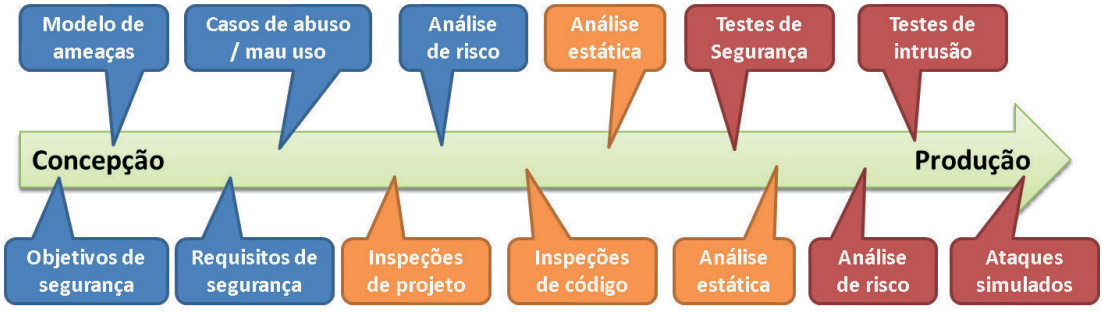


Figura 3.19. Entregáveis de desenvolvimento de software blockchain seguro.

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 nuven (*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 asseguração 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 asseguração é obtida com aderência a normas (de segurança), monitoramento, inspeções e testes contínuos. A asseguração 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 asseguração 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.

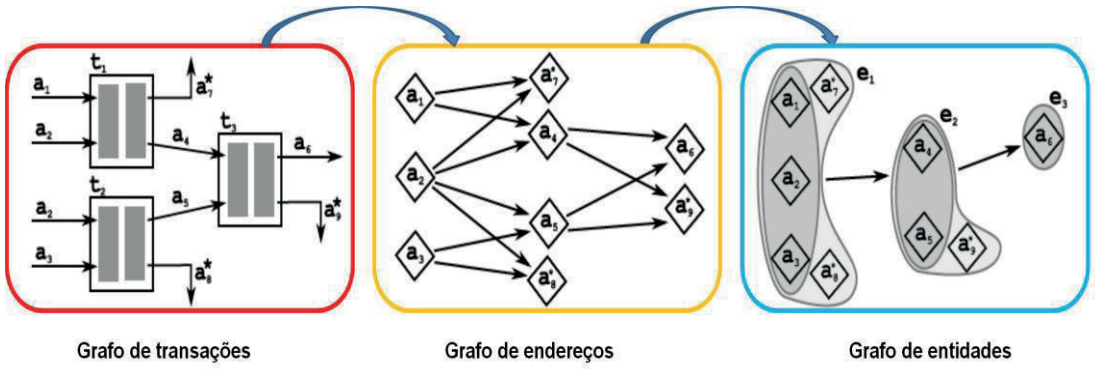


Figura 3.20. Mapeamento de entidades que transacionam Bitcoins (adaptado de [1]).

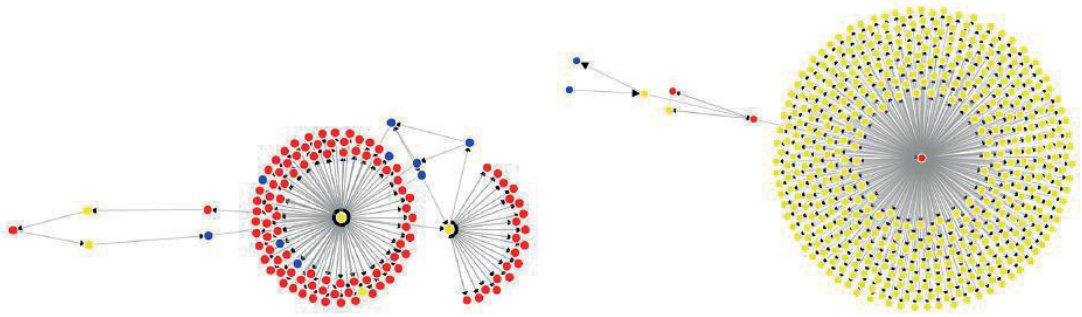


Figura 3.21. Transações que duplicam nonces na assinatura (adaptado de [26]).

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)

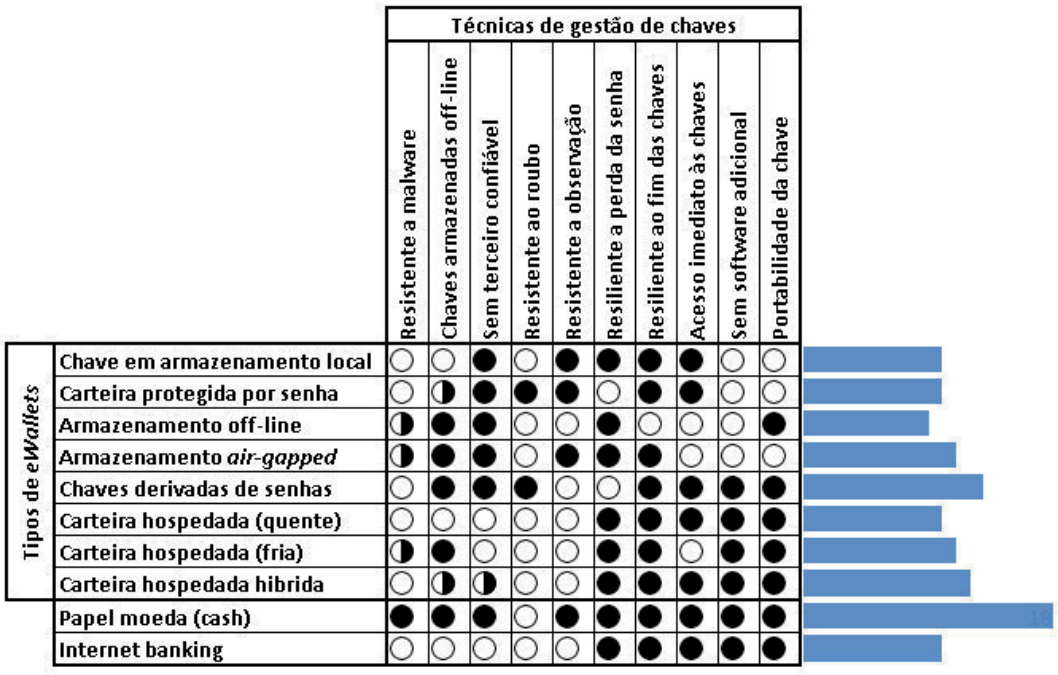


Figura 3.22. Tipos de *eWallets* e como fazem gestão de chaves (adaptada de [18]).

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.

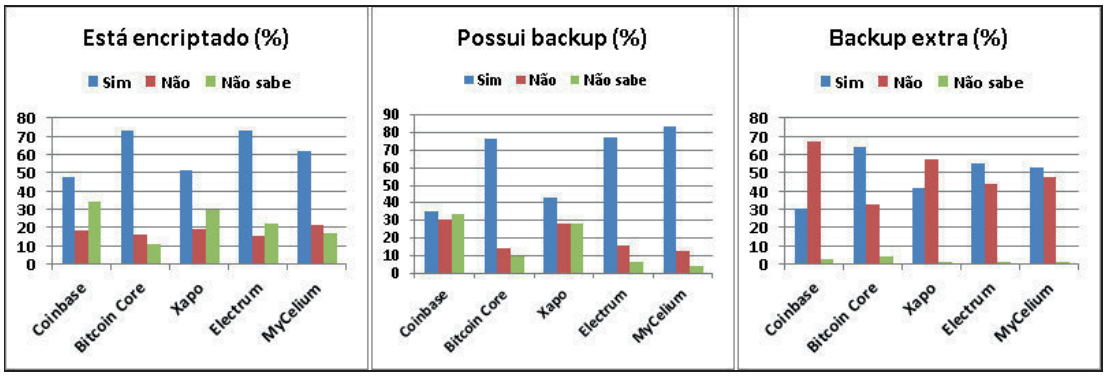


Figura 3.23. Usuários encriptam e fazem backup das suas carteiras (adaptada de [17]).

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].

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

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 acorrer 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 erros 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, o 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 sequencia 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 sequencias 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 sequencias 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 *Gas*	Função *send* para transferência de *ether* para um contrato pode disparar uma exceção de falta de *Gas*, por não ter fundos (*Gas*) 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 (*type casting*)	Capacidade limitada de detecção de erros de conversão de tipo por *casting*. O *casting* estático de tipos simples pode ser detectado na compilação. Já o *casting* 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 *fallback* 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 substitui-lo diretamente. Requer medidas de autodestruição explícitas elaboradas pelo programador.
	Ether perdido na transferência	Na transferência de *ether* 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 *ether* enviado é perdido para sempre.
	Limite de tamanho da pilha	Quando um contrato invoca outro, a pilha de chamadas (*call stack*) do contrato é incrementada de um *frame* (a estrutura de dados da chamada). O limite desta pilha é 1024 *frames*. 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: $C(T_1).C(T_2) \; != \; C(T_2).C(T_1)$. 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. 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 elíticas 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.

[28] SANS/CWE, "TOP 25 Most Dangerous Software Errors," *SANS/CWE*. [Online]. Available: 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/.

Capítulo 4

Uso de Blockchain para Privacidade e Segurança em Internet das Coisas

Vanessa R. L. Chicarino, Emanuel Ferreira Jesus, Célio V. N. de Albuquerque, Antônio A. de A. Rocha

Instituto de Computação (IC), Universidade Federal Fluminense (UFF), Rio de Janeiro.

Abstract

The Internet of Things (IoT) are increasingly a reality today, nevertheless some key challenges still need to be given special attention so that IoT solutions further support the growing demand for connected devices and the services offered. Due to the potential relevance and sensitivity of services, IoT solutions should address the security and privacy concerns surrounding these devices and the data they collect, generate, and process. Recently, the Blockchain technology has gained a lot of attention in IoT solutions. Its main usage scenarios are in the financial domain, where Blockchain creates a world of promising applications and can be leveraged to solve security and privacy issues. However, this emerging technology has a great potential in the most diverse technological areas, and can significantly help achieve the Internet of Things view in different aspects, increasing the capacity of decentralization, facilitating interactions, enabling new transaction models and allowing autonomous coordination of the devices. The goal of this short, is to provide the concepts about the structure and operation of Blockchain and, mainly, to analyze how the use of this technology can be used to provide security and privacy in IoT.

Resumo

Internet das Coisas (IoT) é cada vez mais uma realidade atual, embora alguns desafios-chave carecem ainda de uma atenção especial para que soluções de IoT dêem um suporte ainda maior para a crescente demanda por dispositivos conectados e os serviços oferecidos. Devido a possível relevância e sensibilidade dos serviços, as soluções de IoT devem abordar as preocupações de segurança e privacidade em torno desses dispositivos e dos dados que eles coletam, geram e processam. Recentemente, a tecnologia Blockchain ganhou muita atenção em soluções de Internet das Coisas. Seus principais cenários de uso estão no domínio financeiro, onde a Blockchain cria um mundo de promissoras aplicações e pode ser aproveitado para resolver os problemas de segurança e privacidade.

Porém, essa emergente tecnologia tem um grande potencial nas mais diversas áreas tecnológicas, e pode ajudar significativamente a alcançar a visão da Internet das Coisas em diferentes aspectos, aumentando a capacidade de descentralização, facilitando interações, possibilitando novos modelos de transações e permitindo a coordenação autônoma dos dispositivos. Este minicurso, tem como objetivo fornecer conceitos sobre a estrutura e funcionamento da Blockchain e, principalmente, de analisar como o uso desta tecnologia pode ser usada para prover segurança e privacidade em IoT.

4.1. Introdução e motivação

Internet das Coisas (IoT) e Blockchain são consideradas tecnologias emergentes na atualidade. Ao mesmo tempo que transformam conceitos e criam novas possibilidades, cada uma em seus respectivos cenários, existe a oportunidade de criar aplicações que podem compartilhar as características intrínsecas de ambos, explorando como a IoT pode se beneficiar da natureza descentralizada da Blockchain.

Blockchain (também conhecido como "o protocolo da confiança") é um conceito que visa a descentralização como medida de segurança. São bases de registros e dados distribuídos e compartilhados que possuem a função de criar um índice global para todas as transações que ocorrem em uma determinada rede. Funciona como um livro-razão, só que de forma pública, compartilhada e universal, que cria consenso e confiança na comunicação direta entre duas partes, ou seja, sem o intermédio de terceiros. Está constantemente crescendo à medida que novos blocos completos são adicionados a ela por um novo conjunto de registros. A cadeia de blocos também pode ser usada para comunicações em cadeia de fornecimento, contratos inteligentes, gerenciamento de identidade digital e em uma série de outras aplicações [Pilkington, 2016].

A Internet das Coisas é um termo abrangente referente aos esforços em curso para conectar uma grande variedade de coisas físicas às redes de comunicação. Atualmente não apenas computadores convencionais estão conectados à Internet, como também uma grande heterogeneidade de equipamentos, tais como TVs, laptops, geladeira, fogão, eletrodomésticos, automóveis, smartphones, entre outros. Nesse novo cenário, a pluralidade é crescente e previsões indicam que mais de 50 bilhões de dispositivos estarão conectados até 2020 [Evans, 2011]. Dentro do domínio IoT existem vários tipos de aplicações, como por exemplo: cidades inteligentes (smart cities); saúde (smart healthcare); casas inteligentes (smart home) entre outras.

Ao mesmo tempo que a IoT poderá nos proporcionar benefícios valiosos, ela também aumentará os nossos riscos de exposição a diversas ameaças de segurança e privacidade, algumas dessas ameaças são novas e bem particulares desta tecnologia. Antes do advento da Internet das Coisas, a maioria das ameaças de segurança estavam relacionadas ao vazamento de informações e a negação de serviço. Com a IoT, as ameaças à segurança vão muito além do roubo de informações ou da impossibilidade de uso de determinados serviços. Essas ameaças podem agora estar potencialmente relacionadas com as vidas reais, inclusive de segurança física.

Soluções de segurança e privacidade devem ser implementadas conforme as características de dispositivos IoT heterogêneos. Há uma demanda por soluções de segurança que sejam capazes de fornecer níveis de segurança equivalentes para vários tipos de dis-

positivos. IoT trouxe consigo um aumento da quantidade de informações pessoais que serão entregues e compartilhadas entre os dispositivos conectados. Assim, embora não seja uma demanda nova ou exclusiva deste novo cenário, a privacidade é um elemento importante que, em virtude de suas especificidades, demanda mecanismos capazes de auditar e controlar acesso nestes ambientes.

Neste contexto que Blockchain também se insere, pois essa tecnologia pode ser usada para Autenticar, Autorizar e Auditar os dados gerados pelos dispositivos. Além disso, em virtude de sua natureza descentralizada, elimina a necessidade de confiança em terceiros e não possui um ponto único de falha.

Este minicurso tem por objetivo familiarizar novos interessados, bem como atualizar os leitores que possuem algum conhecimento anterior, a esta nova e promissora tecnologia chamada Blockchain. Isso inclui as recentes aplicações em segurança e privacidade, e como o seu uso pode alavancar a IoT. A abordagem oferecida neste minicurso será orientada a estudos de caso em que o uso de Blockchain já é uma realidade ou que tem sido movido esforços nesta direção. Estudos de caso, como por exemplo o uso de Blockchain para prover controle de acesso e anonimidade em sistemas de armazenamento de dados distribuídos ou o seu uso para controle de operações financeiras, nortearão este minicurso.

O minicurso está estruturado em cinco seções, sendo esta a primeira. A Seção 4.2 irá apresentar os fundamentos teóricos básicos para a compreensão da solução proposta, como os princípios fundamentais de segurança e os conceitos de IoT; criptografia e funções hash e redes peer-to-peer (P2P). A Seção 4.3 apresentará todos os mecanismos de funcionamento da tecnologia Blockchain e descreverá os Contratos inteligentes, uma das aplicações da Blockchain. A Seção 4.4 descreve alguns casos de uso do Blockchain para prover segurança e privacidade em IoT e apresenta alguns casos reais de ataques a rede Blockchain, apresentando os resultados e análises de simulações realizadas. Finalmente, a Seção 4.5 apresenta as considerações finais e questões em aberto, apresentando referências para que o leitor possa complementar seus estudos.

4.2. Fundamentação teórica básica

Esta seção visa ambientar os leitores com as informações mais básicas para entender o que é a Blockchain. Para isso, serão apresentados: os princípios de segurança (Confidencialidade, Integridade, Disponibilidade, Privacidade, Auditoria, Autenticação e Não Repúdio) relacionados à Blockchain, recentes pesquisas a respeito de Internet das Coisas, abordando as classificações e taxonomias propostas em IoT.

O Bitcoin [Bitcoin, 2009] é um conjunto de conceitos e tecnologias que formam a base de um ecossistema de dinheiro digital. Seus usuários comunicam-se através da Internet utilizando uma rede Par-a-Par (ou P2P, do termo em inglês *Peer-to-Peer*) própria, mas outras formas de rede também podem ser usadas. Sua implementação está disponível como software de código aberto, e pode ser executada em diversos tipos de dispositivos, o que torna a tecnologia de fácil acesso. Será explicado o funcionamento desta rede P2P e os conceitos de funções hash e criptografia de chaves públicas utilizados.

4.2.1. Internet das Coisas

A Internet das Coisas (ou IoT, do termo em inglês *Internet of Things*) abrange o processamento de dados e a comunicação entre dispositivos de plataformas e capacidades diferentes de forma autonômica, sem intervenção humana. Nas últimas décadas esse termo despontou como uma evolução da internet e um novo paradigma tecnológico, social, cultural e digital.

A IoT é considerada uma extensão da internet atual, pois proporciona aos objetos do dia-a-dia (eletrodomésticos, meios de transporte e até acessórios, como por exemplo, óculos e relógios) com capacidade computacional e de comunicação a se conectarem a Internet. A conexão com a rede mundial de computadores viabilizará o controle remoto dos objetos e permitirá que os próprios objetos sejam acessados como provedores de serviços, tornando-os objetos inteligentes ou *smart objects*. Os objetos inteligentes possuem capacidade de comunicação e processamento aliados a sensores.

O primeiro dispositivo IoT foi apresentado em 1990 na *INTEROP '89 Conference* por John Romkey que criou uma torradeira que poderia ser ligada e desligada pela Internet, conectando a torradeira a um computador com rede TCP / IP. Em setembro de 1999, Kevin Ashton [Ashton, 2011], cofundador e diretor executivo do Auto - ID Center, proferiu uma palestra para a Procter & Gamble, apresentando a ideia de utilizar etiquetas eletrônicas nos produtos da empresa, para facilitar a logística da cadeia de produção, através de identificadores de rádio frequência (RFID, em inglês), para chamar a atenção dos executivos, ele colocou no título da apresentação a expressão *"Internet of Things"*, sendo considerado o criador desse termo, ao descrever que os objetos do mundo físico poderiam se conectar à internet, criando um mundo mais inteligente.

A partir de 2005, a discussão sobre a Internet das Coisas se generalizou, começou a ganhar a atenção dos governos e aparecer relacionada a questões de privacidade e segurança de dados. Foi neste ano que a Internet das Coisas se tornou a pauta do *International Telecommunication Union* (ITU), agência das Nações Unidas para as tecnologias da informação e da comunicação, que publica anualmente um relatório sobre tecnologias emergentes. Assim, depois da banda larga e da internet móvel, a Internet das Coisas ganhou a atenção do órgão e passou a figurar como o "próximo passo da tecnologia em comunicações *'always on'*, que prometem um mundo de dispositivos interconectados em rede" [Peña-López et al., 2005].

Entre os anos de 2008 e 2010, devido ao amadurecimento das Redes de Sensores Sem Fio (RSSF) (do inglês *Wireless Sensor Networks* - WSN), que trazem avanços na automação residencial e industrial, e técnicas para explorar as diferentes limitações dos dispositivos, como por exemplo, memória, energia, escalabilidade e robustez da rede, o termo Internet das Coisas ganhou popularidade rapidamente. Nesse período foi publicado o livro *The Internet of Things* por Rob Van Kranemburg, que aborda esse termo sob um novo paradigma em que os objetos produzem informação. Esse livro é uma das grandes referências teóricas sobre Internet das Coisas [Cui, 2016]. Em 2011 o termo Internet das Coisas foi adicionado, como tecnologia emergente, ao *Gartner Hype Cycle* [Fenn and LeHong, 2011], que fornece uma representação gráfica da maturidade e adoção de tecnologias e aplicativos e fornece uma visão de como uma tecnologia ou aplicação irá evoluir ao longo do tempo. A Figura 4.1 apresenta o gráfico deste ano, onde as

Plataformas IoT permanecem como tecnologia promissora.

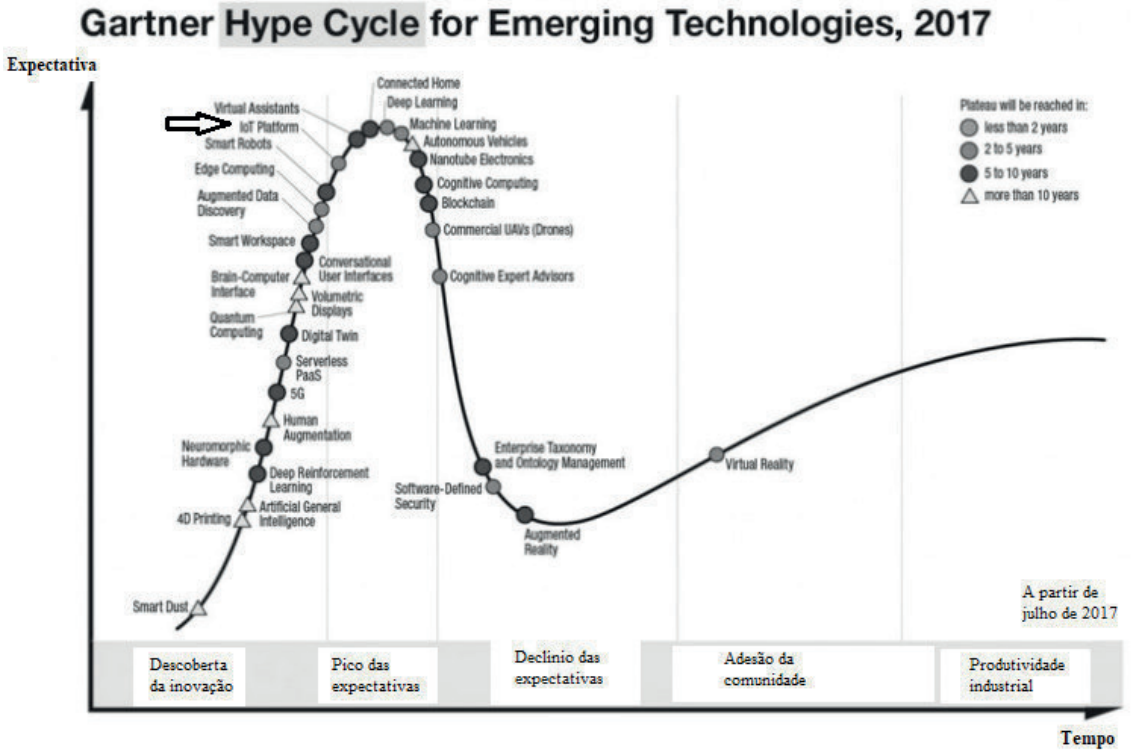


Figura 4.1. Gartner Hype Cycle para Tecnologias Emergentes Fonte: https://http://www.gartner.com acessado em: 10/09/2017

Atualmente não há um único conceito de Internet das Coisas definido na literatura, porém vários autores e instituições contribuíram para a construção da visão de IoT. Uma das referências mais citadas em [Atzori et al., 2010], o autor descreve IoT como sendo uma variedade de coisas ou objetos - como tags de identificação de radiofrequência (RFID), sensores, atuadores, telefones celulares, entre outros em torno de nós que interagem uns com os outros e cooperam com seus vizinhos, através de esquemas de endereçamento únicos, com a finalidade de alcançar metas comuns, sendo o resultado da convergência de três paradigmas: orientado para a internet (middleware), orientado a coisas (sensores e atuadores) e orientado à semântica (a representação e o armazenamento das informações trocadas), porém a utilidade do IoT pode ser desencadeada somente em um domínio de aplicação onde os três paradigmas se cruzam.

Algumas instituições relevantes enfatizaram o conceito de que o IoT deve se concentrar principalmente nas "Coisas"e que o caminho para sua implantação total deve começar a partir do aumento na inteligência das coisas. Algumas definições na literatura, derivam dessa visão, uma delas, proposta pelo Grupo de pesquisa europeu em IoT (European Research Projects on the Internet of Things - IERC) apresenta IoT como "Uma infra-estrutura de rede global e dinâmica com capacidades de autoconfiguração baseadas em protocolos de comunicação padronizados e interoperáveis onde as "coisas"físicas e virtuais têm identidades, atributos físicos e personalidades virtuais e usam interfaces inteligentes, sendo integradas perfeitamente na rede de informações" [Guillemin et al., 2009].

O IERC está ativamente envolvido no Grupo de Estudos, que lidera o trabalho da União Internacional de Telecomunicações (International Telecommunications Union - ITU) sobre padrões para redes de próxima geração (NGN) que apresentou a seguinte definição [Strategy and Unit, 2005]: Internet of things (IoT) é uma infra-estrutura global de informação, em constante evolução para a sociedade, permitindo serviços avançados, interligando fisicamente e virtualmente as "coisas"providas de um certo grau de tecnologia de comunicação e informação interoperáveis existentes. A IoT fará pleno uso das "coisas"para oferecer serviços a todos os tipos de aplicações, através da exploração da capacidades de identificação, captura de dados, processamento e comunicação, garantindo simultaneamente que os requisitos de segurança e privacidade sejam cumpridos. Essa perspectiva faz com que IoT seja percebida como uma visão com implicações tecnológicas e sociais.

Em [Recommendation, 2012] foi proposto o modelo de referência IoT pelo Setor de Normalização das Telecomunicações (Telecommunication Standardization Sector (ITU-T)), como visto na Figura 4.2, composto por quatro camadas:

- **camada de aplicação:** responsável por prover os serviços para os clientes;
- **camada de suporte a aplicação e ao serviço:** consiste em suporte genérico e suporte específico. Suporte genérico é aquele em que os recursos de suporte são comuns que podem ser usados por diferentes aplicações de IoT, como processamento ou armazenamento de dados. Suporte específico é aquele que possui recursos particulares que atendem aos requisitos de aplicações específicas, eles podem consistir em vários grupos com recursos detalhados, a fim de fornecer diferentes funções de suporte.
- **camada de rede:** responsável por funções relevantes de controle de conectividade de rede, tais como funções de controle de recursos de acesso e transporte, gerenciamento de mobilidade ou autenticação, autorização e contabilidade, fornecimento de conectividade para o transporte do serviço IoT e informações de dados específicos da aplicação, bem como o transporte de informações de controle e gerenciamento relacionadas ao IoT.
- **camada de dispositivos:** Os recursos dessa camada podem ser logicamente categorizados em recursos do dispositivo e recursos do *gateway*:
  - Recursos do dispositivo incluem: Interação direta com a rede de comunicação sendo os dispositivos capazes de coletar e fazer upload de informações diretamente, sem usar recursos do *gateway*, para a rede de comunicação e podendo receber diretamente informações (por exemplo, comandos) da rede de comunicação; Interação indireta com a rede de comunicação, onde os dispositivos são capazes de coletar e fazer o upload de informações para a rede de comunicação, através de recursos do gateway; Rede ad hoc em que os dispositivos podem ser capazes de construir redes de forma ad hoc em alguns cenários que precisam de escalabilidade aumentada e implantação rápida; *Sleeping and waking-up* onde os recursos do dispositivo podem utilizar mecanismos de "dormir"e "acordar"para economizar energia.

– Os recursos do *Gateway* incluem: Suporte a várias interfaces, por exemplo, na camada de dispositivos, os recursos do *gateway* suportam dispositivos conectados através de diferentes tipos de tecnologias com ou sem fio, ZigBee, Bluetooth ou Wi-Fi e na camada de rede, os recursos do *gateway* podem se comunicar através de várias tecnologias, como rede telefônica comutada (PSTN), redes de segunda geração ou de terceira geração (2G ou 3G), Ethernet ou assinante digital linhas (DSL); Conversão de protocolo, quando as comunicações na camada do dispositivo usam diferentes protocolos, por exemplo, protocolos de tecnologia ZigBee e protocolos de tecnologia Bluetooth e quando comunicações envolvendo camada de dispositivo e camada de rede usam protocolos diferentes, por exemplo, um protocolo de tecnologia ZigBee na camada de dispositivo e um protocolo de tecnologia 3G na camada de rede.

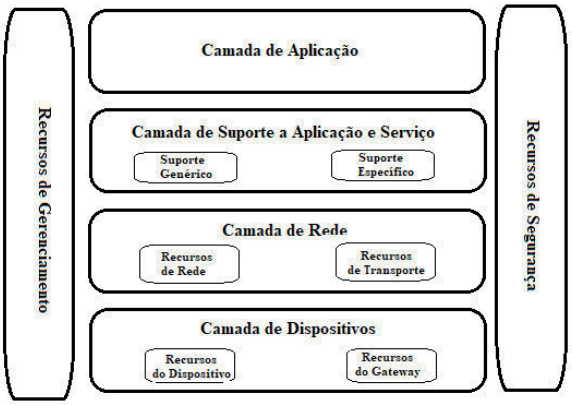


Figura 4.2. Modelo de Referência ITU-T para IoT. Adaptado de: Recommendation ITU-T Y.2060

Esse modelo inclui Recursos de Gerenciamento e Recursos de Segurança associados às quatro camadas. Os Recursos de Gerenciamento cobrem as classes tradicionais de falhas, configurações, contabilidade, desempenho e segurança (FCAPS), como gerenciamento de falhas, gerenciamento de configurações, gerenciamento de contabilidade, gerenciamento de desempenho e gerenciamento de segurança, também são divididos em recursos genéricos e específicos:

- Recursos genéricos de gerenciamento na IoT incluem: gerenciamento de dispositivos, como ativação e desativação remota de dispositivos, diagnóstico, atualização de firmware e/ou software, gerenciamento de status de funcionamento do dispositivo; gestão de topologia de rede local; gerenciamento de tráfego e congestionamento, como a detecção de condições de transbordamento de rede e a implementação de reserva de recursos para fluxos de dados críticos de tempo e/ou vida.
- Recursos específicos de gerenciamento na IoT estão intimamente associadas aos

requisitos específicos da aplicação, por exemplo, requisitos de monitoramento da linha de transmissão de energia da rede inteligente.

De forma similar, também existem dois tipos de recursos de segurança: recursos genéricos de segurança e recursos específicos de segurança:

- Recursos genéricos de segurança são independentes dos aplicativos. Eles incluem na camada de aplicação: autorização, autenticação, confidencialidade e proteção de integridade dos dados da aplicação, proteção de privacidade, auditoria de segurança e antivírus. Na camada de rede incluem: autorização, autenticação, confidencialidade dos dados de uso e da sinalização e proteção da integridade de sinalização. Na camada do dispositivo: autenticação, autorização, validação da integridade do dispositivo, controle de acesso, confidencialidade de dados e proteção de integridade.
- Recursos específicos de segurança estão intimamente associados aos requisitos específicos da aplicação, por exemplo, aplicação de pagamento com mobilidade.

As aplicações de Internet das Coisas são inúmeras e diversas, e permeiam praticamente a vida diária das pessoas, das empresas e sociedade como um todo, transformando o mundo em *smart world* que permite que a computação se torne "invisível" aos olhos do usuário, por meio da relação entre homem e máquina, tornando um mundo mais eficiente e eficaz [Gubbi et al., 2013]. A Figura 4.3 a seguir mostra um panorama da atuação da internet das coisas:

- **Produtos Inteligentes** - Bens adquiridos pelos consumidores, tais como *smartphones, smart house, smart car, smart TV e wearables.*
- **Saúde Inteligente *(eHealth)*** - Fitness, bioeletrônica e cuidados com a saúde. Por exemplo: monitoramento e controle da frequência cardíaca durante os exercícios; Monitoramento das condições dos pacientes em hospitais e em casas de idosos.
- **Transporte Inteligente** -Notificação das condições de tráfego, controle inteligente de rotas, monitoramento remoto do veículo, coordenação das rodovias e integração inteligente de plataformas de transporte.
- **Distribuição Inteligente de Energia *(smart grid)*** - Acompanhamento de instalações de energia, subestações inteligentes, distribuição de energia automática e medições remotas de relógios residenciais.
- **Logística** - *Smart e-commerce*, rastreabilidade, gerenciamento na distribuição e inventário.
- **Indústria Inteligente** - Economia de energia, controle da poluição, segurança na manufatura, monitoramento do ciclo de vida dos produtos, rastreamento de produtos manufaturados na cadeia de abastecimento, monitoramento de condições ambientais e controle de processos de produção.

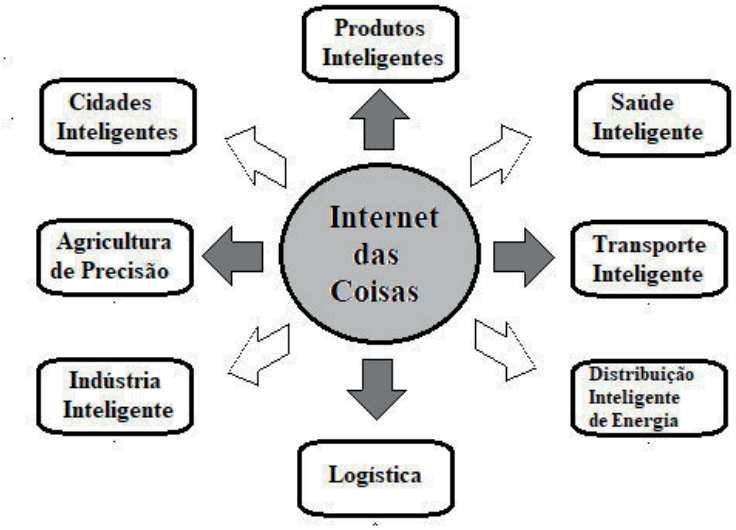


Figura 4.3. Aplicações de IoT

- **Agricultura de Precisão** - Segurança e rastreabilidade de produtos agrícolas, gerenciamento de qualidade, monitoramento ambiental para produção e cultivo, gerenciamento no processo de produção, utilização de recursos para a agricultura.
- **Cidades Inteligentes** - Monitoramento estrutural: monitoramento de vibrações e condições dos materiais em edifícios, pontes e monumentos históricos. Energia elétrica: iluminação inteligente e adaptável conforme a rua. Segurança: monitoramento por meio de vídeo digital, gerenciamento de controle de incêndio e sistemas de anúncio público. Transporte: estradas inteligentes com avisos, mensagens e desvios de acordo com as condições climáticas e eventos inesperados como acidentes ou engarrafamentos. Estacionamento: monitoramento em tempo real da disponibilidade de espaços de estacionamento, sendo possível identificar e reservar vagas disponíveis. Gestão de resíduos: detecção de níveis de lixo em recipientes para otimizar a rota de coleta de lixo.

4.2.2. Princípios Fundamentais de Segurança

Segurança e privacidade são princípios basilares de qualquer sistema de informação. Nos referimos a segurança como a combinação de Integridade, Disponibilidade e Confidencialidade. Normalmente é possível obter segurança usando uma combinação de autenticação, autorização e identificação. Esses conceitos são definidos a seguir [Stallings, 1995]:

- **Integridade:** Certeza que uma informação não foi alterada, exceto por quem tem

o direito de realizar estas alterações. A integridade dos dados é a garantia de que os dados não foram manipulados, estão corretos. No contexto da Blockchain é a garantia de que os dados que constam nas transações não podem ser modificados intencionalmente ou por eventos fortuitos, como surtos de energia ou erros na propagação dos dados. Mecanismos criptográficos de verificação de integridade são comumente utilizados para sua confirmação.

- **Disponibilidade:** Garante que os usuários de um determinado sistema conseguirão utilizá-lo sempre que for necessário. Em outras palavras, os serviço estará sempre ativo quando solicitado por um usuário legítimo. Isso requer que tanto a infraestrutura de comunicação quanto as bases de dados possam ser utilizadas. A Blockchain alcança este objetivos ao permitir que os usuários estabeleçam conexão com vários usuários e ao manter os blocos de maneira descentralizada com várias cópias dos blocos na rede.

- **Confidencialidade:** É a garantia de que a informação não será obtida por pessoas não autorizadas. Isto é, apenas aqueles com os direitos e privilégios necessários serão capazes de acessar a informação, esteja ela armazenada, em processamento ou em trânsito. Na rede Bitcoin, para garantir este princípio são utilizados mecanismos de pseudo anonimização do usuário, como uso dos endereços Bitcoin. Os endereços Bitcoin são resumos criptográficos das chaves públicas.

- **Autenticação, Autorização e Auditoria:** Busca verificar a identidade de quem realiza uma determinada função em um sistema, verificar que direitos esse usuário possui e armazenar informações de uso desse usuário. A estrutura da Blockchain é totalmente desenvolvida para garantir estas três funções, pois somente os usuários que possuem as chaves privadas podem realizar transações, e todas as transações são públicas e auditáveis.

- **Não Repúdio:** Garantia que a pessoa não negue ter feito uma determinada ação em um sistema. O não repúdio fornece provas de que um usuário realizou uma determinada ação, como transferir dinheiro, autorizar uma compra, ou enviar uma mensagem. Como todas as transações são assinadas, um usuário não pode negar que a realizou.

A privacidade pode ser definida como o direito que um indivíduo tem em compartilhar suas informações. Os usuários do Bitcoin usam um pseudônimo (endereço) para realizar suas transações. Normalmente cada usuário possui centenas de endereços. Uma transação pode ser vista como uma cadeia de assinaturas que comprovam a posse e a transferência de valores, de maneira auditável. Assim uma das preocupações é que essas transações possam revelar informações do usuário que vão além de simplesmente uma identificação, como hábitos de compra e locais frequentados do usuário.

O conceito de privacidade em Blockchain consiste em manter o anonimato e a desvinculação de transações. O anonimato de transações exige que não seja possível vincular uma transação particular a um usuário, para isto, o usuário utiliza um endereço diferente a cada nova transação. A desvinculação das transações exige que duas transações do mesmo indivíduo não possam ser vinculadas como tal.

4.2.3. Funções Hash e Criptografia

Toda a posse de recursos e transações na rede são feitas utilizando-se o conceito de chaves e assinaturas digitais. As chaves usadas são geradas aplicando o conceito de criptografia de chaves públicas. São geradas um par de chaves, uma pública que pode ser compartilhada e uma secreta que somente o dono tem acesso. Toda transação requer uma assinatura para ser considerada válida e para provar a posse do recurso dispendido.

Resumos criptográficos

Resumos criptográficos, ou hash, são funções matemáticas que geram um resumo, uma espécie de impressão digital dos dados de entrada. Quando aplicadas a um determinado conjunto de dados ela irá gerar como saída um valor, que a princípio, é único[1]. Um dos usos mais frequentes para o hash é verificar a integridade de arquivos. Por exemplo, ao assinar digitalmente um documento, a pessoa que o recebe, além de verificar a chave usada para a assinatura, também compara o hash fornecido pelo emissor do documento com o calculado na hora do recebimento. Caso o documento sofra alguma alteração, os resumos serão diferentes. O tamanho da saída do hash depende do algoritmo usado, mas o importante é que ela seja sempre do mesmo tamanho, não importando o tamanho da entrada. Exemplos de algoritmos de hash são o SHA-256 e o RIPEMD160, ambos usados pelo Bitcoin. Os algoritmos de hash devem possuir algumas características:

- **Unidirecionalidade:** Deve ser computacionalmente muito difícil encontrar a entrada a partir do resumo.
- **Compressão:** É desejável que o tamanho do resumo represente uma fração pequena dos dados.
- **Facilidade de cálculo:** Não deve ser custoso calcular o valor do resumo.
- **Difusão:** Para dificultar a engenharia reversa do algoritmo ao mudar um bit dos dados de entrada o valor do resumo deve ser alterado de uma quantidade de bits próxima a 50%.
- **Colisão:** Deverá ser computacionalmente difícil encontrar dois valores de entrada que gerem o mesmo resumo.

Criptografia

Todos os sistemas de criptografia usam uma transformação de uma mensagem clara em uma ilegível. Para realizar esta transformação são usadas algumas transformações matemáticas sobre a mensagem clara juntamente com uma chave. Após essas transformações é obtido um texto cifrado que poderá ser lido apenas por quem possuir a chave para decifrar.

[1]Podem existir dois conjuntos de dados com o mesmo hash, mas a probabilidade dessa ocorrência é extremamente baixa.

Um dos sistemas de criptografia mais simples foi o utilizado por Júlio Ceasar, que consistia em substituir as letras do texto por outras posteriores espaçadas da mesma chave. Por exemplo, escrever SBC com chave 13 resulta em FOP. Este tipo de criptografia também pode ser classificado como Criptografia de Chave Privada ou Simétrica, onde uma única chave é usada para criptografar e de-criptografar o segredo.

Outro tipo de criptografia é a Assimétrica ou de Chave Pública. Onde usa-se um par de chaves, uma pública e outra privada, a primeira para criptografar e a segunda para de-criptografar e vice-versa. Isso é possível graças ao uso de algumas funções matemáticas que possuem a propriedade de serem irreversíveis. As mais usadas são a fatoração em números primos (IFP - *Integer Factorization Problem*), curvas elíptica s(ECDLP - *Elliptic Curve Discrete Logarithm Problem*) ou logaritmos discretos (DLP - *Discrete Logarithm Problem*). A eficiência de um sistema de criptografia pode ser medida considerando:

- **Carga Computacional:** Mede a eficiência com que os algoritmos podem implementar as transformações com as chaves públicas e privadas.
- **Tamanho da Chave:** O NIST indica o uso de pares de chave (pública, privada) com tamanhos, em bits, para cada tipo de implementação: RSA (1088,2048), DSA (1026,160), e ECC (161,160). O ECC apresenta grande vantagem nesse aspecto.
- **Tamanho de Banda:** Corresponde a quantidade de bits necessária para transmitir uma mensagem, após codificar ou assinar.

[Jansma and Arrendondo, 2004] comparou o ECC com os RSA e chegou a conclusão de que para um mesmo nível e segurança a ECC possui uma menor carga computacional, menor tamanho de chave e menor tamanho de banda. Por esses motivos O Bitcoin adotou o sistema de curvas elípticas definida em um padrão chamado *secp256k1*, estabelecido pelo Instituto Nacional de Padronização e Tecnologia (NIST). Para maiores informações sobre curvas elípticas é recomendado [Hankerson et al., 2006].

Assinatura Digital, Endereço e Carteira

Uma assinatura digital é a cifragem do hash de um documento usando uma chave privada, tal que a chave pública, da mesma pessoa que assinou, seja usada para provar que foi ela quem assinou aquele documento.

O Bitcoin adota o *Elliptic Curve Digital Signature Algorithm* (ECDSA) para realizar assinaturas. É uma versão baseada em curvas elípticas. Assume-se que a dificuldade do logaritmo não permita que terceiros assinem um documento sem que tenha conhecimento da chave privada de uma pessoa. Pensando de modo inverso, se é impossível forjar a assinatura, então uma assinatura válida não pode ser refutada pelo dono da chave. Normalmente, o processo de assinar um documento é realizado sobre seu resumo criptográfico. Uma vantagem de se usar estas funções é que elas sempre geram como saída uma pequena quantidade de bits de mesmo tamanho. A assinatura deve ser capaz de prover integridade, não repúdio e autenticidade.

No Bitcoin a chave privada é obtida gerando um número aleatório de 256bits, uma chave pública é obtida ao efetuar a multiplicação da chave privada por um ponto na curva conhecido como "ponto gerador". Ele é sempre o mesmo para todos os usuários do Bitcoin e é definido na especificação secp256k1. O resultado da multiplicação da chave privada pelo ponto gerador é um ponto na curva, este ponto é a chave pública. Os nós armazenam somente as suas chaves privadas, pois ele pode a qualquer momento gerar a pública correspondente.

A partir deste ponto, o nó já possui um par de chaves ele pode gerar o endereço. O endereço, não confundir com endereço IP, é um número obtido usando a chave pública do nó. Ele é usado para informar ao sistema quem é o dono daquela transação, pois somente quem possuir a chave privada que gerou aquele endereço poderá usar o que estiver na transação, seja um montante monetário ou um dado. Para gerar o endereço o nó deve realizar uma operação de duplo hash, primeiro SHA-256 depois RIPEMD160: $Endereco = RIPEMD160(SHA256(ChavePublica))$, após deve converter o resultado para Base58. Este é o endereço Bitcoin.

Os usuários do Bitcoin possuem chaves que permitem provar a posse de transações. Essas chaves precisam ser armazenadas, e geralmente são armazenadas em uma carteira digital. A carteira tem a função de gerar as chaves dos usuários. Existem dois tipos de carteira: as determinísticas e as aleatórias. As determinísticas usam uma chave inicial, chamada de semente, para criar as demais através de uma função hash. Armazena apenas a primeira chave, pois todas as outras podem ser recalculadas. As carteiras Aleatórias precisam usar algum algoritmo de geração de números aleatórios para gerar as chaves, e precisa armazenar todas as chaves criadas.

4.2.4. Rede peer-to-peer(P2P)

A rede do Bitcoin foi pensada para ser uma rede de consenso descentralizada, pois essa descentralização é um dos pontos chaves de sua mentalidade. Desta forma foi desenvolvida uma rede par-a-par (ou P2P, do termo em inglês *peer-to-peer*), onde todos os participantes da rede são iguais, não existindo um nó centralizador, e todos são onerados para manter a rede funcionando. Todos os nós se interconectam na forma de uma rede sobreposta.

Existem quatro funções que podem ser assumidas por um nó na rede: Roteamento; Base de dados Blockchain; Mineração; e Carteira. Um nó completo possui todas as quatro funções, mas todos os nós possuem pelo menos a função de roteamento. Estas funções foram separadas pois nem todos os participantes da rede precisam executar todas as funções. Um usuário comum, por exemplo, que busca somente um meio de pagamento possui apenas a carteira e o roteamento. Desta forma ele pode se conectar a rede e realizar transações somente com um celular, sem a necessidade de armazenar toda a cadeia de blocos.

Para entrar na rede, é necessário conhecer ao menos um nó participante. Cada nó pode iniciar até 8 (*Outbounds Conections*) conexões e aceitar até 117 (*Inbounds Conections*). No core do Bitcoin existe gravado uma lista com alguns nós conhecidos como *Seeders*, que tem o objetivo de entregar uma lista de outros nós ativos da rede, para que o novo nó estabeleça conexão. Mas o nó não é obrigado a se conectar aos *Seeders*. Todas

as conexões são TCP. Para estabelecer a conexão inicial o nó realiza um *HandShake* com uma mensagem *version*. Assim que a conexão é estabelecida, o nó envia uma mensagem GETADDR solicitando uma lista de endereços IP conhecidos. De posse desta lista, inicia o processo novamente para outros nós, desta lista, a fim de se tornar bem conectado. Após a primeira vez que o nó é conectado, ele guarda em disco uma lista com todos os nós que estabeleceu conexão recentemente. Assim das próximas vezes que se ligar a rede pode não necessitar do auxílio dos *Seeders*.

Existem duas tabelas chamadas de *Tried e New*. A primeira armazena os endereços que o nó já estabeleceu alguma conexão, iniciada ou recebida. Ela é uma estrutura de dados com 64 entradas, chamadas de recipiente, onde cada recipiente armazena 64 endereços. A segunda tabela possui 256 recipientes, também com a capacidade de armazenar 64 endereços, e é usada para guardar os endereços recebidos das mensagens ADDR e os endereços removidos da *Tried*. Quando um nó precisar estabelecer uma nova conexão, ele irá escolher um endereço de uma das duas tabelas: *Tried ou New*. Para isso, ele usa a seguinte fórmula, que dá a probabilidade de escolha da *Tried*:

$$P_{tried} = \frac{\sqrt{\theta(9-\varepsilon)}}{(\varepsilon+1)+\sqrt{\theta(9-\varepsilon)}}$$

Onde θ é a razão entre a quantidade de endereços armazenados na *Tried* sobre a *New* e ε é quantidade conexões iniciadas.

Além das mensagens *version e ADDR* o protocolo especifica mensagens para troca de dados, que são as mensagens para difusão das transações e dos blocos. A Figura 4.4 sintetiza as principais mensagens utilizadas.

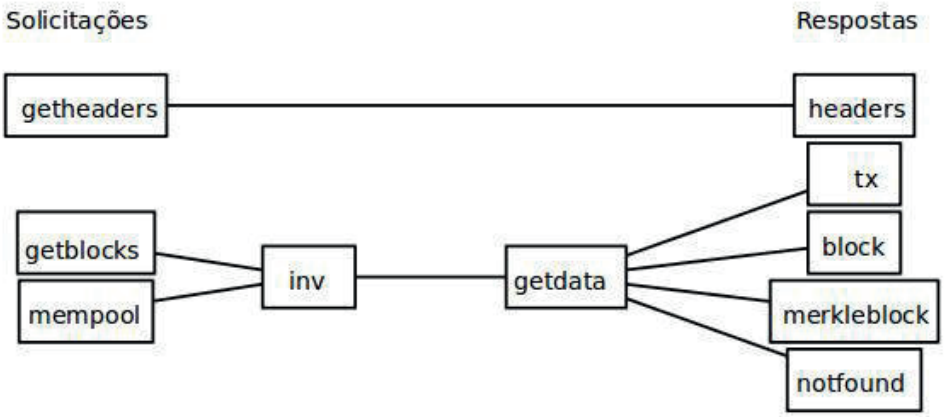


Figura 4.4. Mensagens do protocolo Bitcoin

Alguns nós da rede são nós simples, que possuem apenas funções de roteamento e carteira. Esses nós não possuem uma visão completa da rede e necessitam de ajuda de outros nós para fazer checagens de rotina, por exemplo, ao receber um pagamento um nó quer saber se o valor recebido é válido. O protocolo então específica que os nós completos podem realizar essas checagens e responder aos nós simples. Para isso eles

proveem um serviço de RPC(*Remote Procedure Call*) de modo que aqueles nós que são limitados possam realizar consultas sobre a rede e realizar operações típicas de carteira.

4.3. Blockchain

O conceito da Blockchain começa a deixar claro que vai muito além da inovação tecnológica. Está causando um grande impacto, primeiramente mudando a forma de fazer negócios de modo centralizado para uma forma descentralizada, conferindo confiabilidade na realização de transações entre agentes distribuídos e mutuamente não confiáveis, sem a necessidade de uma entidade intermediária confiável por ambos. Além disso tem a capacidade de mudar a maneira como são realizadas todos os tipos de transações e habilitar uma gama imensa de possibilidades em outras áreas, como computação segura entre múltiplos participantes (MPC - Multi-Party Computation) [Zyskind et al., 2015a], uso em Organizações Autônomas Descentralizadas(DAC - Descentralized Autonomous Corporation) [Swan, 2015b], e aplicações governamentais [Andrea, 2014].

É possível dividir sua evolução em três etapas [Swan, 2015a]: Blockchain 1.0, 2.0 e 3.0. Blockchain 1.0 é o uso comercial com transferência de moeda, remessa, e sistemas de pagamento digital, amplamente difundido pelo uso do Bitcoin e derivados. Blockchain 2.0 é o seu uso com contratos, toda a lista das questões econômicas, mercado e aplicações financeiras que o utilizam de maneira mais extensa do que transações simples de caixa, como: ações, títulos, empréstimos, hipotecas e contratos inteligentes. Blockchain 3.0 refere-se o seu uso em aplicações além da moeda, finanças e mercados, particularmente nas áreas de governo, saúde, ciência e etc.

4.3.1. Definição da Blockchain

Satoshi Nakamoto [Nakamoto, 2008] (pseudônimo dos desenvolvedores iniciais do Bitcoin) introduziu o Blockchain como mecanismo para garantir irretratabilidade, auditabilidade, e imutabilidade a fim de prover segurança a transações eletrônicas, servindo como um grande livro razão distribuído. Este mecanismo é visto como a principal inovação introduzida pelo Bitcoin. Ele representa uma forma de alcançar um consenso entre participantes não confiáveis. Normalmente instituições como bancos ou cartórios são responsáveis pela guarda e segurança do registro de transações e são chamados de terceiros de confiança. O sistema proposto por Nakamoto elimina a necessidade destas entidades, pois todos os registros são, além de públicos, mantidos de maneira descentralizada por diversos participantes da rede. A Figura 4.5 é uma visão simplificada da rede, onde pode-se observar as funções principais que cada nó pode utilizar. Observa-se que é uma rede sobreposta e que os vizinhos podem estar, inclusive, em outros continentes.

De maneira simplificada a Blockchain é uma estrutura de dados que armazena transações de forma ordenada e ligada ao bloco anterior, servindo como um sistema de registros distribuído. Essa estrutura é dividida em duas partes: cabeçalho e transações, e armazena informações detalhadas a respeito das transações que contém. Assim é possível associar uma transação ao seu endereço de origem e destino. Cada bloco possui uma identificação única gerada a partir de um resumo criptográfico de hash conforme explicado na seção anterior. O cabeçalho possui um campo que armazena o hash do bloco imediatamente anterior, desta forma conseguimos estabelecer uma ligação, um "elo", en-

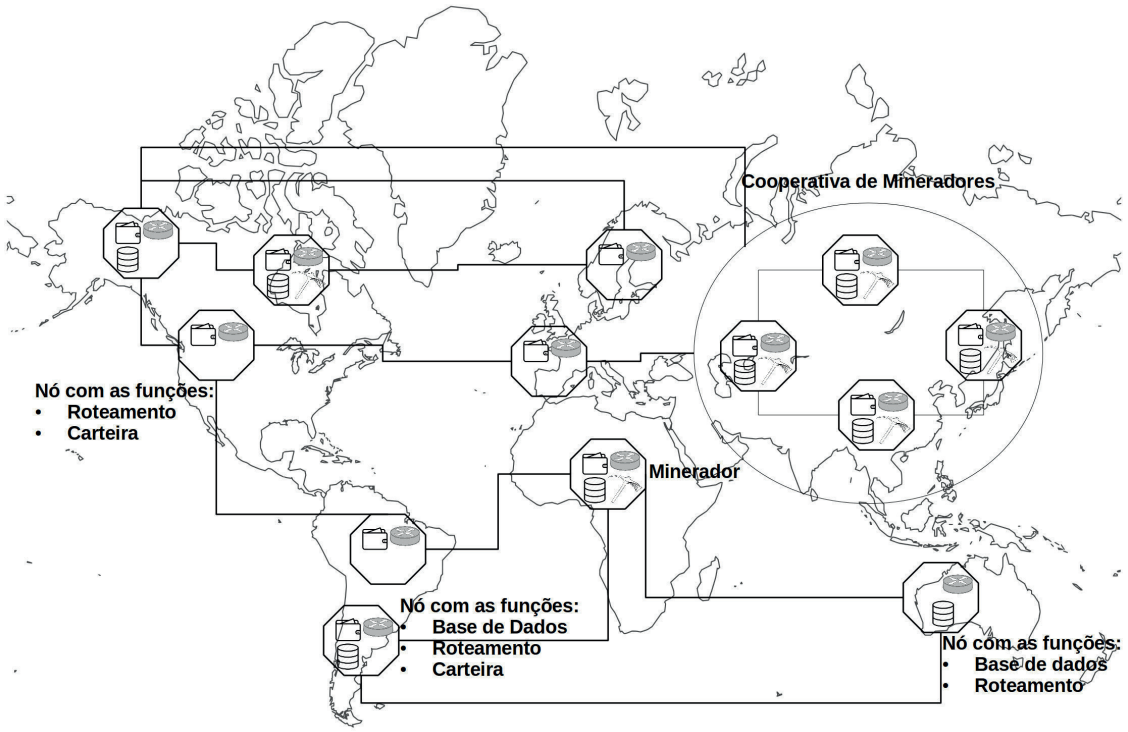


Figura 4.5. Visão Geral da Rede Bitcoin

tre os blocos. Por esse motivo, essa estrutura recebeu o nome de "corrente de blocos"ou Blockchain (ver Figura 4.6). Outra característica desta ligação é que esse hash é obtido a partir de uma colisão parcial, que será explicada mais detalhadamente a seguir, processo esse que requer um grande poder computacional para ser calculado. Como cada bloco faz referência ao seu antecessor, se um bit do bloco anterior for alterado, seu hash irá mudar e consequentemente será necessário recalcular o hash de todos os blocos descendentes. Por esse motivo assume-se que a existência de em uma cadeia longa de descendentes torna o bloco imutável, garantindo a segurança das transações armazenadas.

4.3.2. Estrutura de um Bloco

As partes principais de um bloco são o cabeçalho e as transações. As transações são o agrupamento dos dados que são armazenados no bloco. Por sua vez, o cabeçalho possui diversos campos, dois quais os mais importantes para seu funcionamento são: hash do bloco anterior, dificuldade, nonce, e raiz da árvore de merkle. Além destes, também é preciso entender dois metadados: altura do bloco e hash do cabeçalho, que são armazenados de forma a identificar o bloco e sua posição na cadeia. Estes campos serão detalhados abaixo, pois o correto entendimento da Blockchain depende deles.

4.3.2.1. Cabeçalho do Bloco

- **Altura:** Os blocos são incluídos na cadeia de forma linear em ordem cronológica, cada novo bloco recebe um número de ordem, a diferença entre o número do último

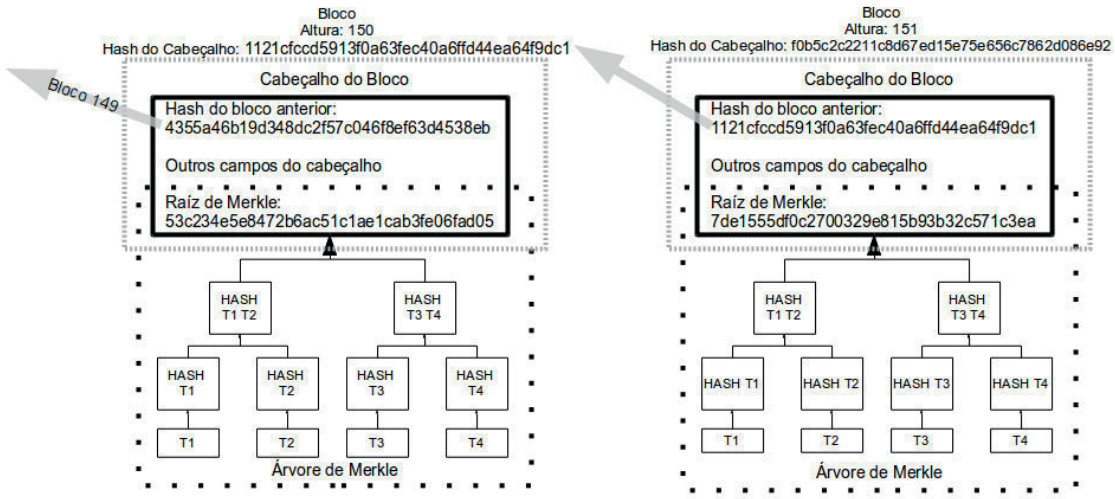


Figura 4.6. Estrutura dos Blocos Simplificada

bloco e o primeiro é chamada de altura. Este campo nem sempre é usado para identificar um bloco, pois pode haver, momentaneamente, dois ou mais blocos com a mesma altura. Neste caso ocorre um desvio, um fork, na cadeia.

- **Hash do cabeçalho:** É o principal identificador do bloco, é obtido ao realizar uma operação de resumo criptográfico no próprio cabeçalho. Ele não faz parte da estrutura de dados do bloco e também não é enviado pela rede junto com o bloco. Ele é computado isoladamente por cada nó completo no ato do recebimento de um novo bloco, e é armazenado em um banco de dados separado como parte dos metadados do bloco. Ao contrário da altura, o hash do cabeçalho pode ser usado para identificar um bloco de forma inequívoca.

- **Hash do bloco anterior:** Este campo é incluído no cabeçalho de modo a possibilitar que um novo bloco seja ligado ao anterior. Como vimos na Figura 4.6, o bloco 236 possui, em seu cabeçalho, o hash do bloco 235. Os nós completos armazenam os metadados dos blocos. Assim todos os nós possuem o hash do bloco 235, tão logo o bloco 236 seja recebido, por um nó completo, ele irá verificar este campo e definirá que o bloco 236 é filho do 235.

- **Nonce:** É um número usado como uma variável para modificar a saída da função hash do cabeçalho. Em conjunto com o campo dificuldade alvo ele é usado para provar que um minerador realizou um trabalho e encontrou um cabeçalho que atenda aos critérios estabelecidos para essa prova. Para ficar mais claro imagine que seja estabelecido que o hash do cabeçalho inicie com uma sequencia de três zeros, o minerador então irá por força bruta iterar o nonce até que o hash do cabeçalho atenda a esse requisito. No recebimento do novo bloco os nós completos irão calcular o hash do cabeçalho apenas uma vez.

- **Dificuldade:** A dificuldade nada mais é que uma colisão parcial de hash, ou seja, como descrito anteriormente, um algoritmo de hash gera sempre um mesmo resumo

para uma determinada entrada. Se for alterado um bit que seja desta entrada o hash resultante será completamente diferente. Assim depende do poder computacional do nó minerador achar um hash que satisfaça a essa colisão parcial. O mecanismo usado para gerar a colisão é o nonce. Como ele faz parte do cabeçalho, sempre que é alterado o resumo também muda. Quando a dificuldade é configurada para com 1bit (zero) basta achar um hash que inicie com um zero e qualquer valor para os outros 255bits, ou seja $2^2$55 possibilidades, será considerado válido. Caso seja setado com 2bits as possibilidades serão reduzidas para 254bits ou $2^2$54, com 10bits serão $2^2$46 possibilidades e assim por diante. É possível observar que a diminuição do espaço de possíveis valores que satisfaçam a colisão implica em maior dificuldade em achar um resumo que satisfaça a dificuldade, logo, mais computação é requerida, ou mais tempo de mineração, e maior gasto com energia.

O processo de incluir novos blocos à cadeia é chamado de mineração, e o nós que realizam o trabalho de gerar um novo bloco é chamado de minerador. A taxa pela qual novos blocos são incluídos na cadeia é definida pelos desenvolvedores de cada projeto de Blockchain. Na rede Bitcoin foi estabelecido um alvo de 10min, ou seja, a dificuldade é ajustada por todos os nós completos e mineradores para que, em média, a cada 10min um novo bloco seja incluído na cadeia. É esperado que novos mineradores se juntem a rede e novos equipamentos mais poderosos sejam lançados, com isso, em média, o tempo de inclusão de novos blocos tende a diminuir. Para evitar que novos blocos sejam incluídos a intervalos menores que 10min a dificuldade é ajustada, aumentando a quantidade de bits para a colisão. Assim, como será mais difícil achar o novo hash o tempo de inclusão de novos blocos irá se ajustar até ficar próximo ao alvo de 10 minutos. Cada nó minerador recalcula, independentemente, a nova dificuldade a cada 2016 novos blocos realizando a seguinte operação matemática:

$$Nova\ Dificuldade = Dificuldade\ Antiga * (\frac{Tempo\ n\ Blocos}{(Tempo\ Alvo * n\ Blocos)})$$

- **Transações:** No Bitcoin uma transação é uma transferência de valores. De maneira simplificada é um conjunto de entradas (endereços de onde os valores serão retirados) e saídas (endereços para onde os valores serão enviados). Um nó após criar uma transação a envia a todos os seus vizinhos. Os nós que receberam a transação a retransmitem aos seus vizinhos, para que a transação alcance todos os nós da rede. Quando um minerador recebe a transação ele irá guardá-la para que ela seja incluída em um próximo bloco que será minerado. Quando este bloco for incluído na cadeia, a transação se torna pública e imutável. As transações são assinadas com um sistema de chaves públicas. Para enviar um valor a alguém é necessário possuir a chave privada para assinar a transação, provando a posse do valor. Também é necessário conhecer a chave pública do usuário que irá receber o valor, para cifrar a transação de modo que somente o detentor da chave privada, que faz par com a pública de destino, conseguirá decifrá-la. Desta forma é possível que o sistema seja público e ainda assim somente quem realmente é dono da transação poderá usá-la.

  Existem outros dois tipos de transações na rede Bitcoin, os contratos inteligentes, que serão explicados melhor ao longo do capítulo, e o armazenamento de dados, chamado de *OP_RETURN*. Foi destinado um campo com 40bytes para ser usado

para armazenamento de dados diversos. É endereçado da mesma forma que uma transação financeira, assim o recebedor dos dados precisa possuir a chave necessária para usá-los.

- **Árvores de Merkle:** Uma árvore de Merkle [Merkle, 1987], ou árvore de hash binário, é definida como uma árvore binária completa com um valor de k bits associado a cada nó da árvore, de modo que cada valor de nó interior seja uma função unidirecional dos valores de seus filhos. São projetadas para que um valor de folha possa ser verificado em relação a um valor de raiz conhecido publicamente, bastando serem fornecidos os valores dos pares correspondentes no caminho da folha até a raiz.

  Na Blockchain ela é usada para resumir, eficientemente, as transações contidas em um bloco, para tal é necessário produzir $2 * \log_2 N$ hashes. Logo, ela fornece um processo muito eficiente para verificar se uma transação consta em um bloco. Para construir esta árvore deve-se iniciar pelas folhas, que contêm o hash das transações. Para criar a árvore é necessário um número par de transações, caso haja um número ímpar a última folha da árvore será duplicada. As folhas são então agrupadas duas a duas e seu hash produz um nó pai, os nós pai são então agrupados em pares e sofrem o mesmo processo de modo que esse processo continue até que não haja mais pares, gerando assim um nó raiz chamado de raiz de merkle, conforme Figura 4.7.

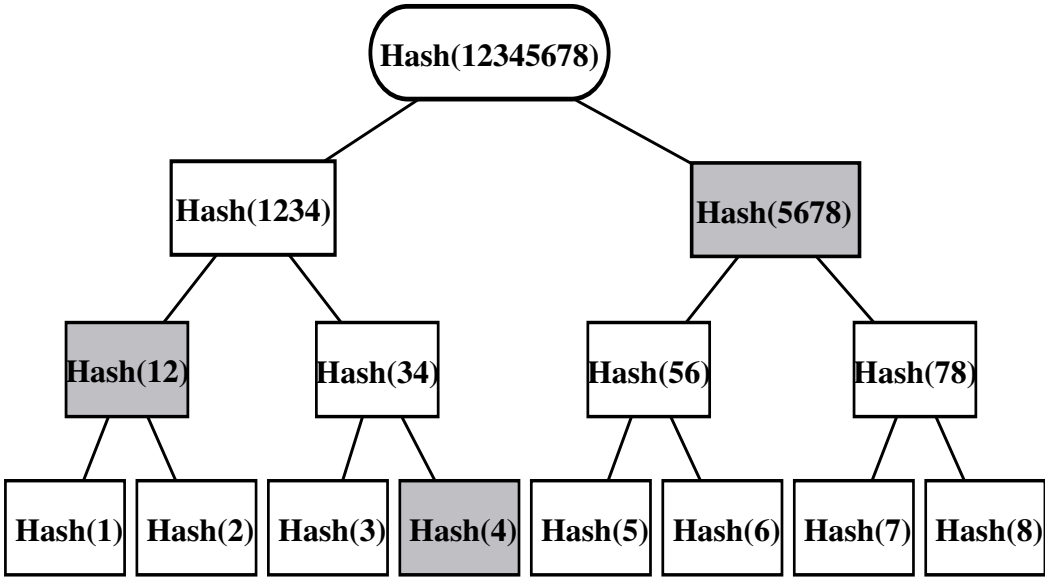


Figura 4.7. Árvore de merkle

Para provar que uma transação está inclusa em um bloco basta fornecermos o caminho que a transação ira percorrer na árvore, este caminho consiste no hash do complemento dos pares. Assim é possível realizar esta verificação rapidamente em meio a milhares de transações. Isso é particularmente útil pois para verificar se uma transação consta em um determinado bloco não é necessário solicitar o bloco inteiro à rede, basta o cabeçalho do bloco e o caminho até a transação. Como vimos anteriormente um nó simplificado não possui a cadeia de blocos armazenada.

Caso ele necessite confirmar uma transação precisará da ajuda de um nó completo. Por exemplo, na Figura 4.7 cada folha corresponde ao hash de uma transação e os valores em cinza correspondem ao caminho para provar que a transação consta no bloco. Para provar que a transação 3 consta no bloco o nó completo enviará o cabeçalho deste bloco e os hash(4), hash(12) e hash(5678) ao nó simplificado. De posse desses dados é possível calcular a raiz da árvore e comparar com o valor da raiz de merkle que consta no cabeçalho do bloco. O Nó simplificado fará o cálculo do hash(3) que juntamente com o hash(4) calculará o hash(34), pega o valor do hash(12) e chega a hash(1234) e por último usa o hash(5678) para calcular a raiz, cujo valor é hash(12345678).

4.3.3. Mineração

A mineração é o processo responsável por atualizar a Blockchain, pelo qual alguns nós especiais, chamados de mineradores, incluem as transações em um bloco e geram um cabeçalho válido para essas transações. O mineradores gastam muita energia para realizar a Prova de Trabalho, por esse motivo precisam ser recompensados. A primeira transação do bloco é sempre uma transação especial chamada de *Coinbase*. Ela tem dois propósitos, incluir novas moedas ao sistema e recompensar o minerador. Na rede Bitcoin, a mineração tem dois propósitos. Primeiramente, incluir novas moedas ao sistema e em segundo lugar proteger as transações realizadas. Para gerar esse cabeçalho os mineradores devem calcular a árvore de merkle das transações, verificar a dificuldade estabelecida, incluir a estampa de tempo e realizar uma série de cálculos a fim de encontrar um nonce que satisfaça a dificuldade em vigor. Assim será descrito a importância da dificuldade e como ela se ajusta automaticamente, além de mostrar um passo a passo do processo de mineração.

A mineração consiste em gerar um novo bloco. Para isso o minerador primeiro cria um "rascunho"de um bloco. É sobre esse rascunho que ele vai trabalhar até que obtenha um bloco viável para ser enviado aos todos os nós da rede. O rascunho é a estrutura de dados que vai comportar os dados do cabeçalho e as transações. Após criar essa estrutura em branco, o minerador preenche alguns campos do cabeçalho: hash do bloco anterior, estampa de tempo, versão e dificuldade. Restando preencher a raiz da arvore de merkle, o nonce e agrupar as transações.

As transações, ao serem geradas, são enviadas via *broadcast* a todos os nós vizinhos e estes reencaminham aos seus vizinhos. Os mineradores, ao receberem uma mensagem com uma transação, às armazenam em uma base de dados de transações ainda não mineradas. As transações permanecem temporariamente em uma espécie de fila com prioridade até que sejam retiradas para ser incluídas em um novo bloco. Cada minerador possui uma fila diferente de transações, e pode selecionar quais transações ele vai incluir nesse novo bloco. Após selecionar quais transações serão incluídas ele irá gerar uma árvore de merkle e incluir o valor da sua raiz no cabeçalho.

Agora falta achar o valor do nonce que fará parte do novo bloco. Esta é a etapa demorada do processo, requer um grande poder computacional dos mineradores e consequentemente um enorme gasto de energia, conforme foi explicado na seção anterior. Para se ter ideia da tempo para achar um hash válido atualmente são comercializados dispositivos especializados em calcular hash, esses dispositivos atingem a marca de 9TH/s,

ou seja conseguem calcular nove trilhões de hash por segundo, que com a dificuldade atual da rede Bitcoin seriam necessários 13 anos para achar um hash válido. A seguir um exemplo: Para achar o nonce que produza um hash válido para "Simpósio Brasileiro de Segurança"com a dificuldade alvo de "000"(12bits), ou seja o alvo é que o hash inicie com três zeros em sequência. para isso é pode-se concatenar o nonce com a informação que será resumida - sha256("Simpósio Brasileiro de Segurança+nonce") - incrementando o nonce a cada insucesso até que seja encontrado um hash válido. Em um terminal do Linux é possível fazer:

for nonce in $(echo {0..10000}); do echo "$(echo "Simpósio Brasileiro de Segurança +$nonce sha256sum) $nonce"; done | grep ^000

Como resultado será obtido: 000a45ed0ebded66019dff14fc916c429aac6021494 e4983960e27c917696db5 - 4060

O que significa dizer que o nonce 4060 ao ser acrescido a "Simpósio Brasileiro de Segurança"gera um hash que atende a dificuldade alvo. É importante notar que existem outros valores de nonce que geram resultados válidos, como o 5470. A partir deste momento qualquer um que possua uma implementação do sha256 pode calcular o resumo de "Simpósio Brasileiro de Segurança+4060"e comparar com o hash fornecido, demonstrando assim que o resultado é válido.

Assim que o nonce é encontrado o nosso rascunho fica completo e portanto o bloco está pronto para ser enviado a todos os nós da rede. Os nós da rede ao receberem um novo bloco iniciam uma série de verificações a fim de validar o bloco e chegar a um consenso em caso de bifurcações (*"forks"*).

4.3.4. Consenso e Prova de Trabalho

A cadeia de blocos não é criada por uma autoridade central. Os blocos são criados independentemente pelos mineradores da rede. Os nós, usando as informações que são transmitidas através de conexões inseguras, conseguem chegar à mesma conclusão e fabricar o mesmo registro público que todos os outros nós. Atingindo assim um consenso global. Os nós completos armazenam toda cadeia com os blocos que foram validados por ele. Quando diversos nós possuem os mesmos blocos em sua cadeia principal é considerado que eles chegaram ao consenso. Esta subseção descreve as regras de validação de cada bloco e como o consenso é alcançado e mantido e também explica alguns dos vários mecanismos de consenso que são utilizados atualmente.

O mecanismo de consenso é composto por duas etapas: validação do bloco e seleção da maior cadeia. Estas duas etapas são realizadas de maneira independente por cada nó. Os blocos são enviados em *broadcast* pela rede, e cada nó ao receber um novo bloco o retransmite aos seus vizinhos, mas antes desta retransmissão o nó faz a validação do bloco a fim de garantir que somente blocos válidos sejam propagados. Existe uma extensa lista de verificação a ser seguida, dentre elas:

- Estrutura do bloco;
- Verificar se o hash do cabeçalho atende a dificuldade estabelecida;
- Tamanho do bloco dentro dos limites projetados;

- Verificação de todas as transações; e
- Verificação da estampa de tempo.

Por definição da Blockchain cada bloco tem somente um pai, mas pode ocorrer uma situação em que um ou mais mineradores gerem novos blocos quase ao mesmo tempo, fazendo com que haja um ou mais filhos com um mesmo pai. Neste caso, entende-se que ocorreu um fork, uma bifurcação, na cadeia. A última etapa do mecanismo de consenso serve exatamente para selecionar qual destes blocos fará parte da cadeia principal e qual será descartado. Isso é possível em virtude da prova de trabalho, que será abordada nessa seção, fundamental para o mecanismo de consenso adotado, pois como vimos anteriormente para gerar o bloco, mineradores gastam muita energia em busca de um bloco válido.

Como é possível a ocorrência de bifurcações, os nós armazenam os blocos sem pai(órfãos) [2] e mantém duas cadeias, uma principal e uma secundária. Os blocos órfãos acontecem quando dois blocos são gerados em espaços curtos de tempo e chegam em ordem inversa, ou seja, um bloco foi recebido e não faz referência a um bloco na cadeia. Ele é armazenado por um período de tempo, caso o nó receba um bloco que seja pai do órfão ele será incluídos na cadeia em sua ordem correta. Note que neste caso não houve a ocorrência de uma bifurcação, os blocos apenas foram recebidos fora de ordem.

Como existem diversos mineradores gerando blocos de forma descentralizada, os novos blocos enviados por eles podem chegar a diferentes nós em momentos diferentes, o que pode resultar em visões diferentes. Para ficar mais claro quando dois mineradores geram blocos fazendo referência a um mesmo pai ocorre a bifurcação, e os outros mineradores deverão escolher qual bloco eles irão adotar como referência. Se uma parte dos mineradores adotar um bloco e outra parte adotar o outro, essas duas cadeias irão coexistir até que uma fique maior que a outra. Para resolver esta situação, os nós que se comportam de maneira honesta, de acordo com o mecanismo de consenso, sempre irão adotar a maior cadeia e o fork estará resolvido. A corrente principal é a maior cadeia, aquela onde há a maior quantidade de trabalho acumulada. Na Figura 4.8 os blocos cinza bifurcaram da cadeia principal, como alcançaram uma altura maior passaram a ser a cadeia principal. Os blocos brancos 127,128 e 129 são descartados e suas transações são consideradas como não confirmadas, devendo ser incluídas futuramente em outros blocos.

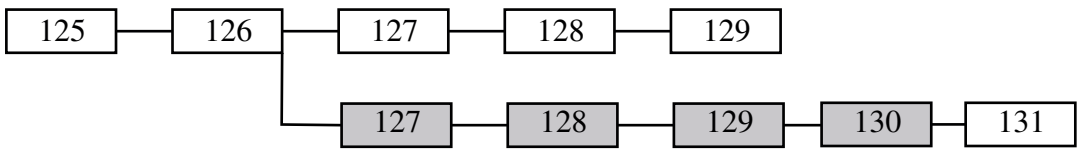


Figura 4.8. Bifurcação

Uma das preocupações mais comuns para os sistemas de moedas digitais é a possibilidade do gasto duplo, quando um usuário malicioso gasta um mesmo valor em duas

[2] Situação rara e temporária

transações diferentes da cadeia. Note que para ocorrer a tentativa de um gasto duplo é necessário uma bifurcação, pois se o gasto ocorrer na mesma cadeia, quando o novo bloco for criado, ele não passará nas verificações iniciais de consistência e será descartado. Com o fork, o usuário malicioso faz um gasto e envia para rede, gasta a mesma quantia novamente em outro lugar e começa a minerar sobre esse gasto. Desta forma, há a possibilidade de ele conseguir minerar um bloco e realizar o fork. A partir deste momento, a rede estará dividida e como mencionado anteriormente haverá uma corrida que será vencida pela maior cadeia. Uma das transações será descartada e o gasto duplo será rejeitado. Como uma das cadeias irá ser aceita pela rede e, a outra descartada, eventualmente o gasto duplo será detectado. É usualmente aceito na rede Bitcoin que uma transação é considerada confirmada quando existem seis novos blocos com altura maior que a sua, pois será necessário muito esforço para alterá-la.

Um cenário de ataque contra o mecanismo de consenso é chamado de "ataque de 51%". Nesse cenário, um grupo de mineradores, controlando uma maioria (51%) do poder de hash total da rede, conspira para atacar o Bitcoin. Com a habilidade de minerar a maioria dos blocos, os mineradores atacantes podem gerar bifurcações deliberadas na Blockchain, gerar transações de gasto duplo ou executar ataques de negação de serviço (DoS) contra endereços ou transações específicas. Um ataque de bifurcação/gasto duplo é um ataque onde o atacante faz com que blocos já confirmados sejam invalidados ao fazer uma bifurcação em um nível abaixo deles, com uma posterior re-convergência em uma cadeia alternativa. Com poder suficiente, um atacante pode invalidar seis ou mais blocos em uma sequência, invalidando transações que antes eram consideradas imutáveis (com seis confirmações). Note que o gasto duplo só pode ser feito nas transações do próprio atacante, para as quais o atacante pode produzir uma assinatura válida. Fazer um gasto duplo da própria transação é rentável quando, ao invalidar uma transação, o atacante puder receber um pagamento irreversível ou um produto sem ter que pagar por isso.

Alcançar o consenso em um sistema distribuído é um desafio. Os algoritmos de consenso devem ser resilientes a falhas de nós, particionamento da rede, atrasos de mensagens, mensagens que chegam fora de ordem e corrompidas. Eles também têm que lidar com nós egoístas e deliberadamente maliciosos. Vários algoritmos tem sido propostos para resolver isso, cada um realizando o conjunto de suposições necessárias em termos de sincronia, transmissões de mensagens, falhas, nós maliciosos, desempenho e segurança das mensagens trocadas. Para uma rede Blockchain, alcançar consenso garante que todos os nós na rede concordem com um estado global consistente da cadeia de blocos.

Segundo [Kim, 2014, Natoli and Gramoli, 2016], um protocolo de consenso tem três propriedades fundamentais com base nas quais sua aplicabilidade e eficácia podem ser determinadas:

- **Segurança:** Um protocolo de consenso é determinado para ser seguro se todos os nós produzirem o mesmo resultado (*agreement*) e os resultados produzidos pelos nós são válidos de acordo com as regras do protocolo (*validity*). Isso também é referido como consistência do estado compartilhado.

- **Vivacidade:** Um protocolo de consenso garante a vivacidade se todos os nós que seguem o protocolo eventualmente produzem um valor (*termination*), ou seja, se

um nó gerar uma transação e enviá-la a todos os nós da rede em algum momento um minerador irá incluí-la em um bloco.

- **Tolerância a falhas:** Capacidade de continuar a operar, chegar ao consenso, adequadamente, mesmo após a falha de alguns nós da rede.

O resultado de impossibilidade de Fischer Lynch Paterson (FLP) afirma que um sistema de consenso assíncrono determinista pode ter no máximo duas destas três propriedades. Este é um resultado comprovado, ou seja, qualquer sistema de consenso distribuído na Internet deve sacrificar uma dessas propriedades [Fischer et al., 1985].

A maioria das plataformas Blockchain existentes, mais de 90% da capitalização do mercado total de moedas digitais, utilizam o mecanismo de consenso, na sua forma original e computacionalmente cara, que é a Prova de Trabalho. Porém, existem vários outros mecanismos que oferecem certas vantagens desejadas em relação ao modelo original, como a prova de Prova de Posse (PoS, do inglês *Proof of Stake*) [King and Nadal, 2012], o Algoritmo de Tolerância a Falhas Bizantinas (PBFT, do inglês *Practical Byzantine Fault Tolerance*) [Castro and Liskov, 2002] e a Prova do Tempo Decorrido (PoeT, do inglês *Proof of Elapsed Time*) [Intel,] aparecem como outras alternativas e serão brevemente explicadas a seguir:

- **Prova de Trabalho:** A ideia principal da Prova de Trabalho (PoW, do inglês *Proof of Work*) é tentar evitar ataques cibernéticos. Para atingir este objetivo, utiliza-se de um sistema onde o usuário deve provar que gastou um certo tempo para encontrar alguma resposta que satisfaça algum requisito que o verificador pedir. A tarefa de encontrar tal resposta, é baseada em dois princípios. Em primeiro lugar, a PoW tem que ser difícil e trabalhosa, mas não impossível; e em segundo lugar a verificação dessa prova deve ser muito mais rápida e fácil de ser realizada. Este conceito foi inicialmente proposto por Adam Back [Back et al., 2002] e é utilizado por diversos sistemas de prova e também pelo Bitcoin.

  No Bitcoin, a Prova de Trabalho é gerada da seguinte forma: o remetente adiciona um número arbitrário à mensagem (chamado de nonce) e aplica uma função matemática de hash na mensagem. O SHA-256 [Gilbert and Handschuh, 2003] é usado pelo Bitcoin. O objetivo é encontrar uma resposta com um certo número de zeros na frente. Ele repete o procedimento variando o nonce até achar essa resposta. Como é relativamente difícil encontrar tal resposta, ao receber a mensagem, todo usuário será capaz de verificar que houve um grande esforço do remetente em gerá-la. Ao decifrar o problema, o minerador gera um novo bloco. A dificuldade da prova de trabalho é ajustada a cada 2016 blocos, a fim de que seja gerado em média um bloco a cada dez minutos. A segurança da PoW baseia-se no princípio de que nenhuma entidade deve reunir mais de 50% do poder de processamento da rede porque essa entidade poderá efetivamente controlar o sistema, manipulando a cadeia mais longa.

- **Prova-de-Posse (PoS)** - também é utilizada em plataformas Blockchain de criptomoedas. Enquanto a PoW recompensa os participantes que resolvem enigmas criptográficos complicados, baseados em hash, para validar transações e criar novos

blocos (mineração), a PoS requer uma certa quantidade de moeda para sua participação. O criador do próximo bloco é escolhido de uma maneira Probabilística, e a chance de uma conta ser escolhida depende de sua "riqueza"(ou seja, a posse). Os algoritmos de Prova-de-Posse são projetados para superar as desvantagens dos algoritmos PoW em termos do alto consumo de energia envolvido nas operações de mineração. A PoS substitui completamente a operação de mineração por uma abordagem alternativa envolvendo participação de usuários ou propriedade de moeda virtual no sistema de Blockchain. Dito de outra forma, em vez de um usuário gastar R$ 2000 comprando equipamentos de mineração para iniciar os trabalhos no algoritmo PoW e ganhar uma recompensa pela mineração, o mesmo usuário, utilizando a PoS pode comprar R$ 2000 em criptomoedas e usá-las como participação para comprar cotas proporcionadas de criação de blocos no sistema Blockchain, tornando-se um validador. Em criptografia de PoS, os blocos costumam ser validados, em vez de minerados. Essas implementações não oferecem incentivos para que os nós votem no bloco correto. Portanto, os nós podem votar em vários blocos que suportam vários forks para maximizar suas chances de ganhar uma recompensa. Neste caso não gastam nada ao fazê-lo em oposição a PoW, onde o nó dividiria seus recursos para votar em múltiplos forks. Este é o problema *"Nothing-at-Stake"*, que precisa ser abordado para uma implementação correta e eficiente de PoS. A seleção pelo saldo da conta resultaria em centralização (indesejável), pois o único membro mais rico teria uma vantagem permanente. Vários métodos de seleção diferentes foram planejados, por exemplo, Nxt [Kim, 2016] e BlackCoin [Vasin, 2014] usam aleatorização para prever o próximo gerador de blocos, usando uma fórmula que procura o menor valor de hash em combinação com o tamanho da participação. Uma vez que as apostas são públicas, cada nó pode prever, com precisão razoável, qual conta ganhará o direito de validar um bloco.

- **Algoritmo de Tolerância a Falhas Bizantinas (PBFT)** - A função de um protocolo de consenso é manter a ordem das transações em uma rede de cadeias de blocos, apesar das ameaças a essa ordem. Uma dessas ameaças é a falha arbitrária simultânea, um dos tipos de falha bizantina, de múltiplos nós de rede. Usando PBFT, uma rede de nós Blockchain pode tolerar nós defeituosos até f, onde f é uma fração arbitrária conhecida do número total de nós - com uma máquina de estados replicada em nós diferentes (uma réplica sendo definida como primária). O algoritmo PBFT funciona da seguinte forma:

  – Um cliente envia uma solicitação de serviço para a máquina primária.
  – A primária replica o pedido para os backups.
  – As réplicas executam o pedido e enviam respostas.
  – O cliente aguarda $f + 1$ respostas idênticas de réplicas diferentes para considerar um resultado correto.

  Como o número total de nós precisa ser conhecido, o PBFT não é adequado para sistemas públicos, sendo utilizado apenas em sistemas privados. Uma rede PBFT garante a consistência e a integridade dos dados quando ocorrem falhas bizantinas em até $1/3$ dos nós da rede. Por exemplo, usando PBFT, uma rede de cadeia de

blocos de nós N pode suportar f número de nós Bizantinos, onde $f = (N-1)/3$. Em outras palavras, o PBFT garante que um mínimo de $2*f+1$ nós alcancem consenso sobre a ordem das transações antes de anexá-las ao livro razão compartilhado. A regra $2*f+1$ tem as seguintes implicações:

Como são necessários um mínimo de $2*f+1$ nós para chegar a um consenso antes de prosseguir para o próximo bloco de transações, o livro-razão em qualquer nó adicional (além de $2*f+1$) ficará temporariamente atrasado. Este atraso na sincronização do livro-razão geral compartilhado em todos os nós é uma limitação inevitável em qualquer rede PFBT.

- **Prova do Tempo Decorrido (PoET)** um algoritmo de consenso, projetado pela Intel. PoET usa um modelo de eleição aleatória de um líder, que irá validar os blocos. Funciona essencialmente da seguinte forma, existe um *hardware* especializado para gerar um valor de tempo aleatório. Cada validador solicita um tempo de espera a este *hardware*. O validador com o tempo de espera mais curto para um determinado bloco é eleito o líder, e espera este tempo para validar o bloco. Após este bloco ser incluído na cadeia o processo se repete. Este modelo é proposto para uso em Blockchains privados, pois, em teoria os validadores são honestos. A aleatoriedade na geração de tempos de espera garante que a função líder seja distribuída de forma uniforme e entre todos os validadores. Uma desvantagem desse algoritmo é a dependência de hardware especializado.

4.3.5. Categorias de Blockchain com base no acesso aos dados

Blockchain pode ser classificado com base no acesso aos dados e na participação do mecanismo de consenso sobre quaisquer mudanças propostas no seu livro razão. Podendo ser:

- **Blockchain sem permissão ou (Pública):** O mecanismo de consenso está aberto a todos. O objetivo de uma cadeia sem permissão é permitir que qualquer pessoa contribua com dados. Isso cria a chamada resistência da censura, o que significa que nenhum ator pode evitar que uma transação seja adicionada à cadeia. Os participantes mantêm a integridade da cadeia ao chegar a um consenso quanto ao seu estado. Qualquer um pode se juntar à rede e participar do processo de verificação de blocos para criar consenso e também criar contratos inteligentes. Ter um sistema sem permissão implica assumir que pode não haver confiança entre os nós, portanto, um mecanismo de consenso fortemente distribuído deve ser imposto. Em tal sistema, existe a possibilidade de um ataque Sybil [Douceur, 2002], onde um nó de rede tenta aparecer como vários nós distintos criando um grande número de pseudoidentidades. Uma influência desproporcionalmente grande por um único nó é uma ameaça, então a introdução do PoW na validação da transação é logicamente justificada e necessária.

- **Blockchain permissiva ou (Privada):** Participantes no processo de consenso estão pré-selecionados. Quando um novo registro é adicionado, a integridade do livro razão é verificada por um processo de consenso realizado por um número limitado

de atores confiáveis. Isso torna a manutenção de um registro compartilhado muito mais simples do que o processo de consenso sem permissão. As cadeias de blocos permitidas fornecem conjuntos de dados altamente verificáveis porque o processo de consenso cria uma assinatura digital, que pode ser vista por todas as partes. As características que derivam de sistemas confiáveis podem abrir a possibilidade de evitar um protocolo de consenso computacionalmente exigente, como a PoW.

Muitos projetos foram iniciados para tornar a Blockchain mais popular e viável para diferentes modelos de negócios e aplicações, aproveitando as categorias existentes. A Tabela 1.1 resume as principais características de algumas aplicações com base em Blockchain. Bitcoin, Ethereum [Wood, 2014] são exemplos de Blockchain sem permissão e Hyperledger [Cachin, 2016] e Ripple [Pilkington, 2015] são exemplos de Blockchain com permissão. É possível verificar uma diferença crítica entre essas duas categorias que é o modelo de mineração subjacente - Blockchains sem permissão usam a Prova de Trabalho (PoW) onde o poder de *hashing* é oferecido para criar confiança, já as Blockchains permissivas não precisam usar a mineração baseada em energia computacional para chegar a um consenso, já que todos os atores são conhecidos, eles acabam usando algoritmos de consenso como PBFT que podem ser usados para alcançar o consenso sem mineração por PoW, levando a um tempo de processamento de bloco bem inferior comparado ao tempo da Blockchain sem permissão, sendo praticamente considerado realizado em tempo real. .

Tabela 4.1. Comparação entre sistemas Blockchain

Blockchain	**Bitcoin**	**Ethereum**	**Hyperledger**	**Ripple**
Natureza	Sem Permissão	Sem Permissão	Permissiva	Permissiva
Validação	PoW SHA-256	PoW - ethash	PBFT	BFT customizado (RPCA)
Propósito	criptomoeda	contrato inteligente	Chaincode	criptomoeda
Linguagem	Scripts baseados em pilha	Código interno Turing completo	Go, Java	C++
Tempo de proc. do bloco	$\sim$ 600 s	$\sim$ 15 s	$\sim$ tempo real	$\sim$ tempo real

4.3.6. Contratos inteligentes

O uso da tecnologia Blockchain proporcionou múltiplas classes de funcionalidades de aplicações em todos os segmentos de negócios com criptomoedas, mercados e transações financeiras. Desde o início sua utilidade foi pensada para ser usada além da moeda e dos pagamentos prevista para Bitcoin; As possibilidades de dinheiro programável e contratos foram preparadas no Bitcoin, em sua invenção. Uma comunicação de 2010 de Satoshi Nakamoto indica que "o design suporta uma tremenda variedade de possíveis tipos de transações que eu planejei anos atrás: Transações de custódia, contratos vinculados, arbitragem de terceiros, assinatura multipartidária, etc. Se o Bitcoin se encaixar de forma

importante, estas são coisas que queremos explorar no futuro, mas todas elas deveriam ser projetadas no início para, com certeza, fazer que elas sejam possíveis mais tarde". Os conceitos e estrutura desenvolvidos para Bitcoin são extremamente portáteis e extensíveis.

Em 2014, surgiu o termo "Blockchain 2.0", sendo usado para descrever um novo projeto no campo de banco de dados distribuído da Blockchain. Uma parte da terminologia que se refere amplamente ao espaço Blockchain 2.0 e inclui Bitcoin 2.0 e seus protocolos, contratos inteligentes, propriedades inteligentes, Dapps (aplicativos descentralizados), DAOs (organizações descentralizadas autônomas) e DACs (corporações autônomas descentralizadas) [Buterin, 2014].

Considerando que o Blockchain 1.0 é para a descentralização de dinheiro e pagamentos, o Blockchain 2.0 é para a descentralização de mercados de forma mais geral e contempla a transferência de muitos outros tipos de ativos além da moeda usando a cadeia de blocos. A ideia-chave é que as funcionalidades do livro-razão de transações descentralizadas da cadeia de blocos podem ser usada para registrar, confirmar e transferir todo tipo de contratos e propriedades.

Os Contratos Inteligentes são *scripts* armazenados na Blockchain e executam suas instruções de maneira distribuída em todos os participantes do contrato. O termo Contratos Inteligente (do termo em inglês, *Smart Contracts*) guarda similaridade com o contrato legal. Neste sentido ele regula a interação entre diferentes entidades. Nick Szabo introduziu este conceito em 1994 e definiu um *Smart Contract* como "um protocolo de transação computadorizado que executa os termos de um contrato" [Szabo, 1994]. Szabo sugeriu a transposição de cláusulas contratuais em código e incorporá-las em propriedades (hardware ou software) que possam auto-executá-las [Szabo, 1997], de modo a minimizar a necessidade de intermediários confiáveis entre as partes das transações, e a ocorrência de exceções maliciosas ou acidentais.

Os Contratos Inteligentes possuem um endereço, e são acionados endereçando uma transação para ele. Em seguida, é executado de modo independente e automático da forma prescrita em cada nó da rede, de acordo com os dados que foram incluídos na transação desencadeadora. Isto implica que cada nó em uma Blockchain, habilitado por contrato inteligente, está executando uma máquina virtual e que a rede Blockchain atua como uma máquina virtual distribuída [Christidis and Devetsikiotis, 2016].

Três características dos contratos inteligentes os diferenciam são: autonomia, auto-suficiência e descentralização [De Filippi and Mauro, 2014]. Autonomia significa que, depois de lançado e em execução, um contrato e seu agente iniciador não precisam estar em contato. A segunda característica é que os contratos inteligentes podem ser auto-suficientes em sua capacidade de gerar recursos, isto é, arrecadar fundos ao fornecer serviços ou emitir equidade e gastá-los em recursos necessários, como o poder de processamento ou o armazenamento. E a terceira característica é que os contratos inteligentes são descentralizados, na medida em que não subsistem em um único servidor centralizado, sendo distribuídos e auto-executados em nós de rede.

O exemplo clássico usado para demonstrar contratos inteligentes sob a forma de código que é executado automaticamente é uma máquina de venda automática. Ao contrário de uma pessoa, uma máquina de venda automática se comporta de forma algorítmica,

ou seja, o mesmo conjunto de instruções será seguido toda vez em todos os casos. Quando você deposita o dinheiro e faz uma seleção, o item é lançado. Não há possibilidade da máquina não cumprir o contrato um dia, ou apenas cumpri-lo parcialmente (desde que não esteja quebrada). Um contrato inteligente de forma semelhante, executa o código pré-especificado.

A confiança mínima geralmente torna as coisas mais convenientes, tirando o julgamento humano da equação, permitindo uma automação completa. Um exemplo de um contrato inteligente básico na Blockchain são aqueles utilizados em apólices de seguro. As companhias de seguros podem automatizar políticas de seguro, escrevendo-as para um contrato inteligente, quando as condições de entrada do contrato inteligente mudam para um evento segurado, por exemplo, no caso de uma catástrofe natural, o processo de reivindicações é desencadeado imediatamente. Os parâmetros mensuráveis do evento, como a velocidade do vento, a localização de um furacão ou a magnitude de um terremoto podem ser registrados na Blockchain. À medida que os parâmetros atravessam certos limiares pré-acordados, o processo de reclamações é desencadeado imediatamente e a quantidade exata de pagamento financeiro pode ser entregue sem necessidade de intervenção humana. A transparência e a confiança no processo são visíveis para todas as partes interessadas e todos os órgãos reguladores.

Um outro exemplo de contratos inteligentes, agora para a plataforma de Internet das Coisas, é o monitoramento de entrega de encomendas. Atualmente, por exemplo, os pacotes podem se perder na postagem, porém com o advento da Internet das Coisas, com sensores em todos os lugares, da prateleira no armazém, ao endereço do destinatário. Cada sensor forma seu próprio nó em uma Blockchain e contratos inteligentes podem gravar a "posse"do dispositivo em cada sensor individual (e local subsequente). Um dispositivo de rastreamento na embalagem será lido em cada sensor no caminho para o destinatário. Cada vez que é lido por um novo sensor, sua localização é transmitida e acordada por todos os participantes da IoT na Blockchain. Um contrato inteligente, em seguida, mantém as guias de "posse"ao longo de todo o caminho, solidificando a confiança de saber exatamente onde encontrar o pacote.

Ethereum, é o *framework* mais conhecido e utilizado para contratos inteligentes. Ethereum é uma máquina virtual descentralizada, que executa programas chamados contratos a pedido dos usuários. Contratos são escritos em uma linguagem *Turing-completa bytecode*, chamado *EVM bytecode* [Wood, 2014]. Um contrato é um conjunto de funções, cada uma definida por uma sequencia de instruções *bytecode*. Uma característica notável dos contratos é que eles podem transferir éter (uma criptomoeda similar ao Bitcoin) para/de usuários e para outros contratos. As transações são usadas para:

- criar novos contratos;
- invocar funções de um contrato;
- transferência de éter para contratos ou para outros usuários.

Todas as transações são registradas em uma Blockchain pública. A sequência de transações na Blockchain determina o estado de cada contrato, e o saldo de cada usuário.

Na Figura 4.9 é apresentado um exemplo de código para um contrato inteligente básico escrito para uso na Blockchain Ethereum. É apresentado um código básico, na linguagem solidity, para a criação de um *token* digital que no ecossistema Ethereum pode representar qualquer bem negociável: moedas, certificados de ouro, etc. Como todos os *tokens* implementam algumas características básicas de uma maneira padrão, isso também significa que esse *token* será instantaneamente compatível com a carteira Ethereum e qualquer outro cliente ou contrato que use os mesmos padrões.

O comando *mapping*, cria uma matriz associativa, onde é associado endereços com saldos. Os endereços estão no formato hexadecimal básico da ethereum, enquanto os saldos são inteiros, variando de 0 a 115 *quattuorvigintillion* (que corresponde a uma incontável quantia de *vigintillions*). A palavra-chave *public* significa que esta variável será acessível por qualquer pessoa no bloco, o que significa que todos os saldos são públicos (como devem ser, para que os clientes possam exibi-los). A função *MyToken* tem o mesmo nome que o contrato *MyToken*, se for renomear um, deve-se renomear o outro também: esta é uma função de inicialização especial que é executada apenas uma vez e uma vez somente quando o contrato é carregado pela primeira vez para a rede. Esta função irá definir o saldo do *msg.sender*. A função *transfer* é muito direta, ela tem um destinatário e um valor como parâmetro e sempre que alguém a chama, subtrairá o valor de seu saldo e o adicionará ao saldo do destinatário. De imediato, haverá um problema óbvio que acontece se a pessoa quiser enviar mais do que possui. Para solucionar esse problema é necessário a implementação de uma verificação rápida e, se o remetente não tiver fundos suficientes, a execução do contrato simplesmente irá parar.

```
contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken(
        uint256 initialSupply
        ) {
        balanceOf[msg.sender] = initialSupply;              // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) {
        require(balanceOf[msg.sender] >= _value);           // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                    // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
    }
}
```

Figura 4.9. Exemplo de Contrato inteligente fonte: https://www.ethereum.org/token acessado em: 22/08/2017

Uma vez que os contratos têm um valor econômico, é crucial garantir que a sua execução seja executada corretamente. Os potenciais conflitos na execução de contratos (devido, por exemplo, a falhas ou ataques) são resolvidos através de um protocolo de consenso baseado em PoW. Idealmente, a execução de contratos é garantida, mesmo com a presença de um usuário malicioso, desde que ele não possua a maioria do poder

computacional da rede. A rede Ethereum atualmente usa um algoritmo de consenso PoW, chamado Ethash, criado especificamente para Ethereum. Foi construído para dificultar seu processamento por meio de hardwares específicos, como o chips ASICs. Está prevista a alteração do mecanismo de consenso da rede Ethereum até o final do ano de 2017, será utilizado o PoS.

A segurança do processo de consenso baseia-se na suposição de que é mais conveniente para um minerador seguir o protocolo do que tentar atacá-lo. Para manter esse pressuposto, os mineradores recebem alguns incentivos econômicos para executar os cálculos exigidos pelo protocolo. Parte desses incentivos é dada pelas taxas de execução pagas pelos usuários em cada transação, ou etapa de execução de um contrato. Um atacante até poderia então criar um contrato com uma execução longa, mas ele ficaria caro demais, pois ele necessitaria pagar taxas a cada etapa do processo. Desta forma, as taxas limitam a quantidade de etapas de execução de um contrato, impedindo assim ataques de negação de serviço que usam computações demoradas.

4.4. Casos de uso do Blockchain para prover segurança e privacidade em IoT

Os dispositivos na IoT coletam, geram e processam dados, enviam estas informações através da internet, produzindo uma gigantesca massa de informação a ser usada pelos mais diversos serviços.

Apesar dos benefícios, problemas críticos relacionados a privacidade podem emergir. A Blockchain pode ter um papel fundamental no desenvolvimento de aplicações descentralizadas que irão executar em bilhões de dispositivos. Entender como e quando esta tecnologia pode ser usada para prover segurança e privacidade é um desafio, diversos autores apontam esses desafios, dos quais cita-se [Conoscenti et al., 2016, Dorri et al., 2016, Dorri et al., 2017a].

Nesta seção será explorado como a Blockchain pode ser usada para beneficiar as aplicações de segurança para Internet das Coisas, como aplicações descentralizadas que permitam aos objetos inteligentes interagir com segurança, estabelecer mecanismos de pagamentos [Wörner and von Bomhard, 2014], criar serviços de Infra-Estrutura Pública de Chaves (PKI) [Nguyen et al., 2015, Ali et al., 2016], realizar Computação Segura entre Múltiplos Participantes (MPC) [Zyskind et al., 2015a], suportar Ambientes Inteligentes [Dorri et al., 2017b] e etc. Segundo [Zyskind et al., 2015b] a privacidade pode ser alcançada, por exemplo, ao se combinar o uso do Blockchain com um sistema de armazenamento de dados descentralizado via P2P.

Além do problema da privacidade, segurança é também uma questão fundamental para os sistemas que farão uso de Blockchain. Isto porque a Blockchain pode ser usada para prover não repúdio, autenticidade, confidencialidade e autorização de forma descentralizada. Serão apresentados os ataques mais comuns abordados na literatura, tais como: minerador egoísta [Eyal, 2015, Nayak et al., 2016], gasto duplo [Gervais et al., 2016] e o eclipse [Heilman et al., 2015]. Será descrito como a Blockchain pode ser usado para prover controle de acesso em armazenamento descentralizado [Ouaddah et al., 2017], e para realizar Computação Segura entre Múltiplos Participantes [Zyskind et al., 2015a]. Será realizada a simulação de um ataque demonstrativo, que visa mostrar como a latência de propagação dos blocos e a taxa de inclusão de novos blocos [Gervais et al., 2016] podem

influenciar a segurança do mecanismo de consenso.

4.4.1. Blockchain para prover anonimidade e controle de acesso em IoT

Primeiramente é bom ressaltar que a anonimidade provida pelo uso da Blockchain não é absoluta, por isso ela é comumente chamada de pseudo-anonimidade. É possível, em certas circunstâncias, de-anonimizar o dono da transação, ou seu endereço IP. Para de-anonimizar as transações existem algumas técnicas específicas, [Conoscenti et al., 2016] as dividiu em quatro:

- **Múltiplas Entradas:** Em alguns casos para realizar determinado gasto é necessário reunir saldo de diversas contas. Caso seja preciso guardar o saldo total da carteira em uma única conta, é possível realizar a transferência dos saldo menores para uma única conta, esse procedimento é chamado de transação com múltiplas entradas. Como, para realizar esta transação, é necessário possuir a chave privada de cada entrada é sensato supor que todas as contas pertencem a um mesmo usuário. A partir deste momento é possível associar os endereços a um usuário. Esta abordagem foi utilizada em [Spagnuolo et al., 2014, Moser et al., 2013, Herrera-Joancomartí, 2015].

- **Endereços de Troco:** Como já visto, todas as transações no Bitcoin são transferências de recursos. Por definição do protocolo, É obrigatório gastar todo o saldo associado a uma determinada chave. Caso o valor da transação seja menor que o valor da entrada, essa transação irá gerar troco. O valor do troco deve retornar ao dono e por isso deve ser a endereçado uma saída com destino ao próprio usuário. Caso o usuário use sempre o mesmo endereço para receber o troco de suas transações, pode-se associar este endereço aos endereços de entrada anteriores e descrever exatamente todos os gastos de um usuário, além da possível correlação com fontes secundárias de informação como sites de redes sociais. Esta abordagem foi utilizada em [Spagnuolo et al., 2014, Moser et al., 2013, Herrera-Joancomartí, 2015].

- **Associação ao IP:** A rede Bitcoin é uma rede sobreposta a rede IP. Grande parte das mensagens da rede são transmitidas em *broadcast* para os vizinhos diretos de cada nó. Uma grande quantidade de vizinhos permite a um nó é extrair algum conhecimento da rede, como sua topologia, quem são os nós mineradores, localização dos nós e seu endereço IP. Em [Koshy et al., 2014], o autor conseguiu associar o endereço IP ao endereço do usuário ao escutar o tráfego da rede e utilizar um algoritmo de clusterização.

- **Uso de Serviços Centralizados:** Os usuários podem, por diversos motivos, não guardar e gerenciar suas próprias chaves privadas e delegam essa função a serviços terceirizados. Alguns autores [Moser et al., 2013, Valenta and Rowan, 2015] acham um isso risco a privacidade, pois esta entidade pode vazar seus dados, com isso suas identidades e seus recursos, e até mesmo utilizar os recursos de terceiros, pois a prova de propriedade se dá pela posse da chave privada que está nas mãos de terceiros.

Segundo [Conoscenti et al., 2016] são necessários cuidados extras a fim de mitigar estes problemas. Os dispositivos IoT devem se configurados para: sempre usar um endereço diferente para receber troco; sempre gerar um endereço novo para cada recebimento recursos; não usar serviços terceirizados. Essas medidas não são suficientes para prover anonimidade total, mas proverão um certo grau de segurança em manter as identidades preservadas, evitando principalmente correlacionar um determinado dispositivo ao seu dono.

É possível também usar a Blockchain para armazenar dados e prover controle de acesso a eles. Suponha que um sensor de presença queira armazenar seus histórico diário na Blockchain. Ele irá gerar uma transação com os dados a serem armazenados, e assinará essa transação com sua chave secreta, assim todos saberão qual sensor é dono desses dados. O sensor indicará como saída da transação as chaves públicas com direito de ler seus dados. Ele enviará esta transação aos mineradores de sua rede, que autenticarão a transação e a incluirão no próximo bloco. Como a Blockchain é pública todos os usuários tem acesso a seus dados, e saberão que um determinado usuário tem o direito de ler o histórico produzido pelo do sensor de presença. Mas, somente aqueles detentores das chaves privadas, que fazem par com a públicas indicadas pelo sensor, conseguirão ler o histórico diário que foi disponibilizado pelo sensor.

Ouaddah [Ouaddah et al., 2017] propôs o FairAccess, um *framework*, que usa a Blockchain para habilitar aos usuários controlar seus próprios dados. Ele reutiliza o código do Bitcoin e introduz alguns novos tipos de transação usados para controlar o acesso aos dados, como: *"grant"e "revoke"access*. O modelo prevê a existência de alguns atores: recurso a ser compartilhado; o dono do recurso; e os usuários. As transações são usadas para controlar o acesso dos usuários e a Blockchain é usada servir como local para armazenamento e leitura das permissões. Os autores fizeram uma prova de conceito com um *Raspberry PI* com uma câmera("Recurso"). Foi criado um usuário "dono do recurso"e outro usuário "utilizador". O Dono controla o acesso ao recurso através de transações enviadas a Blockchain. Assim para conceder acesso ao usuário ele envia uma transação do tipo *grant access* e a envia ao usuário, como se estivesse vendendo um produto com Bitcoin. Esta transação será minerada pela rede, o que significa dizer que será verificado que o dono possui a chave privada referente ao recurso e a transação será incluída na Blockchain. A partir deste ponto o utilizador solicitará acesso diretamente ao recurso que verifica na Blockchain se existe uma transação que lhe garanta acesso, neste caso o usuário conseguirá acessar a câmera.

Uma das principais críticas ao armazenamento de dados na Blockchain é o uso de estrutura de dados que não foram projetadas para armazenar grandes quantidades de informação. Assim, caso o bloco comece a ser usado com esse fim, haverá diversas cópias de um mesmo arquivo sendo mantidas na rede, uma vez que a cadeia inteira é mantida de foma descentralizada. Além do desperdício de espaço, há a forma ineficiente de gerenciar estes dados. Com o intuito de usar a segurança provida pela Blockchain, Zyskind [Zyskind et al., 2015b] combina o uso do armazenamento de dados fora da cadeia com o controle de acesso na cadeia de blocos. O armazenamento é realizado com um sistema de DHT (distibuted hash table) sendo mantido por um conjunto de nós da rede previamente selecionados. Os dados são replicados de maneira eficiente pelos nós de forma a garantir a alta disponibilidade e repartida de forma que nenhum nó tenha o

arquivo inteiro. A Blockchain é então usada de forma a gerenciar onde esses dados estão distribuídos e quem tem acesso a eles. Para isso, são gerados dois novos tipos de transação, uma para prover o controle de acesso e outro para controlar a distribuição dos dados no DHT.

Como a Blockchain não possui ponto central de falha e não é governada por uma única entidade, ela permite uma nova classe de aplicativos e serviços descentralizados, como por exemplo, um servidor raiz DNS ou uma autoridade de certificação raiz. Esses benefícios motivaram Ali et al. [Ali et al., 2016] a usar a Blockchain para construir um novo sistema de PKI descentralizado e um sistema de identidade, chamado *Blockstack ID*. O formato para publicar chaves públicas é semelhante ao PGP [Zimmermann, 1995]. *O Blockstack* desacopla o registro do nome e propriedade da disponibilidade de dados associados, separando os planos de controle e dados. O plano de controle define o protocolo para o registro de nomes legíveis para humanos, criando elos (nome, hash). O plano de controle consiste em um bloco e uma camada logicamente separados do plano de controle, sendo responsável pelo armazenamento e disponibilidade de dados. Ele consiste em um local para pesquisa de dados por hash ou URL, e um sistema de armazenamento externo de dados. Todos os dados armazenados são assinados pela chave do respectivo proprietário do nome. Ao armazenar os dados fora da cadeia de blocos, o *Blockstack* permite valores de tamanho arbitrário e permite uma variedade de *backends* de armazenamento. Os usuários não precisam confiar na camada de armazenamento porque podem verificar a integridade dos valores de dados no plano de controle.

4.4.2. Uso de Blockchain em cenários econômicos para garantir transações eletrônicas em IoT

O futuro da IoT é se tornar uma rede de dispositivos autônomos que podem interagir uns com os outros e com seu ambiente, e tomar decisões inteligentes sem a intervenção humana. Este é o lugar onde a Blockchain pode ajudar a alavancar a IoT e formar uma base que suportará a economia compartilhada baseada em comunicações da máquina-a-máquina (M2M). Em [Wörner and von Bomhard, 2014] os autores descreveram uma implementação prototípica simples do processo de troca de dados por dinheiro eletrônico entre um sensor e um requisitante, utilizando a rede Bitcoin. O sistema é composto de três partes:

- **Dispositivo IoT Cliente:** Precisa cumprir as seguintes tarefas: anotar uma solicitação de dados ao receber um pagamento e ser capaz de criar e publicar uma transação contendo os dados solicitados.

- **Cliente requisitante:** Precisa poder enviar pagamento ao endereço Bitcoin do sensor e deve monitorar alterações na Blockchain até detectar a transação com os dados enviados pelo dispositivo IoT.

- **Repositório de dispositivo IoT:** Onde os sensores podem ser registrados ou podem ser encontrados pelos solicitantes. Uma entrada no repositório de sensores deve conter pelo menos o endereço do Bitcoin, quais os dados que ele oferece, o preço e metadados adicionais como localização, tags, etc.

Um endereço Bitcoin é anexado ao dispositivo IoT que precisa anotar uma solicitação de dados ao receber Bitcoins e precisa ser capaz de criar e publicar uma transação contendo os dados solicitados. A maneira usada pelos desenvolvedores para utilizar a própria rede Bitcoin para vender dados é a utilização da transação do tipo *OP RETURN*.

Esse trabalho demonstrou um processo simples usando a rede Bitcoin com certas limitações, como por exemplo: os dados adquiridos estão disponíveis publicamente na Blockchain. Isso pode ser resolvido criptografando os dados com a chave pública do solicitante, em que o cliente requisitante descriptografa os dados usando sua chave privada.

Em [Zhang and Wen, 2015], os autores propõem uma arquitetura de comércio eletrônico projetada especificamente para as mercadorias IoT, baseada no protocolo do Bitcoin. Foram utilizadas Corporações Autônomas Distribuídas (DACs) como a entidade de transação para lidar com os dados de dispositivos e propriedade inteligente negociados. Nesse modelo as pessoas podem negociar com DACs para obter mercadorias IoT, utilizando criptomoedas baseadas no protocolo Bitcoin. Para trocar os dados do dispositivo são usados chaves eletrônicas e contratos inteligentes.

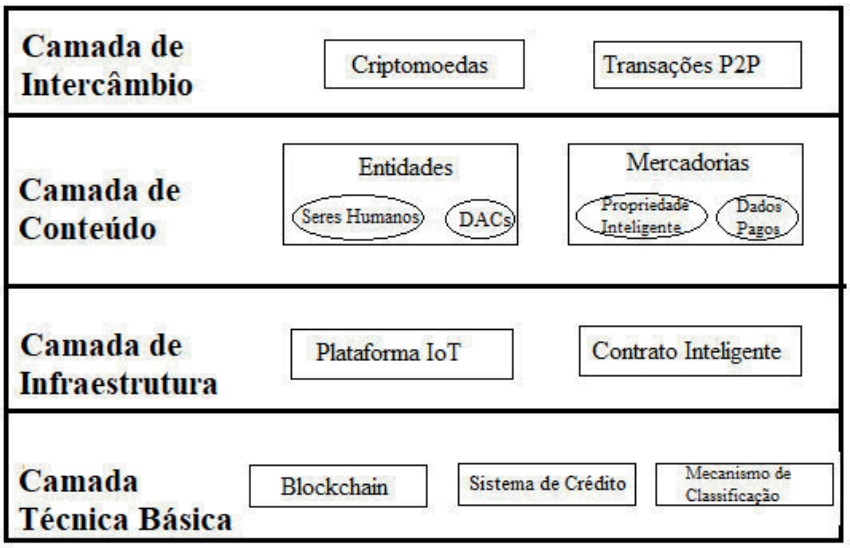


Figura 4.10. Modelo de Negócio para IoT usando Blockchain. Adaptado de: [Zhang and Wen, 2015]

Conforme mostrado na figura 4.10, são propostas 4 camadas para o modelo IoT de comércio eletrônico, que são: camada técnica básica; camada de infraestrutura; camada de conteúdo e camada de intercâmbio. A camada técnica básica inclui o módulo do mecanismo de classificação das mercadorias, módulo de algoritmo de crédito para efetuar o gerenciamento das carteiras e o módulo Blockchain Bitcoin, que foi a criptomoeda adotada pelo projeto. A camada de infraestrutura contém a plataforma do serviço de informações IoT e a plataforma de contratos inteligentes. A camada de conteúdo inclui duas partes: entidades participante e as mercadorias IoT. As Entidades consistem em DAC e seres humanos. DACs são executados automaticamente sem a interferência do ser humano, cada DAC pode comprar produtos de outros DAC como clientes, enquanto isso, todos podem emitir suas próprias mercadorias IoT. As mercadorias são propriedades inteligentes e dados coletados de sensores. As propriedades inteligentes podem ser obras

de arte, bens duráveis como carros, casas e energia como eletricidade, água, gás e óleo que podem ser controlados e quantificadas por dispositivos digitais por chaves eletrônicas ou sistema de controle de acesso. A camada de intercâmbio inclui o sistema de transações P2P que é o núcleo do modelo de negócios da IoT juntamente com com a criptomoeda escolhida que é a Bitcoin.

O emprego da tecnologia Blockchain com a finalidade de introduzir a funcionalidade de transações econômicas para IoT foi abordado por uma série de propostas e aplicativos, incluindo:

- **ADEPT** [Panikkar et al., 2014] - Telemetria Autônoma Peer-to-Peer descentralizada (do inglês, Automated Decentralized P2P Telemetry) é um sistema IoT descentralizado criado por uma parceria entre a IBM e a Samsung que utiliza elementos do projeto do Bitcoin para construir uma rede de dispositivos distribuídos (uma IoT descentralizada) permitindo que bilhões de dispositivos transmitam transações entre pares e realizem auto-manutenção, fornecendo identificação e autenticação seguras aos usuários. O conceito ADEPT utiliza a Blockchain para fornecer a espinha dorsal do sistema, utilizando uma mistura de prova de trabalho e prova de participação para transações seguras. Esta plataforma foi testada em vários cenários, incluindo um que envolve uma máquina de lavar inteligente que pode automaticamente comprar e pagar por detergente com Bitcoin ou ether e será capaz de negociar os melhores preços dos produtos de limpeza baseado nas preferências do seu proprietário. De acordo com o documento, uma máquina de lavar Samsung W9000 reconfigurada para funcionar dentro do sistema ADEPT usando contratos inteligentes para emitir comandos para um revendedor de detergente para receber novos suprimentos. Esses contratos dão ao dispositivo a capacidade de pagar pela própria encomenda e depois receber uma mensagem do revendedor de que o detergente foi pago e enviado. Essa informação é transmitida para o *smartphone* do proprietário da lavadora, um dispositivo que também é conectado à rede da *smart home*.

- **Filament** [Crosby et al., 2016] - É um sistema desenvolvido para permitir que os dispositivos tenham identidades únicas em um livro público e possam descobrir, comunicar e interagir uns com os outros de forma autônoma e distribuída. Além disso, os dispositivos envolvidos podem trocar valor diretamente ou indiretamente com uma ampla gama de entidades. Por exemplo, eles poderiam vender dados sobre condições ambientais para uma agência meteorológica. O objetivo é criar um diretório de dispositivos inteligentes que permita que os dispositivos IoT Filament se comuniquem de forma segura, executem contratos inteligentes e enviem microtransações. A pilha de tecnologia de Filament usa cinco tecnologias: blockname; telehash; contratos inteligentes; pennybank e BitTorrent. Usando o blockname, os dispositivos são capazes de criar um identificador exclusivo que é armazenado em uma parte do chip incorporado no dispositivo e gravado no bloco. O Telehash, por sua vez, fornece comunicações criptografadas de ponta a ponta e o BitTorrent permite o compartilhamento de arquivos. Os pagamentos pelo uso dos dispositivos são tratados por contratos inteligentes, o que permite que os termos dos pagamentos e o acesso ao dispositivo sejam controlados por esses programas. O Filament usa

um protocolo baseado em Bitcoin que foi desenvolvido para microtransações em sua plataforma, chamado Pennybank, devido a restrições específicas de dispositivos IoT. Os dispositivos IoT não são de alta potência e nem sempre estão online. Assim, o Pennybank cria um serviço de garantia entre dois dispositivos IoT, permitindo que eles liquidem as transações quando estiverem conectados on-line.

- **Tilepay** [Kennedy and Duranleau,] - É um sistema que oferece um mercado online seguro e descentralizado onde os usuários podem registrar seus dispositivos na cadeia de blocos e vender seus dados em tempo real em troca de moeda digital, utilizando a cadeia de bloco Bitcoin. Ao usar a Blockchain do Bitcoin, o registro, a autenticidade, a segurança e os dados disponíveis serão transmitidos pelo livro razão descentralizado. O proprietário dos dispositivos publicam seu perfil de dados através do *Marketplace* e as empresas podem comprar dados.

- **Watson IoT Platform** [Kshetri, 2017] - Essa plataforma da IBM permite que os dispositivos IoT enviem dados para Blockchain privadas. Todos os parceiros de negócios, que possuem uma Blockchain IBM podem acessar e fornecer dados de seus dispositivos IoT sem a necessidade de um controle ou gerenciamento central. Podem verificar cada transação, evitando disputas e garantindo que cada parceiro seja responsabilizado por suas funções individuais na transação global. A Blockchain IBM fornece a infra-estrutura de uma rede Blockchain privada que replica os dados do dispositivo e valida a transação através de *smart contract* seguros. A plataforma Watson IoT traduz os dados existentes do dispositivo, de um ou mais tipos de dispositivo, para o formato necessário para as APIs do contrato Blockchain. O contrato Blockchain não precisa conhecer os detalhes específicos dos seus dispositivos IoT. A plataforma Watson IoT filtra os eventos do dispositivo e envia apenas os dados necessários para o contrato.

- **IOTA** [Popov, 2016] - é uma criptomoeda desenvolvida especificamente para a comercialização de dados de dispositivos IoT. Ao invés de utilizar uma Blockchain global a IOTA usa um DAG (Grafo Acíclico Dirigido), as arestas correspondem as transações e os pesos a quantidade de vezes que ela foi confirmada. A ideia principal é que para realizar uma transação um nó precisa primeiro realizar uma série de verificações em transações anteriores com o objetivo de aprová-las. Não há diferenciação entre os nós, e todos são responsáveis por aprovar as transações, o que, segundo o autor, garante à rede uma maior escalabilidade: quanto maior a quantidade de transações, mais eficiente ela se torna. A rede IOTA foi projetada para a IoT, com uma estrutura modular e leve, que pode ser facilmente integrada a equipamentos eletrônicos, permitindo automação e a formação de um mercado M2M, em que os nós não precisam estar ligados diretamente à Internet, bastando uma conexão *bluetooth* entre eles.

Alguns outros casos de uso interessantes envolvendo monetização de dados com Blockchain e dispositivos IoT estão sendo estudados, como por exemplo, organizações como Nasdaq e Chain of Things conduzem pesquisas sobre aplicações que podem ajudar a tornar as fontes de energia renováveis disponíveis para o público em geral, onde a energia produzida por painéis solares IoT gera valor em criptomoedas que serão registrados na

Blockchain. Assim, qualquer pessoa que ingressar na rede pode fazer investimentos em tecnologia de energia renovável.

4.4.3. Uso de Blockchain em Computação Segura entre Múltiplos Participantes

Considere o seguinte problema: dois milionários interessados em saber qual deles possui a maior fortuna sem revelar a sua própria para o outro ou para terceiros. Este é o famoso problema dos milionários proposto por Yao [Yao, 1982], que foi resolvido com um protocolo para computação segura entre dois participantes. O MPC é a generalização desta solução para múltiplos participantes, é definido como o problema de *N* participantes para calcular uma função com suas entradas de forma segura, onde a segurança significa garantir o resultado correto e a privacidade das entradas, mesmo com a presença de alguns participantes maliciosos. Ao final, cada participante obterá apenas o resultado da função e não conhecerá as entradas dos demais participantes. Seu uso abre caminho para diversas aplicações como, votação na internet, mineração e compartilhamento de dados dentre outras.

Partindo do princípio que para realizar qualquer função são necessários apenas circuitos aditivos e multiplicativos, basta construir blocos MPC para adições e multiplicações e depois usar estes blocos para qualquer outras funções aritméticas. Assim, os protocolos propostos para MPC buscam realizar estas duas funções principais, normalmente usando circuitos de Yao [Yao, 1982] ou baseados em compartilhamento de segredo [Shamir, 1979] ou suas variantes. Para realizar estas funções, é necessários que os participantes troquem mensagens, para os circuitos aditivos as trocas de mensagens crescem linearmente com o número de participantes, mas para os circuitos multiplicativos são necessários $O(n^2)$ comunicações. Este fato torna a implementação da MPC restrita a poucos participantes e cenários específicos. Ao longo dos anos surgiram propostas para otimizar as soluções e aumentar a quantidade de participantes [Cramer et al., 1999, Gennaro et al., 1998, Zyskind et al., 2015a].

Na formulação de problemas de computação segura entre múltiplos-participantes, é comum adotar dois modelos de protocolos: O Modelo Semi-honesto e o Modelo Malicioso.

- **Modelo Semi-Honesto:** No modelo semi-honesto as partes seguem o protocolo apropriadamente e guardam todos os passos intermediários das computações para uma posterior análise, a fim de conseguir inferir informações secretas da outra parte.
- **Modelo Malicioso:** No modelo malicioso uma parte,maliciosa, não precisa seguir o protocolo, podendo agir de forma arbitrária ao executar o protocolo, abortar a execução a qualquer instante, usar informações falsas e guardar os passos intermediários para análise posterior.

Enigma [Zyskind et al., 2015a] é uma plataforma para MPC com garantia de privacidade. Usa a Blockchain como controladora da rede, gerenciando o controle de acesso, e servindo como log de eventos para o compartilhamento dos segredos. Tem a capacidade de computar as funções em ambos os modelos de adversários, e é escalável. Cada nó recebe suas entradas e grava as suas saídas na Blockchain. A computação é dividida entre parcelas dos participantes, assim cada parcela realiza uma tarefa e as partes são unidas ao final

para chegar ao resultado. Essa compartimentação permite que haja um maior controle na replicação dos dados, melhorando a escalabilidade do sistema e permitindo um maior número de participantes. Logo um dos seus possíveis usos é exatamente em dispositivos IoT que lidem com dados sensíveis.

Outro trabalho [Yue et al., 2016] usa a Blockchain Blockchain para realizar controle de acesso e armazenamento dos dados dos pacientes. O autor considera que o uso dos dados dos pacientes sem seu consentimento é um ataque a privacidade, mas também descreve a importância da utilização destes dados par a pesquisa médica. Ele classifica os dados em dois tipos: privados e públicos. Qualquer pesquisador ou entidade governamental poderá utilizar os dados públicos. Os dados privados somente poderão ser utilizados via MPC. Assim, o uso de MPC torna possível saber, por exemplo, a quantidade de pacientes que possuem AIDS e pertencem a grupos de risco. Torna possível extrair conhecimento sobre esses dados sem revelar a privacidade dos pacientes.

4.4.4. Uso de Blockchain para garantir segurança em Ambientes Inteligentes

A segurança e a privacidade da Internet da Coisas (IoT) continuam a ser um grande desafio, principalmente devido à enorme escala e à natureza distribuída das redes IoT. As abordagens baseadas em blocos oferecem segurança descentralizada e privacidade, mas envolvem consumo excessivo de energia e atrasos que não são adequados para a maioria dos dispositivos IoT com recursos limitados. Dorri [Dorri et al., 2016, Dorri et al., 2017b] apresenta uma solução leve de um Blockchain especialmente orientada para uso com IoT, eliminando a Prova de Trabalho e o conceito de moedas. Sua abordagem foi exemplificada em uma implementação em Ambientes Inteligentes (*Smart Home*) e consiste em três estruturas principais: armazenamento em nuvem, uma camada de sobreposição e casa inteligente. Cada casa inteligente está equipada com um dispositivo com maior poder computacional que está sempre *on-line*. Este dispositivo é chamado de "minerador", pelo autor, e é responsável por lidar com todas as comunicações dentro e fora da casa. O minerador mantém uma Blockchain privada, usada para controlar e auditar as comunicações e prover controle de acesso entre os dispositivos. Neste cenário a PoW torna-se desnecessária, pois somete um dispositivo terá o trabalho de manter a Blockchain. Os demais dispositivos da casa recebem um par de chaves para que possam realizar transações. Como exemplo, caso um sensor de presença queira ligar uma lampada ele enviará uma transação para a lampada, que irá verificar na Blockchain se aquele sensor tem permissão para acendê-la.

4.4.5. Ataques à Blockchain

No mundo de Bitcoin, as transações são consideradas válidas quando estão em um bloco e confirmadas sempre que exista uma certa quantidade blocos com altura maior na cadeia. Conforme descrito anteriormente, podem surgir bifurcações de curta duração, que tendem a ser resolvidas de acordo com a regra da cadeia mais longa. A maioria dos das bifurcações ocorre naturalmente, sem má intensão, fazendo com que as poucas transações do lado descartado sejam atrasadas. Esta abordagem funciona bem, sob o pressuposto crucial de que nenhum atacante deve ser capaz de reunir tanto poder computacional que seja capaz de falsificar e publicar uma "cadeia alternativa"que tenha maior dificuldade total. Nesse caso, as regras de consenso fariam com que cadeia alternativa fosse adotada ao invés da

cadeia principal, a partir do ponto de bifurcação. Isso é teoricamente possível e é chamado de ataque dos 51% [Kroll et al., 2013]. Conforme amplamente debatido até agora a segurança da Bitcoin depende do consenso distribuído alcançado pela prova de trabalho. Assume-se, que não há conluio de mineradores, ou seja, nenhum grupo de coordenado de mineradores (ou um único) pode possuir mais de 50% da capacidade computacional da rede.

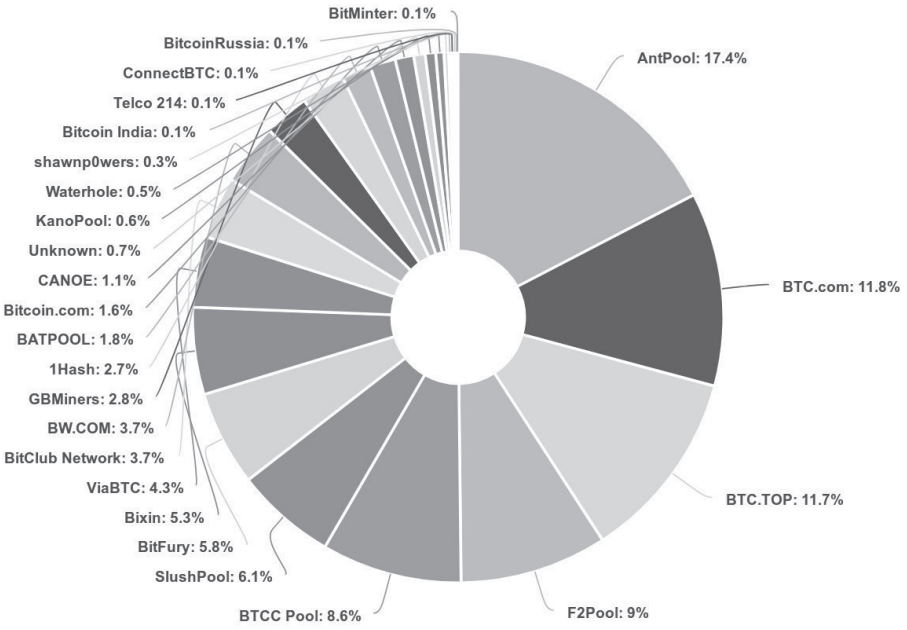


Figura 4.11. Cooperativas de mineradores. Fonte: www.blockchain.info acessado em: 21/08/2017.

No entanto, esta suposição é questionável. Primeiramente, é observado que há algum tempo que a mineração passou a ser organizada em cooperativas de mineradores (*mining pool*) que juntam esforços e compartilham as recompensas. Na Figura 4.11 é possível observar as maiores cooperativas em atividade. Em segundo lugar, não existe nenhuma entidade reguladora e nenhum minerador é obrigado a seguir o protocolo. Portanto uma cooperativa, que possua maioria de poder de computação, pode alterar as regras que são aplicadas pelo mecanismo de consenso e os mineradores que não participam da cooperativa provavelmente serão obrigados a se juntar a ela. Por exemplo, uma cooperativa, com mais de 50% do poder computacional da rede, pode escolher aceitar blocos vindos de outros mineradores na razão de 2:1, ou seja, a cada dois blocos enviados pelos nós honestos apenas um será aceito, e isso será possível pois a cooperativa possuirá o poder de manipular o consenso. Os mineradores honestos terão seus blocos ignorados e, portanto, perderão os pagamentos. O comportamento da cooperativa pode ser inclusive de realizar a negação de serviço a algum minerador ou a alguma transação, pois eles terão o poder de não incluir estas transações em nenhum de seus blocos, e caso outros mineradores o façam, eles podem gerar forks, rejeitando assim uma transação.

Os mineradores maliciosos podem desviar seu comportamento do padrão ao não divulgarem imediatamente os seus blocos recém minerados. Estes ataques são chama-

dos na literatura de *Selfish Mining* [Eyal and Sirer, 2014, Nayak et al., 2016], o qual será abordado com mais detalhes adiante. Primeiramente é preciso entender como a latência na propagação dos blocos e o tempo alvo para inclusão de novos blocos afetam o mecanismo de consenso.

- **Análise da latência de propagação dos blocos:** A propagação de mensagens na rede segue o protocolo P2P próprio. Um nó, ao receber um novo bloco, o retransmitirá aos seus vizinhos. Antes de iniciar a transmissão, ele faz uma série de verificações a fim de garantir que somente blocos válidos sejam propagados. Cada nó que recebe um novo bloco faz essas verificações independentemente. Após isso, o nó envia uma mensagem de inventário (*INV*), informando aos seus vizinhos que possui um novo bloco e sua altura. Caso estes vizinhos ainda não tenham recebido este bloco por outros nós eles enviam uma mensagem solicitando o novo bloco(*GET_DATA*). Somente então será iniciada a transmissão efetiva do novo bloco, Fig. 4.12. A soma de todos esses tempos, verificação e propagação da mensagens, durante a propagação de um bloco, é chamada de Latência de propagação dos blocos. Decker [Decker and Wattenhofer, 2013] fez a análise do tempo de propagação de 10.000 blocos, com diferentes tamanhos, e verificou que a mediana desde a criação do bloco até o recebimento por um determinado nó foi de 6,5 segundos e a média de 12,6 segundos. Outra observação interessante foi que após 40 segundos, 5% dos nós ainda não haviam recebido o novo bloco.

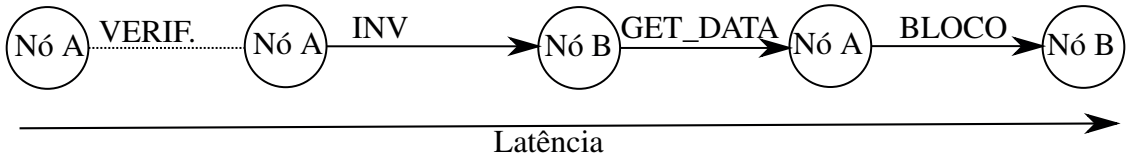


Figura 4.12. Latência de propagação dos blocos

- **Análise do tempo para inclusão de novos blocos:** O intervalo para a inclusão de novos blocos é crucial para a quantidade de *forks* observados na rede. Quanto menor for esse intervalo, maior será a quantidade de blocos gerados e consequentemente maior será a probabilidade de ocorrência de forks e de blocos órfãos.

  Decker [Decker and Wattenhofer, 2013] verificou a existência de 169 forks durante sua observação de 10.000 blocos, ou seja 1,69% dos blocos foram descartados pela ocorrência de forks. Gervais [Gervais et al., 2016] analisou o impacto da diminuição deste tempo, variando de 0,5 segundos a 25 minutos e usando 10.000 blocos em simulações no NS-3, conforme na Tabela 4.2

Caso um atacante resolva desviar do comportamento padrão e manter secretamente uma cadeia onde somente ele produza blocos, e adote uma heurística própria para divulgação destes blocos. Neste momento, todos, inclusive o atacante estariam minerando sobre o bloco *n*. Ao realizar esta ação, caso ele consiga produzir o próximo bloco, o atacante conseguirá uma vantagem em relação aos demais mineradores, mesmo sem possuir maior poder computacional, pois ele poderá iniciar o processo de mineração do

Tabela 4.2. Impacto do intervalo entre blocos na taxa de forks

Intervalo entre blocos	Taxa de Forks(%)	Mediana do tempo de propagação
0,5s	38,15	0,82
1s	26,74	0,82
2s	16,65	0,84
5s	8,64	0,89
10s	4,77	1
20s	3,2	1,21
30s	2,54	1,43
1m	2,15	2,08
2,5m	1,82	4,18
10m	1,51	14,7
25m	1,72	35,73

bloco $n+1$ antes de todos os outros mineradores. Se o atacante receber um bloco da rede ele pode decidir adotar este bloco e jogar fora seu trabalho, ou ignorar o bloco recebido e continuar minerando na cadeia privada. Como ele começou a minerar antes, há uma probabilidade de liberar bloco $n+1$ antes dos outros, gerando um fork com altura maior que a cadeia principal. Como os demais nós se comportam honestamente, eles também irão adotar esta cadeia e o atacante atingirá seu objetivo. Um esquema simples é mostrado na Figura 4.13 onde após a liberação dos blocos n e $n+1$ pelo atacante os nós honestos adotaram esta cadeia e produziram o bloco $n+2$. Este ataque é conhecido como *Selfish Mining* [Eyal and Sirer, 2014] e o atacante é chamado de "minerador egoísta".

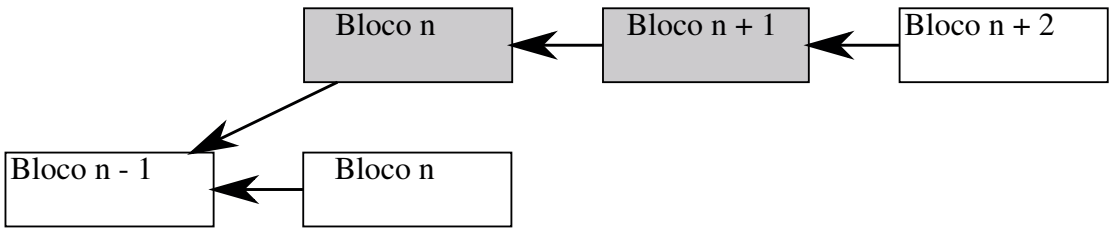


Figura 4.13. fork

Como vimos acima, uma parcela da latência de propagação dos blocos é gerada pela obrigatoriedade de verificação dos novos blocos por todos os nós. Caso um atacante controle alguns nós da rede ele pode tirar vantagem deste fato para amplificar o ataque do minerador egoísta. Estes nós controlados pelo atacante podem ser configurados para não realizar a verificação dos blocos minerados pelo atacante e retransmiti-los tão logo eles cheguem. Assim os blocos do atacante alcançam os outros nós em menor tempo que os nós honestos. Isso pode ser uma vantagem. Outra observação a respeito deste ataque é que era de se esperar que o total de blocos adicionados à cadeia seja a soma dos blocos produzidos pelo atacante e pelos nós honestos. Mas a ocorrência de forks gera blocos que serão descartados, *stale blocks*. Assim, a quantidade total de blocos incluídos é menor

que o total de blocos produzidos. Isso é importante pois se for desconsiderada a entrada de novos mineradores na rede quando os nós forem recalcular a nova dificuldade, esta será menor que a dificuldade anterior.

Em [Eyal and Sirer, 2014], Eyal faz uma análise matemática e propõe um modelo de transição de estados para capturar o melhor momento do atacante liberar seus blocos privados. Ele analisa as probabilidades de ocorrência de cada estado e chega a conclusão que para o atacante conseguir êxito, ou seja, publicar mais blocos que a cadeia honesta, seu poder de mineração deve satisfazer a seguinte inequação:

$$\frac{1-\gamma}{3-2\gamma} < \alpha < \frac{1}{2}$$

, onde α é o poder de mineração do atacante e γ é a razão dos nós honestos que escolhem minerar a cadeia desonesta. Isso é uma vantagem ao atacante, pois ele necessitará de menor poder para conseguir suplantar os nós honestos. Assim, a partir desta inequação é obtido o menor valor de α para um determinado γ. O pior caso para o atacante é quando nenhum nó honesto adota a sua cadeia, neste caso é necessário que o atacante possua um terço do poder computacional da rede.

Nayak et al. [Nayak et al., 2016] amplia a pesquisa de Eyal e verifica que o ataque proposto anteriormente não é ótimo. Ele propõe novas estratégias para aumentar o ganho do atacante, levando em conta não apenas o tamanho das cadeias, mas também seu poder computacional. Por exemplo, mesmo que o atacante esteja perdendo a corrida caso ele possua uma parcela significa de poder computacional, é melhor que ele continue a minerar em sua cadeia privada, pois ele terá uma grande chance de alcançar e ultrapassar a cadeia honesta. Outra contribuição de seu trabalho é mostrar que se o *Selfish Mining* for combinado com o *Eclipse* [Heilman et al., 2015], quando o atacante controla todas as conexões com um determinado nó, o atacante aumentará seus ganhos e surpreendentemente, com determinados parâmetros, o nó que foi alvo do *Eclipse* também conseguirá publicar mais blocos, em relação aos nós honestos.

O autor modela a mineração como um processo de decisão de Markov, que usa como informação as transições de estado quando os mineradores acham um novo bloco. Quando um nó honesto acha o novo bloco ele o publica imediatamente, enquanto o atacante mantém seus blocos ocultos. Depois, ele captura quantos blocos cada cadeia estão a frente uma da outra, e se os nós honestos estão minerando sobre a cadeia honesta ou desonesta. Suas estratégias chamadas de *Stubborn Mining*, são:

- ***Lead Stubborn:*** Vimos anteriormente que γ é um fator importante, pois quanto mais nós honestos estão minerando na cadeia do atacante menor será o α necessário. Vimos também que a latência de propagação dos blocos influencia fortemente quais blocos cada nós recebe primeiro. Assim, uma das estratégias adotadas pelo autor define que, caso o atacante esteja liderando por um bloco, tão logo os nós honestos liberem um bloco, o atacante também libera o seu. O objetivo é que seu bloco alcance uma parcela dos nós honestos que irão adotá-lo como referência para mineração, aumentando o γ e a probabilidade do atacante vencer a corrida. Caso esteja vencendo por dois ou mais ele mantém sua cadeia privada, somente revelando os blocos quando a diferença alcançar 1.

- ***Trail Stubborn:*** Quando a cadeia privada do atacante está atrás da pública. O atacante continua a mineração na cadeia privada ao invés de abandoná-la, na esperança de alcançar e ultrapassar a cadeia pública. Esta estratégia se mostra promissora caso o atacante possua um certo poder computacional.

- ***Equal Fork Stubborn:*** Esta estratégia é usada quando os nós honestos igualam a corrida e o atacante continua minerando em sua cadeia até que fique um bloco a frente, quando então ele libera sua cadeia.

O Ataque *Eclipse* [Heilman et al., 2015] é um ataque a nível de rede. Ocorre quando um atacante monopoliza todas as conexões de um determinado nó, isolando a vítima e filtrando todas as mensagens enviadas e recebidas. Como resultado, a vítima tem uma visão diferente da cadeia, pode ter seus blocos impedidos de serem incluídos na cadeia e pode ser forçada a trabalhar na cadeia do atacante. Na rede Bitcoin, os nós mantém no máximo 125 conexões com seus vizinhos, sendo 8 conexões de saída, e 117 de entrada. As conexões de saída são aquelas iniciadas pelo próprio nó, e as de entrada são as que outros nós solicitam, conforme vimos na seção que trata da rede P2P. Para armazenar o endereço destas conexões os nós usam duas tabelas, uma tabela de conexões bem sucedidas, onde são armazenadas as informações de todas as conexões estabelecidas, de entrada e de saída, e uma tabela de endereços fornecidos, por solicitação ou não. A primeira é chamada de *Tried Table* e a segunda de *New Table*.

- ***Tried Table:*** É formada por 64 recipientes que podem armazenar 64 endereços cada. Os recipientes são escolhidos da seguinte forma: Quando o nó é iniciado ele escolhe um valor aleatório SK, e calcula:

  $$Recipiente = Hash(SK, Grupo, Hash(SK, IP)\%4)\%64,$$

  onde o Grupo é o prefixo /16 do endereço IP. Quando um nó estabelece uma conexão ele então mapeia o IP do novo vizinho para um recipiente. Caso o recipiente esteja cheio, o nó então chama uma função para remover endereços do recipiente. Quatro endereços são escolhidos aleatoriamente e o mais antigo é então movido para a *New Table*.

- ***New Table:*** É formada por 256 recipientes e cada um armazena até 64 endereços. É preenchida pelos endereços removidos da *Tried Table* ou por endereços fornecidos pelos *DNS seedres* ou por mensagens *ADDR*, que são mensagens para informar novos endereços aos vizinhos. De forma similar ao item anterior, aqui também é executada uma função para mapear o recipiente e existe uma função de remoção de endereços antigos.

O ataque consiste em encher a *Tried Table* com endereços controlados pelo atacante e encher a *New Table* com endereços inválidos. Desta forma sempre que a vítima buscar uma nova conexão ela se conectará a um endereço controlado pelo atacante. A *New Table* é preenchida com endereços inválidos para que o atacante economize IPs. Assim, basta realizar duas rotinas repetidas vezes para popular as tabelas da vítima. Primeiro, para estabelecer conexões de entrada, basta que o atacante solicite uma conexão. Em

seguida, desconecta e solicita uma nova conexão com outro endereço. Isto basta para preencher a *Tried Table*. Para o atacante preencher a *New Table*, é necessaŕio enviar diversas mensagens *ADDR* com endereços inválidos para a vítima. Cada *ADDR* pode conter até 1000 endereços. São usados endereços classe C ou reservados para uso futuro, como a faixa destinada ao multicast, pois o atacante precisa de no mínimo 16384 endereços para preencher a *New Table*. Em seus experimentos, com uma *botnet* de apenas 400 bots foi capaz de popular totalmente a *New Table* e 60% da *Tried Table*, alcançando sucesso em controlar todas as conexões da vítima em 80% das vezes.

Com o intuito de exemplificar melhor o ataque *Selfish Mining* forma realizadas diversas simulações utilizando o módulo do NS-3 desenvolvido por [Gervais et al., 2016]. O módulo foi desenvolvido com o objetivo de analisar o impacto na taxa de forks, *throughput* da rede, o tempo de propagação dos blocos e ganho com gasto duplo. A conexão entre os blocos é feita com canais ponto-a-ponto, abstraindo dispositivos intermediários. Para configurar as características do canal (latência e banda) foram usados dados estatísticos de diversas fontes, como Verizon e testmy.net. A prova de trabalho é modelada atribuindo valores de poder de mineração aos nós, distribuindo estatisticamente a geração dos blocos. Os dados de entrada são: a taxa de inclusão de novos blocos, tamanho do bloco e valor do gasto duplo. Ao analisar a Figura 4.11, chega-se a conclusão que as 15 maiores cooperativas possuem 96,3% do poder de mineração. Por esse motivo, foram simulados 16 nós, representando as 15 cooperativas e os outros mineradores agrupados. Os nós honestos adotam o protocolo padrão, enquanto o atacante segue a heurística proposta pelo autor:

- **Adotar:** O Adversário adota a cadeia honesta. Isto corresponde a reiniciar o ataque, pois o atacante infere que os nós honestos possuem uma probabilidade maior de vencer a corrida.
- **Sobrepor:** Ocorre quando o atacante possui um bloco a mais que a cadeia honesta. É uma boa estratégia quando existe uma parcela dos nós honestos minerando sobre a cadeia do atacante.
- **Igualar:** O atacante publica tantos blocos quanto os publicados pelos mineradores honestos. Esta ação tem por objetivo fazer que alguns nós honestos minerem sobre a cadeia do atacante. Em seguida, o atacante pode usar a ação *Sobrepor*.
- **Esperar:** O atacante minera constantemente em sua cadeia, sem a revelar.
- **Publicar:** Corresponde a revelar sua cadeia.

Primeiramente foram realizadas 30 rodadas de simulação gerando 10.000 blocos em cada. As simulações foram feitas com 16 nós mineradores e todos seguiam o protocolo padrão, sem atacante. Como resultado uma taxa de 0,13% de forks foi obtida. Na segunda rodada de simulações um dos nós foi escolhido como atacante. Inicialmente ele possui 20% do poder de mineração total, que foi incrementado até 50%. É possível observar na Figura 4.14 que, a medida que a força do atacante é incrementada, há um considerável aumento na taxa de blocos descartados e consequentemente diminuição da quantidade de blocos na cadeia principal.

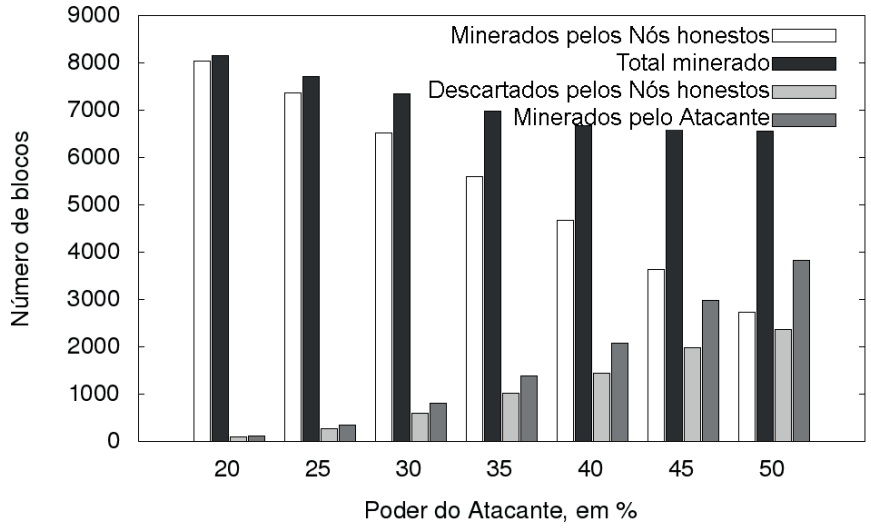


Figura 4.14. Evolução dos blocos descartados(*Stale Blocks)*

4.5. Considerações Finais, Perspectivas Futuras e Problemas em Aberto

A IoT processa e troca grandes quantidades de dados sem a intervenção humana e estes dados frequentemente possuem informações que podem ser críticas em relação a segurança e privacidade. Portanto, são alvos atraentes aos atacantes. Normalmente esses dispositivos são de baixa energia e de baixo poder computacional e devem dedicar seus parcos recursos a suas atividades principais, o que torna a tarefa de suporte a segurança e privacidade bastante desafiadora. Métodos de segurança tradicionais tendem a ser caros em termos computacionais e energéticos. Além disso, muitos dos *frameworks* de segurança são altamente centralizados e, portanto, não são necessariamente adequados para o cenário IoT devido à dificuldade de escalabilidade, e ao fato de se tornar um ponto único de falha. Consequentemente, a IoT exige uma proteção de segurança e privacidade leve, escalável e distribuído. A tecnologia Blockchain, que sustenta o Bitcoin, tem o potencial de superar esses desafios como resultado de sua natureza distribuída, segura e privada. Porém não é leve, necessitando de adaptações e otimizações.

A combinação de Blockchain e IoT pode ser bastante poderosa, pois o Blockchain pode dar resiliência a ataques e a capacidade de interagir com os pares de forma confiável e auditável. A contínua integração de Blockchain no domínio IoT causará transformações significativas em vários setores, trazendo novos modelos de negócios e nos fazendo reconsiderar como os sistemas e processos existentes são implementados.

A "cadeia de blocos"possibilita não meramente o movimento do dinheiro, mas pode também ser usada para transferir informações e a alocar recursos entre os dispositivos, habilitando o uso da Blockchain como um serviço [Swanson, 2015]. O mundo conectado pode incluir de forma útil a tecnologia Blockchain como uma camada para o que cada vez mais dispositivos (vestíveis, sensores, IoT, *smartphones, tablets, laptops*, casas, carros e cidades inteligentes) possam se beneficiar de suas características.

Blockchain, portanto, apresenta muitas promissoras oportunidades para o futuro da IoT. Os desafios, no entanto permanecem, como modelos de consenso e os custos

computacionais de verificação de transações. Mas ainda está nos estágios iniciais do desenvolvimento de cadeias de blocos, e esses obstáculos serão eventualmente superados, abrindo o caminho para muitas possibilidades.

Referências

[Ali et al., 2016] Ali, M., Nelson, J. C., Shea, R., and Freedman, M. J. (2016). Blockstack: A global naming and storage system secured by blockchains. In *USENIX Annual Technical Conference*, pages 181–194.

[Andrea, 2014] Andrea, A. (2014). *Mastering BitCoin*, volume 50.

[Ashton, 2011] Ashton, K. (2011). That 'internet of things' thing. *RFiD Journal*, 22(7).

[Atzori et al., 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54.

[Back et al., 2002] Back, A. et al. (2002). Hashcash-a denial of service counter-measure.

[Bitcoin, 2009] Bitcoin (2009). Bitcoin developer reference. https://bitcoin.org/en/developer-reference. Accessed: 2017-07-30.

[Buterin, 2014] Buterin, V. (2014). Daos, dacs, das and more: An incomplete terminology guide. *Ethereum (accessed 12 July 2016) https://blog. ethereum. org/2014/05/06/daos-dacs-das-and-more-an-incomplete-terminology-guide*.

[Cachin, 2016] Cachin, C. (2016). Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*.

[Castro and Liskov, 2002] Castro, M. and Liskov, B. (2002). Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461.

[Christidis and Devetsikiotis, 2016] Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303.

[Conoscenti et al., 2016] Conoscenti, M., Vetrò, A., and De Martin, J. C. (2016). Blockchain for the internet of things: a systematic literature review. *Porto.Polito.It*.

[Cramer et al., 1999] Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., and Rabin, T. (1999). Efficient multiparty computations secure against an adaptive adversary. In *Eurocrypt*, volume 99, pages 311–326. Springer.

[Crosby et al., 2016] Crosby, M., Pattanayak, P., Verma, S., and Kalyanaraman, V. (2016). Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2:6–10.

[Cui, 2016] Cui, X. (2016). The internet of things. In *Ethical Ripples of Creativity and Innovation*, pages 61–68. Springer.

[De Filippi and Mauro, 2014] De Filippi, P. and Mauro, R. (2014). Ethereum: the decentralised platform that might displace today's institutions. *Internet Policy Review*, 25.

[Decker and Wattenhofer, 2013] Decker, C. and Wattenhofer, R. (2013). Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE.

[Dorri et al., 2016] Dorri, A., Kanhere, S. S., and Jurdak, R. (2016). Blockchain in internet of things: Challenges and Solutions. *arXiv:1608.05187 [cs]*.

[Dorri et al., 2017a] Dorri, A., Kanhere, S. S., and Jurdak, R. (2017a). Towards an Optimized BlockChain for IoT. *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation - IoTDI '17*, 6.

[Dorri et al., 2017b] Dorri, A., Kanhere, S. S., Jurdak, R., and Gauravaram, P. (2017b). Blockchain for iot security and privacy: The case study of a smart home.

[Douceur, 2002] Douceur, J. R. (2002). The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer.

[Evans, 2011] Evans, D. (2011). A internet das coisas: como a próxima evolução da internet está mudando tudo. *CISCO IBSG*.

[Eyal, 2015] Eyal, I. (2015). The miner's dilemma. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 89–103. IEEE.

[Eyal and Sirer, 2014] Eyal, I. and Sirer, E. G. (2014). Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer.

[Fenn and LeHong, 2011] Fenn, J. and LeHong, H. (2011). Hype cycle for emerging technologies, 2011. *Gartner, July*.

[Fischer et al., 1985] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382.

[Gennaro et al., 1998] Gennaro, R., Rabin, M. O., and Rabin, T. (1998). Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111. ACM.

[Gervais et al., 2016] Gervais, A., Karame, G. O., Wüst, K., Glykantzis, V., Ritzdorf, H., and Capkun, S. (2016). On the Security and Performance of Proof of Work Blockchains. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*.

[Gilbert and Handschuh, 2003] Gilbert, H. and Handschuh, H. (2003). Security analysis of sha-256 and sisters. In *International workshop on selected areas in cryptography*, pages 175–193. Springer.

[Gubbi et al., 2013] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660.

[Guillemin et al., 2009] Guillemin, P., Friess, P., et al. (2009). Internet of things strategic research roadmap. *The Cluster of European Research Projects, Tech. Rep*.

[Hankerson et al., 2006] Hankerson, D., Menezes, A. J., and Vanstone, S. (2006). *Guide to elliptic curve cryptography*. Springer Science & Business Media.

[Heilman et al., 2015] Heilman, E., Kendler, A., Zohar, A., and Goldberg, S. (2015). Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security*, pages 129–144.

[Herrera-Joancomartí, 2015] Herrera-Joancomartí, J. (2015). Research and challenges on bitcoin anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, pages 3–16. Springer.

[Intel,] Intel. Proof of elapsed time (poet). available from: http://intelledger.github.io/.

[Jansma and Arrendondo, 2004] Jansma, N. and Arrendondo, B. (2004). Performance comparison of elliptic curve and rsa digital signatures. *nicj. net/files*.

[Kennedy and Duranleau,] Kennedy, S. and Duranleau, C. Tilepay. https://http://www.tilepay.org. Accessed: 2017-09-08.

[Kim, 2014] Kim, J. (2014). Safety, liveness and fault tolerance—the consensus choices stellar.

[Kim, 2016] Kim, T. H. (2016). A study of digital currency cryptography for business marketing and finance security. *Asia-pacific Journal of Multimedia Services Convergent with Art, Humanities, and Sociology*, 6(1):365–376.

[King and Nadal, 2012] King, S. and Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19.

[Koshy et al., 2014] Koshy, P., Koshy, D., and McDaniel, P. (2014). An analysis of anonymity in bitcoin using p2p network traffic. In *International Conference on Financial Cryptography and Data Security*, pages 469–485. Springer.

[Kroll et al., 2013] Kroll, J. A., Davey, I. C., and Felten, E. W. (2013). The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013.

[Kshetri, 2017] Kshetri, N. (2017). Can blockchain strengthen the internet of things? *IT Professional*, 19(4):68–72.

[Merkle, 1987] Merkle, R. C. (1987). A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*.

[Moser et al., 2013] Moser, M., Bohme, R., and Breuker, D. (2013). An inquiry into money laundering tools in the bitcoin ecosystem. In *eCrime Researchers Summit (eCRS), 2013*, pages 1–14. IEEE.

[Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.

[Natoli and Gramoli, 2016] Natoli, C. and Gramoli, V. (2016). The blockchain anomaly. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, pages 310–317. IEEE.

[Nayak et al., 2016] Nayak, K., Kumar, S., Miller, A., and Shi, E. (2016). Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 305–320. IEEE.

[Nguyen et al., 2015] Nguyen, K. T., Laurent, M., and Oualha, N. (2015). Survey on secure communication protocols for the Internet of Things. *Ad Hoc Networks*, 32.

[Ouaddah et al., 2017] Ouaddah, A., Elkalam, A. A., and Ouahman, A. A. (2017). FairAccess : a new Blockchain-based access control framework for the Internet of Things. *Security and Communication Networks*, 9.

[Panikkar et al., 2014] Panikkar, S., Nair, S., Brody, P., and Pureswaran, V. (2014). Adept: An iot practitioner perspective. *IBM Institute for Business Value*.

[Peña-López et al., 2005] Peña-López, I. et al. (2005). Itu internet report 2005: the internet of things.

[Pilkington, 2015] Pilkington, M. (2015). Blockchain technology: principles and applications. *Browser Download This Paper*.

[Pilkington, 2016] Pilkington, M. (2016). Blockchain technology: principles and applications. research handbook on digital transformations, edited by f. xavier olleros and majlinda zhegu.

[Popov, 2016] Popov, S. (2016). The tangle. *Available electronically at http://iotatoken.com/IOTA Whitepaper. pdf*.

[Recommendation, 2012] Recommendation, Y. (2012). 2060 «overview of internet of things». *ITU-T, Geneva*.

[Shamir, 1979] Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 22(11):612–613.

[Spagnuolo et al., 2014] Spagnuolo, M., Maggi, F., and Zanero, S. (2014). Bitiodine: Extracting intelligence from the bitcoin network. In *International Conference on Financial Cryptography and Data Security*, pages 457–468. Springer.

[Stallings, 1995] Stallings, W. (1995). *Network and internetwork security: principles and practice*, volume 1. Prentice Hall Englewood Cliffs.

[Strategy and Unit, 2005] Strategy, I. and Unit, P. (2005). Itu internet reports 2005: The internet of things. *Geneva: International Telecommunication Union (ITU)*.

[Swan, 2015a] Swan, M. (2015a). *Blockchain: Blueprint for a new economy*. "O'Reilly Media, Inc.".

[Swan, 2015b] Swan, M. (2015b). Blockchain thinking: The brain as a dac (decentralized autonomous organization). In *Texas Bitcoin Conference*, pages 27–29.

[Swanson, 2015] Swanson, T. (2015). Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. *Report, available online, Apr*.

[Szabo, 1994] Szabo, N. (1994). Smart contracts. *Unpublished manuscript*.

[Szabo, 1997] Szabo, N. (1997). The idea of smart contracts. Nick Szabo's Papers and Concise Tutorials.

[Valenta and Rowan, 2015] Valenta, L. and Rowan, B. (2015). Blindcoin: Blinded, accountable mixes for bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 112–126. Springer.

[Vasin, 2014] Vasin, P. (2014). Blackcoin's proof-of-stake protocol v2.

[Wood, 2014] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151.

[Wörner and von Bomhard, 2014] Wörner, D. and von Bomhard, T. (2014). When your sensor earns money. *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing Adjunct Publication - UbiComp '14 Adjunct*.

[Wörner and von Bomhard, 2014] Wörner, D. and von Bomhard, T. (2014). When your sensor earns money: exchanging data for cash with bitcoin. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 295–298. ACM.

[Yao, 1982] Yao, A. C. (1982). Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE.

[Yue et al., 2016] Yue, X., Wang, H., Jin, D., Li, M., and Jiang, W. (2016). Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control. *Journal of medical systems*, 40(10):218.

[Zhang and Wen, 2015] Zhang, Y. and Wen, J. (2015). An iot electric business model based on the protocol of bitcoin. In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pages 184–191. IEEE.

[Zimmermann, 1995] Zimmermann, P. R. (1995). *The official PGP user's guide*. MIT press.

[Zyskind et al., 2015a] Zyskind, G., Nathan, O., and Pentland, A. (2015a). Enigma: Decentralized Computation Platform with Guaranteed Privacy. *arXiv:1506.03471 [cs]*.

[Zyskind et al., 2015b] Zyskind, G., Nathan, O., and Pentland, A. S. (2015b). Decentralizing privacy: Using blockchain to protect personal data. *Proceedings - 2015 IEEE Security and Privacy Workshops, SPW 2015*.

SBSeg 2017

XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

6 A 9 DE NOVEMBRO - BRASÍLIA - DF

sbseg2017.redes.unb.br

Realização

Apoio

Patrocínio

Agência Brasileira do ISBN
ISBN 978-85-7669-410-6
9 788576 694106