XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

22 a 25 de outubro de 2018
Natal - RN - Brasil

MINICURSOS

Editora
Sociedade Brasileira de Computação - SBC

Organizadores
Aldri Luiz dos Santos, UFPR
Marjory Da Costa Abreu, UFRN
Carlos Eduardo Da Silva, UFRN

Realização
UFRN, Universidade Federal do Rio Grande do Norte

Promoção
SBC – Sociedade Brasileira de Computação

Capa e Editoração: Gustavo Alves Bezerra e Patrícia Pontes Cruz

Dados Internacionais de Catalogação na Publicação (CIP)

Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (18. : 2018 : Natal, RN)

Minicursos [do] XVIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, 22 a 25 de outubro de 2018, Natal, RN, Brasil ; organizadores, Aldri Santos, Marjory Da Costa Abreu, Carlos Eduardo da Silva. – Natal, RN : Sociedade Brasileira de Computação, 2019.

xviii, 194 p. ; il. ; 23 cm

Acima do título: SBSeg 2018
ISBN: 978-65-87003-88-7

1. Ciência da Computação – Eventos 2. Informática – Eventos 3. Sistemas de Informação – Eventos 4. Segurança de Sistemas – Eventos 5. Segurança de Informação – Eventos I. Santos, Aldri II. Da Costa Abreu, Marjory III. Da Silva, Carlos Eduardo IV. Título V. Título: Minicursos SBSeg 2018.

RN/UF/BSE-CCET

Ficha catalográfica elaborada por Joseneide Ferreira Dantas - CRB-15/324 Biblioteca Setorial Prof. Ronaldo Xavier de Arruda - CCET da UFRN

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
Aldri Luiz dos Santos, UFPR
Igor Monteiro Moraes, UFF
Pedro Braconnot Velloso, UFRJ
Rafael Timóteo de Souza Júnior, UnB
Paulo Lício de Geus, UNICAMP
Carlos Eduardo da Silva, UFRN
Marcos Antônio Simplício Júnior, USP
Edna Dias Canedo, UnB
Marjory da Costa Abreu, UFRN

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 despertam grande interesse entre acadêmicos e profissionais.

Nesta edição do SBSeg (2018), 14 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 em torno de 28%. O Comitê de Avaliação dos Minicursos foi composto por 9 renomados pesquisadores para a elaboração dos pareceres. Cada proposta recebeu ao menos 3 pareceres, gerando ao todo mais 42 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 2018.

Este livro reúne 4 capítulos produzidos pelos autores das propostas de minicursos aceitas. O Capítulo 1 discute os principais conceitos, ameaças e contramedidas no contexto de redes IoT convencionais e IoT baseadas em SDN, bem como a análise do comportamento da rede através da extração de análise de dados. O Capítulo 2 aborda o uso programático de criptografia assimétrica por desenvolvedores de software com pouca experiência em segurança da informação e criptografia, mostrando os bons e maus usos da criptografia assimétrica, por meio de exemplos reais, contraexemplos, trechos de código e programas ilustrativos em Java. O Capítulo 3 introduz o uso de suporte de hardware para a inspeção de binários e sistemas, abrangendo desde o funcionamento dos processadores modernos até a aplicação de suas instruções especiais para a análise de binários e sistemas de forma transparente. O Capítulo 4 mostra como os contratos inteligentes, códigos residentes na *blockchain* que automatizam processos de múltiplas etapas, podem ser usados na comercialização automática de energia, assim como as iniciativas, os desafios e as oportunidades de pesquisa da tecnologia *blockchain* no setor elétrico. 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 realizado no processo de avaliação e seleção dos minicursos. Gostaria de também agradecer aos Coordenadores Gerais do SBSeg 2018, Marjory da Costa Abreu (UFRN) e Carlos Eduardo da Silva (UFRN), pela disponibilidade e suporte ao longo de todo o processo e pela confiança depositada em mim para coordenar estes minicursos. Finalmente, agradeço aos autores por terem prestigiado este evento ao submeterem suas propostas de minicursos.

Aldri Luiz dos Santos, UFPR
Coordenador de Minicursos do SBSeg 2018

Comitês

Comitê de Organização

Coordenação Geral
Marjory Da Costa Abreu – UFRN
Carlos Eduardo Da Silva – UFRN

Coordenação de Minicursos
Aldri Luiz dos Santos, UFPR

Comitê de Programa dos Minicursos

Alberto Schaeffer-Filho, UFRGS
Altair Santin, PUCPR
Dorgival Guedes, UFMG
Edna Canedo, UnB
Eduardo da Silva, IFC
Eduardo Luzeiro Feitosa, UFAM
Luis Kowada, UFF
Michelle Silva Wangham, UNIVALI

Sumário

Capítulo 1

Ameaças de Segurança, Defesas e Análise de Dados em IoT Baseada em SDN

Nelson G. Prates Jr.[1], **Mateus Pelloso**[1,3], **Ricardo T. Macedo**[1,2], **Michele Nogueira**[1]

Centro de Ciência de Segurança Computacional e
Universidade Federal do Paraná[1]
Universidade Federal de Santa Maria[2]
Instituto Federal Catarinense[3]

Abstract

Software Defined Networks (SDN) have been proposed to solve problems related to scalability, management and mobility in the traditional network model. Hence, emerging network paradigms, –Internet of Things (IoT) and Internet of Everything (IoE),– have integrated SDN into their infrastructure for benefiting from its advantages. However, these networks are susceptible to threats against the authenticity, confidentiality, integrity and availability of data and/or network services. This chapter presents the main threats and defenses in SDN-based IoT, following a theoretical and a practical perspectives. The theoretical perspective introduces the main concepts, threats and countermeasures of conventional IoT and SDN-based IoT. The practical perspective addresses a study case related to Denial of Service (DoS) attacks, involving simulation and network traffic analysis based on the extraction and analysis of data by statistical learning methods. This chapter offers an opportunity to identify the main research challenges related to SDN-based IoT, and to the analysis of data focused on network security.

Resumo

As Redes Definidas por Software (SDN) foram propostas para solucionar problemas relacionados à escalabilidade, gerenciamento e mobilidade do modelo de redes tradicional. Desta forma, paradigmas emergentes de redes, – Internet das Coisas (IoT) e Internet de Todas as Coisas (IoE), – vêm integrando a SDN em suas infraestruturas para usufruir dos seus benefícios. No entanto, estas redes são suscetíveis a ameaças contra a confidencialidade, integridade e disponibilidade dos dados e/ou serviços da rede. Este capítulo apresenta as principais ameaças e defesas em redes IoT baseadas em SDN, seguindo uma perspectiva teórica e uma prática. A parte teórica introduz os principais conceitos, ameaças e contramedidas no contexto de redes IoT convencionais e IoT baseadas em SDN. A

perspectiva prática aborda um estudo de caso sobre ataques de negação de serviço, envolvendo a simulação e a análise do tráfego de rede através da extração de dados usando métodos de aprendizagem estatística. Este capítulo oferece a identificação dos principais desafios de pesquisa na segurança da IoT e da IoT baseada em SDN, e na análise de dados focando na segurança de redes.

1.1. Introdução

O paradigma de Internet de Todas as Coisas (*Internet of Everything* – IoE) vêm expandindo o conceito de Internet das Coisas (*Internet of Things* - IoT) visando proporcionar serviços ainda mais relevantes para as pessoas. A IoT, muitas vezes considerada como um sinônimo de IoE, representa uma vasta gama de dispositivos (coisas) capazes de se conectar à Internet para prover serviços inteligentes por meio da troca de uma grande quantidade de dados em tempo real [Atzori et al. 2010]. Estas coisas podem ser, por exemplo, computadores de bordo de um veículo, *smartphones*, refrigeradores ou fontes de energia elétrica. No entanto, a IoE estende o conceito de IoT ao considerar a associação de pessoas, processos, dados e coisas [Schatten et al. 2016b]. Por meio desta associação, a IoE explora a íntima relação entre estas entidades, podendo prover serviços ainda mais relevantes para as pessoas e gerando oportunidades econômicas sem precedentes para empresas, indivíduos e países [Miraz et al. 2015]. Entretanto, o advento da IoE está intrinsecamente atrelado com a solução de questões abordadas pela IoT [Iannacci 2018], sendo um dos principais pontos o gerenciamento do número massivo de coisas e sua integração com Internet atual [Lamaazi et al. 2014].

A integração da IoT com as Redes Definidas por Software (do inglês, *Software-Defined Network* - SDN) persegue uma forma de resolver o problema de gerenciamento das coisas na IoT. A SDN propõe um modelo de rede que desacopla o plano de controle (papel do controlador SDN) dos comutadores/dispositivos de encaminhamento (*switches*). Ao integrar a IoT com a tecnologia SDN, os procedimentos de gerenciamento dos dispositivos da IoT são centralizados no plano de controle da rede SDN, proporcionando como principal vantagem uma simplificação significativa do gerenciamento da rede [Bera et al. 2017]. A Figura 1.1 ilustra esta integração. As aplicações da IoT, tais como identificação, sensoriamento, comunicação e computação, são consideradas para a rede SDN como logicamente localizadas acima do limite norte da interface do controlador SDN. Os *switches* atuam na interface sul apenas encaminhando os dados destas aplicações conforme as regras instaladas pelo controlador [Bizanis and Kuipers 2016]. Deste modo, as funções de gerenciamento da rede são concentradas no controlador, permitindo a programação das regras da rede em tempo real e possibilitando rápidas adaptações da topologia da rede diante da dinamicidade dos dispositivos da IoT.

Todavia, em paralelo a estas características e avanços, diferentes relatórios de incidentes de segurança vêm apresentando um crescimento significativo do número de ataques e ameaças contra as redes de computadores tradicionais no cenário nacional e global. As estatísticas disponibilizadas pelo CERT.br, por exemplo, mostram que em 2015 foram relatados cerca de 722 mil incidentes de segurança em 2015, aumentando para 647 mil em 2016 e atingindo a casa de 833 mil em 2017 [CERT.br 2018]. Considerando a escala global, existem estatísticas ainda mais alarmantes neste mesmo período. Em 2015, a Cisco reportou que cerca de 43% dos setores públicos falhavam em prestar serviços

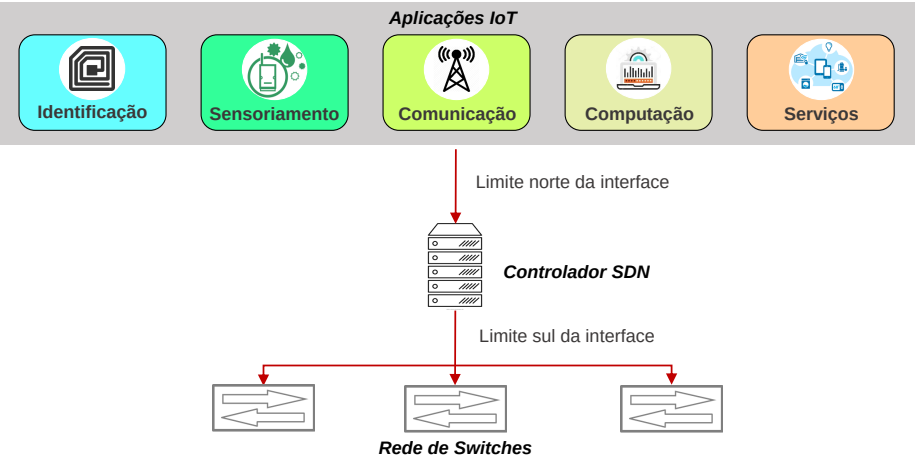


Figura 1.1: Rede IoT Baseada em SDN

de segurança para suas infraestruturas [Cisco 2018]. Em 2016, a Akamai confirmou 19 mega ataques, sendo dois deles os maiores ataques de negação de serviços já registrados, alcançando 623 Gbps e 555 Gbps, respectivamente [Akamai 2016]. Em 2017, os ataques envolvendo *ransomwares* cresceram 36% e 6,5% das pessoas foram vítimas de fraudes de identidades, resultando em prejuízos de 16 bilhões de dólares [Mason 2018]. Em 2018, este cenário acrescentou uma nova característica, pois foi registrado um aumento de 600% de ataques envolvendo dispositivos da IoT [Symatec 2018].

Em decorrência da presença de ameaças iminentes contra as redes de computadores tradicionais, cresce a preocupação com a segurança na integração entre IoT e SDN [Flauzac et al. 2015]. Os principais desafios de segurança para o advento da IoT abrangem questões de privacidade, autorização, verificação, controle de acesso, configuração de sistemas, armazenamento de informações e aspectos de gerenciamento [Alaba et al. 2017]. Em relação às SDNs, pesquisas acadêmicas revelam que as mesmas são suscetíveis a ameaças contra a autenticidade, a confidencialidade, a integridade e a disponibilidade dos dados e/ou componentes da rede [Scott-Hayward et al. 2016]. A captura e a análise dos pacotes do plano de controle das redes SDN podem fornecer informações privilegiadas ao atacante sobre as configurações dos *switches*, violando o princípio de confidencialidade. Ao obter conhecimento sobre a rede, o atacante pode falsificar pacotes para gerar um ataque de negação de serviço no controlador, ferindo os princípios de integridade dos pacotes e de disponibilidade do controlador, comprometendo consequentemente as aplicações da IoT.

Este capítulo apresenta os conceitos, a história, as principais ameaças e as defesas existentes para IoT tradicional e para as redes IoT baseadas em SDN, bem como as principais vantagens na relação entre estes dois modelos. O capítulo segue uma abordagem teórico/prática. A parte teórica introduz os fundamentos sobre a IoT e as SDNs, o histórico, as principais ameaças e as defesas. Também, a parte teórica apresenta como estes modelos se relacionam e porque essa relação traz vantagens significativas em termos de gerência de rede. Para cada uma dessas tecnologias existem uma diversidade de padronizações, neste documento optamos por seguir as tecnologias normatizadas pelas instituições e grupos de padronização pesquisas mais influentes encontrados na literatura, tais como a ***I**nternet **E**ngineering **T**ask **F**orce* (**IETF**), a ***I**nternational **O**rganization for*

***S**tandardization* (**ISO**) e o ***I**nstitute of **E**lectrical and **E**lectronics **E**ngineers* (**IEEE**). A parte de perspectiva prática descreve um estudo de caso da análise do comportamento de um ataque de negação de serviço, detalhando as ferramentas utilizadas para a simulação de uma rede sob ataque e a análise do seu tráfego. Neste capítulo, são descritas didaticamente as experiências do Centro de Ciência de Segurança Computacional (CCSC)[1] em extrair dados da rede, analisar esses dados através de métodos de aprendizado estatístico e concluir sobre os comportamentos da rede e de ataques.

O capítulo tem como objetivo incentivar a comunidade a desenvolver soluções de segurança para a IoT, destacando como as SDNs podem apoiar neste objetivo. Através deste capítulo, esperamos que os leitores possam compreender os principais conceitos e como relacionar a IoT e as SDNs, e identificar desafios de pesquisa em aberto quanto às ameaças e contramedidas existentes. Além disto, os leitores conhecerão um estudo de caso sobre a análise do comportamento da rede frente a um ataque de negação de serviço e terão a oportunidade de aprender sobre o uso de ferramentas para extração e análise de dados da rede, assim como métodos atuais de análise.

O restante do capítulo está organizado como segue. A Seção 1.2 introduz os principais conceitos e terminologias da IoT, sua arquitetura, a realação com a computação em nuvem e o conceito da IoE. A Seção 1.3 apresenta a história das SDNs, as principais definições e terminologias, a infraestrutura incluindo o protocolo *OpenFlow* e também como a IoT se relaciona com as SDNs para solucionar os desafios encontrados por estas redes. A Seção 1.4 descreve as principais ameaças encontradas nas SDNs, na IoT e IoT baseadas em SDN. A Seção 1.5 detalha as principais defesas. A Seção 1.6 finaliza o capítulo abordando uma perspectiva prática ao descrever a condução de uma simulação de uma rede IoT baseada em SDN sob ataque DDoS, mostrando a análise dos dados extraídos da rede em busca de padrões que revelem o ataque realizado.

1.2. Internet das Coisas e Internet de Todas as Coisas

A evolução das tecnologias de redes de sensores sem fio resultou na inclusão da IoT no cotidiano do cidadão moderno. Estudos realizados pela Cisco estimam que em 2021 haverá cerca de 8,3 bilhões de dispositivos pessoais portáteis conectados à rede mundial de computadores, gerando um tráfego de até 49 exabytes por mês [Cisco 2016]. Este novo modelo de rede conecta os mais variados dispositivos computacionais à Internet, exigindo redes com flexibilidade para acomodar um alto nível de escalabilidade e heterogeneidade. Além disso, esses dispositivos são distribuídos oferecendo diferentes tipos de aplicações em tempo real [Bera et al. 2017]. Estes aspectos somados à escalabilidade, heterogeneidade e aos serviços distribuídos geram grandes quantidades de dados e potencializam desafios relacionados a *Big Data* e dados em *Stream*. Esses desafios demandam serviços que atualmente vão além das conexões entre os dispositivos compondo a IoT, expandindo as possibilidades de desenvolvimento de soluções tanto para tecnologias nas camadas mais baixas da rede até a camada de aplicação, mas também soluções que considerem o comportamento das pessoas e dos processos envolvidos, ou seja, a IoE. Esta perspectiva tem aquecido o mercado, trazendo expectativas de investimentos em torno de US$ 1,7 trilhões em 2020 [Tayyaba et al. 2017].

[1]Centro de Ciência de Segurança Computacional (CCSC): Página Web `www.ccsc-research.org`

As principais soluções para os desafios da IoT estão fundadas em tecnologias como a computação em nuvem. O principal desafio de uma rede IoT consiste na limitação da capacidade dos recursos computacionais nos dispositivos de borda (dispositivos finais). A computação em nuvem trata os desafios relacionados ao *Big Data*, fornecendo recursos computacionais pela Internet e seguindo o modelo cliente/servidor. No entanto, o desafio de gerenciar a escalabilidade permanece. Para tratá-lo, uma tentativa consiste na proposição do conceito de computação em névoa (*fog computing*). Esta abordagem simplifica a disseminação dos dados e serviços da nuvem, aproximando-os da borda. Essas soluções requerem a definição e padronização de um conjunto de protocolos e tecnologias de rede, as quais podem aumentar a complexidade dessas redes dificultando a gerência da rede e agravando o problema de heterogeneidade. Esse fato gera uma demanda por sistemas de gerência igualmente escaláveis capazes de simplificar o ônus da manutenção da rede ao promover a heterogeneidade.

Esta seção detalha os principais conceitos da IoT, sua respectiva relação com a computação em nuvem e o advento da IoE. A Subseção 1.2.1 apresenta a arquitetura conceitual de rede para IoT e descreve como a IoT se conecta com a Internet. A Subseção 1.2.2 aborda a relação entre a computação em nuvem e IoT. A Subseção 1.2.3 introduz a relação entre IoT e IoE.

1.2.1. Arquitetura Conceitual da IoT

A IoT e a IoE suportam uma ampla variedade de aplicações e vêm sendo apoiadas pela evolução das tecnologias de rede, protocolos, meios de comunicação, dispositivos e serviços de rede. Esta evolução vem incentivando a proposição de diversas arquiteturas de redes IoT [Atzori et al. 2010, Khan et al. 2012, Bandyopadhyay and Sen 2011]. A arquitetura de rede conceitual mais popular está dividida em camadas de **aplicação**, **rede** e **percepção** [Palattella et al. 2013], como ilustra a Figura 1.2. A camada de aplicação utiliza as informações adquiridas e tratadas respectivamente pelas camadas de percepção e rede. Entretanto, a aplicação acaba por determinar como os dispositivos da IoT são organizados. A camada de rede realiza a comunicação dispositivo-a-dispositivo. A camada de percepção compreende dispositivos sensores responsáveis por coletar dados e por interagir com o ambiente físico. Esta organização em camadas suporta o desempenho de atividades direcionadas aos requisitos dos diferentes contextos, como casas inteligentes, cidades inteligentes e outros.

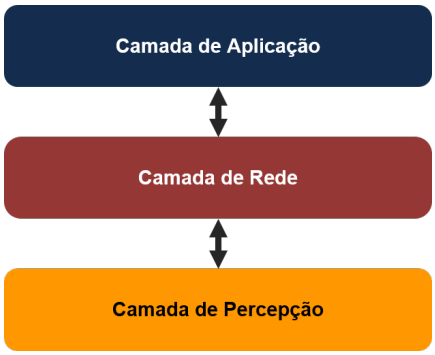


Figura 1.2: Arquitetura IoT [Palattella et al. 2013]

Camada de Aplicação

A camada de aplicação na arquitetura conceitual da IoT, proposta por [Palattella et al. 2013], oferece um nível de abstração que engloba protocolos responsáveis por oferecer serviços aos usuários finais e uma interface entre a camada de aplicação e a camada de rede. Esta camada orienta como os dispositivos devem se comunicar para atender aos requisitos das aplicações suportadas pela IoT. Além disso, a camada de aplicação oferece serviços de gerenciamento da rede e de seus dispositivos. As funcionalidades de gerenciamento da rede assumem que a camada de aplicação está ciente do cenário e das aplicações apoiadas pela IoT. As aplicações em si e os cenários em que a IoT está inserida guiam quais tipos de sensores e atuadores são necessários para atender aos seus requisitos. A camada de aplicação, de posse dos dados coletados, podem utilizar técnicas de inteligência artificial e outras para analisar o estado da rede, tomar decisão e desempenhar atividades que cumpram os objetivos dos serviços oferecidos aos usuários.

Além disso, com base nos dados coletados, a camada de aplicação pode coordenar ações associadas ao usuário, tais como emitir alertas, ligar, desligar ou coordenar uma atividade através de um ou mais dispositivos. Na literatura, os principais usos de IoT estão associados ao projeto de soluções para cidades inteligentes e casas inteligentes [Feng et al. 2017, Pradhan et al. 2018, Vlacheas et al. 2013]. Em cada uma dessas aplicações, os dispositivos são heterogêneos, apresentando diferentes características em termos de capacidade computacional, energética e tecnologia de comunicação. Por exemplo, em [Feng et al. 2017], os autores propuseram integrar a IoT com um **S**istema **C**ognitivo **D**inâmico (CDS) e desenvolveram um estudo de caso sobre um cenário de casas inteligentes. Nesse trabalho, os autores apresentaram um exemplo em que o ambiente compreende dispositivos inteligentes, como a televisão, o sofá e o ar condicionado. No cenário, os dispositivos estão habilitados a coletar dados para identificar quando um usuário está cansado e prestes a dormir, analisando seus movimentos, gestos e temperatura do corpo. Este sistema ciber-humano (*cyber-human system – CHS*) permite que os dispositivos inicializem ações apropriadas para prover conforto ao usuário. Desta forma, a televisão pode baixar o volume, o ar condicionado pode ajustar a temperatura e o sofá se inclinar. Nessa estrutura, o CDS organiza técnicas computacionais para desempenhar um conjunto de funções baseadas nas capacidades e características humanas, com o objetivo de monitorar, organizar os dados, tomar decisões e interagir com o ambiente de forma coordenada. Esta técnica exige alta disponibilidade de recursos computacionais. Entretanto, os dispositivos possuem recursos limitados e, para contornar esta limitação, os autores sugerem que o CDS opere na nuvem computacional.

Com relação aos principais protocolos da camada de aplicação, podemos citar o ***Co**nstrained **A**pplication **P**rotocol* (CoAP) [Shelby et al. 2014] e o ***M**essage **Q**ueuing **T**elemetry **T**rans-port* (MQTT) [Banks and Gupta 2014]. O CoAP foi padronizado pela IETF (RFC 7252), sendo destinado às aplicações web para dispositivos com baixa capacidade computacional e energética. Sua principal característica consiste em utilizar o modelo REST (**RE**presentational **S**tate **T**ransfer), o qual permite que os sistemas solicitantes acessem e manipulem representações textuais dos recursos através de comandos básicos como GET, PUT, POST e DELETE. Diferente do CoAP, o protocolo MQTT utiliza o modelo *Publish/Subscribe*, onde os dispositivos geradores de dados (comummente sensores – *Publishers*) os enviam para um *broker* (normalmente um *gateway*), que por

sua vez os encaminha para os dispositivos interessados (*subscribers*). A ISO/IEC 20922 padronizou o MQTT e definiu regras destinadas às camadas inferiores para o controle da Qualidade do Serviço.

Camada de Rede

A camada de rede na arquitetura conceitual da IoT proposta por [Palattella et al. 2013] é responsável por controlar como as mensagens são transmitidas do emissor para o receptor e abstrai três principais tipos de comunicação: *i*) uma comunicação direta entre dois dispositivos da IoT, *ii*) a comunicação de um dispositivo da IoT (*gateway*) com outro dispositivo através da Internet, e a *iii*) a comunicação virtual fim-a-fim entre dois dispositivos. Ou seja, a camada de rede do modelo conceitual para a IoT proposto por [Palattella et al. 2013] acaba por agregar funcionalidades e serviços presentes nas camadas física/enlace, rede e transposte da arquitetura TCP/IP. Para cada tipo de comunicação mencionado, diferentes protocolos e padrões de comunicação são encontrados na literatura para transmitir mensagens. Essas mensagens por sua vez podem ser apenas de controle e coordenação da rede ou podem ser mensagens que carregam dados coletados na camada de percepção. Neste capítulo, nos referimos ao primeiro tipo de mensagem como *mensagem de controle* e ao segundo tipo, como *mensagem de dados*.

Salienta-se que a camada de rede precisa considerar as limitações de recursos presentes em grande parte dos dispositivos da IoT. Este aspecto é relevante principalmente para a comunicação direta entre dois dispositivos da IoT. Os principais padrões de comunicação direta entre dois dispositivos da IoT consistem no IEEE 802.15 e no *Bluetooh Low Energy* (LE). A Cisco estimou que 46% das conexões seguirão o IEEE 802.14 e *Bluetooth*. Assim como o IEEE 802.15, o *Bluetooth* LE é uma tecnologia de comunicação sem fio arquitetada para operar com dispositivos de baixa capacidade energética, ou seja, prioriza a economia no consumo de energia. Esses padrões atendem às especificidades das redes sem fio com uma abrangência de área pessoal (*Wireless Personal Area Network*). Em geral, esse tipo de comunicação se dá entre dispositivos com baixo recurso energético. Esses padrões especificam a camada física e o controle de acesso ao meio.

Como na IoT mais de um meio de comunicação físico pode estar presente, soluções que tornem a heterogeneidade transparente são necessárias. Devido a este fato, para se conectar com a Internet alguns dispositivos precisam do suporte de um outro protocolo ou de uma adaptação nos protocolos de comunicação direta para seguir os padrões estipulados pela camada de rede da arquitetura TCP/IP. Para solucionar esse problema surgiu o padrão que implementa IPv6 sobre Redes Sem Fio de Área Pessoal e Baixo Consumo de Energia (do inglês, *IPv**6** over **L**ow **P**o**w**er **W**ireless **P**ersonal **A**rea **N**etworks* - 6LoWPAN). O 6LoWPAN foi padronizado pela IETF (RFC 4919) e pode ser implementado diretamente nos dispositivos ou, no caso de dispositivos que utilizam outros meios de comunicação, através de um *gateway*. A principal função do 6LoWPAN é permitir que a estrutura de rede de curto alcance e pouca disponibilidade de banda se comunique com dispositivos na Internet através do protocolo IPv6. A principal técnica utilizada é a definição e compressão dos cabeçalhos IPv6, diminuindo o tamanho dos pacotes. Isso permite que mais dados sejam inseridos no *payload* dos pacotes sem exceder a ***M**aximum **T**ransmission **U**nit* (MTU) do protocolo responsável pela camada física e enlace na IoT.

As abstrações propostas pelo 6LoWPAN habilitam os dispositivos a estabelecer conexões entre dispositivos com tecnologias diferentes, utilizando o 6LoWPAN como tecnologia de comunicação comum. O 6LoWPAN foi desenvolvido para operar sobre o IEEE 802.15.4. No entanto, a RFC 7668 normatiza a integração do IPv6 sobre o *Bluetooth LE* [Nieminen et al. 2015]. A Figura 1.3 compara o projeto original original do *Bluetooth* LE com o projeto de integração do 6LoWPAN e o *Bluetooth LE*. Conforme ilustra a Figura 1.3(a), o projeto original consistia na camada física, camada de enlace e uma interface modo de teste direto. A camada física transmite e recebe os pacotes atuais. A camada de enlace é responsável por prover o controle de acesso ao meio, estabelecimento de conexões, controle de erros e controle de fluxo. A interface de modo de teste direto é usada somente em testes. Nas camadas superiores encontram-se o controle do enlace lógico e o protocolo de adaptação, o Protocolo de Atributo, o Gerenciador de segurança, o Perfil de Atributo Genérico e o Perfil de Acesso Genérico. A Interface Controladora do Host (HCI) separa as camadas inferiores, sendo geralmente implementada na pilha do host. A Figura 1.3(b) mostra os componentes herdados da pilha original do *Bluetooth*, destacando os componentes que agregam o 6LoWPAN.

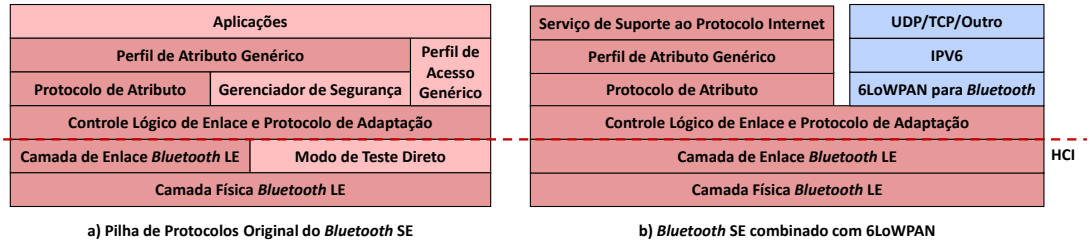


Figura 1.3: Comparação entre o *Bluetooth* e o 6LoWPAN [Nieminen et al. 2015]

Os dispositivos da IoT com menor capacidade, tais como os sensores, não realizam todas as funções para o tratamento dos dados sensoreados. Para isso, eles precisam transmiti-los para um dispositivo com maior capacidade computacional para então processar os dados necessários. Esses dispositivos com maior capacidade podem estar dentro da própria IoT ou podem estar fora da mesma como por exemplo em um ambiente em nuvem. Desta forma, são necessárias comunicações de múltiplos saltos e a comunicação com a Internet. Diante desta necessidade, a IETF criou o grupo ROLL (*Routing Over Low Power and Lossy Networks*). Este grupo definiu o RPL (*IPv6 Routing Protocol for Low-Power and Lossy Networks*)(RFC 6550) [Thubert et al. 2012], um protocolo de roteamento para redes IoT fundado no IPv6. Este protocolo suporta os três tipos de comunicação especificados.

O RPL constrói uma rede com topologia em árvore como um Gráfico Acíclico Direcionado Orientado ao Destino (do inglês, **D**estination-**O**riented **D**irected **A**cyclic **G**raph - DODAG). Um DODAG está ligado a um ou mais nós raízes, que servem como um ponto de trânsito que vincula a rede IoT às redes IPv6. Então as rotas são traçadas sempre de/-para um nó raiz. O RPL possui quatro valores de instância, ou seja, ID da instância, ID DODAG, número da versão DODAG e classificação. Esses quatro valores de instâncias são usados para manter uma topologia DODAG. Em particular, qualquer nó no RPL pode ser identificado exclusivamente com esses quatro valores de instâncias. A instância RPL é usada para identificar os DODAGs compartilhando o mesmo tipo de serviço. Os nós

conectados à mesma raiz têm o ID DODAG comum. O número da versão DODAG é atualizado conforme a topologia das alterações do DODAG. A classificação é usada para representar a distância relativa de um nó a raiz e é um parâmetro muito importante para nós no RPL. Os nós com as menores classificações indicam que eles estão mais próximos da raiz. O RPL também define rotas descendentes como as rotas da raiz para outros nós, enquanto as rotas ascendentes são definidas como rotas de nós para raiz. Para traçar as rotas e montar o grafo, o RPL implementa um padrão de trocas de mensagens que realizam a manutenção das rotas e inclusão de novos nós na estrutura [Zhao et al. 2017].

A comunicação virtual fim-a-fim segue um dos dois modelos: orientada à conexão e não-orientada à conexão. Quando quando a aplicação exige maior confiabilidade, o protocolo ***T**ransmition **C**ontrol **P**rotocol* (TCP) da Internet é empregado para oferecer uma comunicação virtual fim-a-fim orientada à conexão. O TCP foi padronizado pela IETF (RFC 793). Ele garante que os dados sejam entregues entre dois dispositivos. O ***U**ser **D**atagram **P**rotocol* (UDP) (RFC 768) tem como principal característica oferecer uma comunicação virtual fim-a-fim não orientada à conexão. Em geral, ele pode ser aplicado quando a aplicação não possui restrições estritas em relação à confiabilidade da entrega e, focando na IoT, quando a rede é formada por dispositivos com alta mobilidade e baixa capacidade computacional e energética. Dispositivos com restrições de recursos, como energia, precisam fazer um uso eficiente dos mesmos. Desta forma, ao utilizar o protocolo UDP, os dispositivos não precisam manter conexões, fazendo com que os dispositivos possam poupar energia entrando em estado de hibernação sem prejudicar as aplicações e seus serviços [Sonar and Upadhyay 2014].

Camada de Percepção

A camada de percepção determina como os dispositivos interagem com o ambiente ao seu redor. Estes dispositivos podem atuar como sensores, atuadores ou de maneira híbrida. Os sensores captam dados do ambiente e os enviam para serem interpretados por um dispositivo com capacidade de processar esses dados e obter informações. Os atuadores interagem com o ambiente fisicamente, então eles recebem comandos para realizar atividades, por exemplo, ligar uma lâmpada inteligente. Os dispositivos híbridos normalmente já contêm uma determinada capacidade computacional, onde ele se torna capaz de perceber, processar os dados, interagir com o ambiente e compartilhar suas ações com a rede. Alguns dispositivos híbridos dotados de maior poder computacional (também chamados de coordenador) podem assumir o papel de *gateway*, executando atividades de gerenciamento da rede.

A Figura 1.4 mostra alguns exemplos de dispositivos utilizados na camada de percepção.As aplicações para casas inteligentes implementam redes IoT através dos móveis e eletrodomésticos, eles são equipados com sensores de temperatura, ruído, temporizadores. Uma geladeira, por exemplo, pode identificar a falta de insumos e gerar uma lista de compras e enviar para o *smartphone* do usuário. Os dispositivos pessoais podem se equipar de sensores de localização (GPS) ou acompanhamento cardíaco e promover um acompanhamento completo da saúde do usuário. Os dispositivos para controle de energia elétrica podem identificar oscilações na transmissão de energia elétrica e desligar os dispositivos, ou controlar o consumo excessivo de energia elétrica. Para diminuir os cus-

tos de mão de obra ou até otimizar a produção, as aplicações para fábricas inteligentes utilizam dispositivos como braços eletrônicos, equipados com sensores e atuadores de movimentação e recebem ordens remotamente. Os dispositivos de segurança permitem a monitoração remota de ambientes à distância, para isso câmeras IP capturam imagens e as transmitem através da rede, ou fechaduras que recebem ordens remotas.

Figura 1.4: Dispositivos IoT

1.2.2. Computação em Nuvem e IoT

Mesmo com alguns dispositivos assumindo a posição de coordenador da topologia da rede, nem sempre eles possuem a capacidade de realizar tarefas mais complexas. Incluir dispositivos mais robustos pode sair muito custoso, tanto financeiramente como também em termos de tempo com instalação e manutenção. Isso trouxe a necessidade de implementar arquiteturas que ofereçam recursos extras e sob demanda para a IoT. Esses recursos incluem poder de processamento, armazenamento, serviços de rede, aplicações completas e até ganho no consumo de bateria, visto que a IoT terceirizaria os recursos e as atividades de processamento. O modelo de computação em nuvem proporciona o fornecimento de recursos computacionais necessários para a IoT através da Internet e seguindo o modelo cliente/servidor. Além de ser um modelo amadurecido, a nuvem oferece formas de tratar Big Data e a heterogeneidade de dispositivos e tecnologias [Botta et al. 2016]. Os serviços em nuvem são adquiridos através de contratos de nível de serviço (do inglês, *Service Level Agreement* - SLA) que quantificam os recursos e especificam as regras de uso e valores entre o fornecedor do serviço e o cliente. A principal vantagem de utilizar a computação em nuvem consiste na possibilidade de integrar recursos computacionais e serviços que na maior parte das estruturas IoT são escassos.

No entanto, devido ao constante aumento no número de dispositivos, a demanda de serviços de rede também cresceu e acabou gerando problemas de escalabilidade e latência para os serviços ofertados em nuvem. Desta forma, o paradigma de computação em névoa (*fog*) surgiu para aliviar a sobrecarga dos servidores e enlaces responsáveis por interligar a borda da Internet com a nuvem. A *fog* aproxima da borda parte dos serviços oferecidos originalmente em nuvem [Alrawais et al. 2017], permitindo o pré-processamento de dados ou a pré-seleção dos recursos da nuvem. Além disso, a *fog* alivia a sobrecarga e permite um melhor desempenho em aplicações em tempo real. A Figura 1.5 apresenta uma visão geral sobre a organização dos serviços (nuvem e *fog*), e como os dispositivos se organizam para se conectar à Internet.

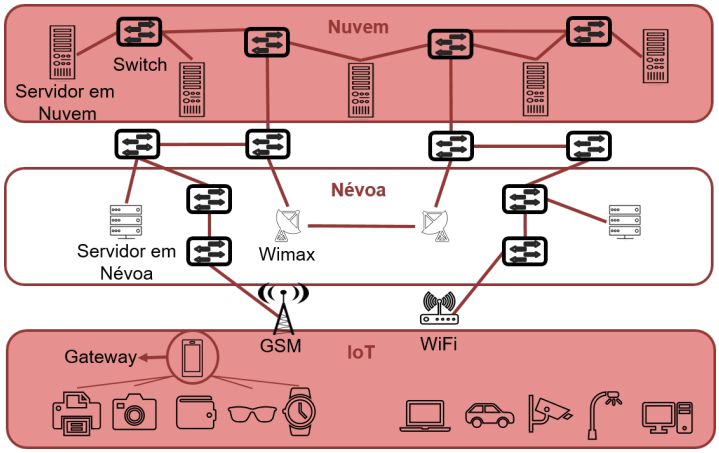


Figura 1.5: Exemplo de uma Arquitetura de Provimento de Serviços para IoT

1.2.3. Internet de Todas as Coisas

Devido à alta capacidade de impactar o dia-a-dia do cidadão moderno, a Cisco alavancou a discussão sobre a iteração das "coisas"com as pessoas [Bradley et al. 2013]. Neste contexto, além de considerar as comunicações diretas entre os dispositivos, eles também consideram iterações entre as pessoas e os dispositivos, e entre as pessoas e as pessoas. Assim como a banda larga consistiu em um fator crítico de crescimento econômico, é esperado que a IoE ocasionará um impacto semelhante ao abranger a inclusão social, a melhoria na prestação de serviços e a criação de muitas novas oportunidades. Outro benefício da IoE consiste no seu alto potencial de coletar e analisar dados de milhões de dispositivos, habilitando a automatização dos processos baseados nas pessoas [Mitchell et al. 2013]. Para organizar essa quantidade de informações, dispositivos e pessoas [Evans 2012], considera-se quatro pilares para construção de sistemas eficientes IoE, são eles; pessoas, dados, coisas e processos (Figura 1.6).

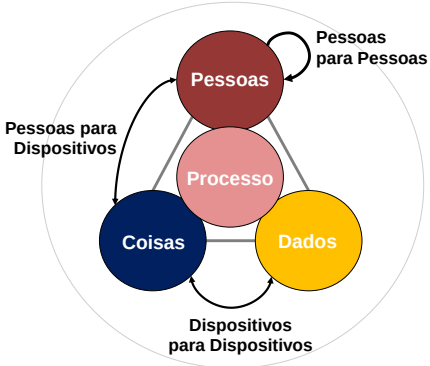


Figura 1.6: Internet de Todas as Coisas [Evans 2012]

As pessoas interagem com esses sistemas se conectando à Internet através de seus dispositivos e gerando dados. Os dispositivos implementam funcionalidades associadas a aplicações como cuidados com a saúde e iteração social (redes sociais). Os dados normalmente coletados por dispositivos são transmitidos através da Internet para serem analisados. Estima-se que a medida que as tecnologias desses dispositivos e sistemas de análise

de dados forem se desenvolvendo, a capacidade de cruzar dados e trazer soluções efetivas também aumentará. A IoT como relatada nesta seção está em constante desenvolvimento, elevando a capacidade dos dispositivos a desenvolverem ciência do contexto em que estão inseridos ou dos objetivos que devem cumprir. Os processos envolvem a gestão dos outros pilares, tornando os objetivos que levam a relação entre pessoas, dados e dispositivos mais eficientes.

Alguns autores como em [Schatten et al. 2016a] levantam questionamentos de como esses sistemas que envolvem milhões de dispositivos devem ser construídos. No mesmo trabalho os autores sugerem que além de inteligência, os dispositivos também precisam considerar as relações sociais. Na literatura, já existem propostas que definem arquiteturas e formas de relacionamento social entre as coisas [Bernabe et al. 2016, Bassi et al. 2013], onde os dispositivos se agrupam através do contexto em que estão inseridos. Em [Atzori et al. 2017], os autores propõem um *framework* para gerência de multidões de dispositivos móveis conscientes de sociabilidade, para isso ele utiliza uma ferramenta de virtualização de dispositivos que segue um algoritmo de controle de recursos. Esses trabalhos demonstram a evolução para uma IoE inteligente e organizada.

Esta seção mostrou os principais fatores que devem ser levados em consideração ao desenvolver soluções IoT, bem como a sua arquitetura, os principais protocolos utilizados, modelo de provimento de serviço mais utilizado e também um breve estudo sobe a IoE. Ao somar diferentes tecnologias, a complexidade acaba aumentando, dificultando o desenvolvimento de soluções para gerência de redes e segurança, principalmente quando consideramos aspectos iterativos entre as pessoas e os dispositivos como na IoE. Nas próximas seções, apresentamos os conceitos básicos de redes definidas por software, as quais trazem vantagens na gerência de redes, bem como as principais ameaças de segurança e defesas para essas arquiteturas.

1.3. Redes Definidas por Software

O propósito fundamental das redes de comunicação consiste em transferir dados de um ponto para outro. Uma das principais funcionalidades envolvidas na transferência dos dados consiste no encaminhamento de pacotes (unidade de dados tratada na camada de rede da arquitetura TCP/IP). Esta funcionalidade determina o modo como os pacotes são trafegados entre os diferentes equipamentos de rede intermediários. Tipicamente, as redes são construídas com muitos equipamentos, compreendendo roteadores, comutadores e dispositivos intermediários, tais como *firewalls*, balanceadores de carga e sistemas de detecção de intrusão. Cada um destes diferentes equipamentos necessita ser configurado de maneira específica para desempenhar suas respectivas tarefas associadas com a manipulação de pacotes [Feamster et al. 2014, Nunes et al. 2014]. Desta forma, encaminhar pacotes de maneira eficiente entre os equipamentos é essencial.

Nas redes tradicionais, a configuração referente às decisões de encaminhamento de pacotes e a configuração física dos dispositivos são combinadas no mesmo equipamento da rede. Através dessa abordagem, após a definição inicial do gerenciamento de fluxo (política de encaminhamento), a única maneira de ajustar esta política é através da configuração individual dos equipamentos. Essa característica ocasiona limitações quanto à administração de redes em larga escala, pois demanda a configuração individual de cada

equipamento da rede [Sezer et al. 2013], requerendo um nível elevado de conhecimento dos seus operadores [Nunes et al. 2014]. Além disso, novas demandas para configuração do tráfego de rede têm sido originadas por tendências emergentes nas tecnologias de comunicação, tais como mobilidade, relações sociais e *Big Data* [Xia et al. 2015]. Em virtude das limitações existentes nas redes tradicionais e as novas demandas para configuração do tráfego de rede, emerge a necessidade em repensar a forma como os pacotes são encaminhados na rede.

As redes SDN surgiram como uma iniciativa para contornar estas limitações e atender às novas demandas de configuração de tráfego de rede. Estas redes foram citadas no relatório da *IEEE Computer Society* como uma das 23 tecnologias que prometem mudar o mundo até 2022 [Alkhatib et al. 2015]. A *Open Networking Foundation* define SDN como um tipo de arquitetura de rede que desacopla o plano de controle do plano de dados, centraliza de modo lógico a inteligência e o estado da rede, abstraindo a infraestrutura de rede das aplicações [Open Networking Foundation 2012]. No plano de dados são tratadas as necessidades da infraestrutura física para o encaminhamento de dados, envolvendo os equipamentos de rede como comutadores e roteadores [Nunes et al. 2014, Sezer et al. 2013]. O plano de controle compreende a tomada de decisão no encaminhamento de pacotes na rede, possibilitando a programação dos caminhos que serão usados pelos fluxos de dados e representando inteligência da rede. Através das redes SDN, espera-se simplificar o gerenciamento de redes em larga escala, de maneira a alcançar os requisitos das novas demandas de configuração do tráfego de rede.

Uma rede SDN típica é composta por três partes principais: aplicação, plano de controle e plano de dados [Farhady et al. 2015], como ilustra Figura 1.7. A *aplicação* indica a parte que explora o desacoplamento do plano de controle e o plano de dados para alcançar objetivos específicos, tais como mecanismos de segurança ou soluções de monitoramento de rede. As aplicações se comunicam com o controlador do plano de controle através da interface *limite norte* do plano de controle. O *plano de controle* consiste na parte que manipula os dispositivos de encaminhamento de fluxo de dados através de um controlador visando alcançar objetivos específicos de uma dada aplicação. O controlador usa a interface *limite sul* do *switch* SDN para se conectar com o plano de dados. O *plano de dados* consiste na parte que suporta um protocolo compartilhado (por exemplo o *OpenFlow*) com o controlador e manipula os pacotes atuais com base nas configurações que são manipuladas pelo controlador.

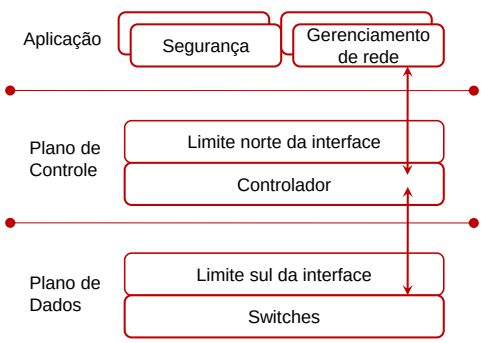


Figura 1.7: Componentes de uma Rede SDN. Adaptado de [Farhady et al. 2015].

Considerando o advento das SDNs, atualmente os paradigmas de rede podem ser divididos em três tipos de acordo com a implantação dos planos de controle e dados [Farhady et al. 2015]. A Figura 1.8 ilustra em alto nível os paradigmas de rede. No paradigma tradicional, onde a rede é centrada no hardware, os *switches* são usualmente sistemas fechados que agregam em si mesmos os planos de controle e dados e suportam interfaces de controle específicas desenvolvidas por seus fabricantes. Portanto, na abordagem de rede centrada no hardware, a implantação de novos protocolos e serviços (e mesmo novas versões de protocolos existentes) se torna desafiadora, pois todos os *switches* precisam ser atualizados ou substituídos. Em contraste, nas redes SDN, os *switches* se tornam simples dispositivos de encaminhamento de dados e controladores centralizados são responsáveis pelo gerenciamento da rede como um todo. Este desacoplamento dos planos de dados e controle facilita a implantação de novos protocolos e serviços, pois a decomposição possibilita que os *switches* sejam programados através dos controladores. Finalmente, a abordagem híbrida suporta tanto o modelo tradicional de rede, quanto o modelo SDN.

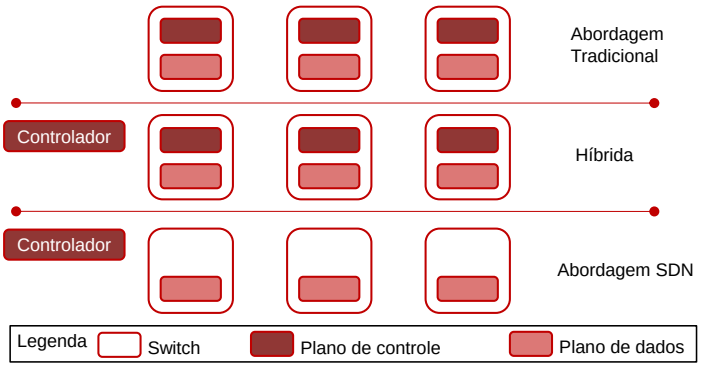


Figura 1.8: Paradigmas de Rede. Adaptado de [Farhady et al. 2015].

1.3.1. Evolução das Redes Programáveis

As redes definidas por software vêm recebendo atenção da indústria e academia. Todavia, vale salientar que as ideias da programação de redes de computadores através do desacoplamento do controle lógico das redes vêm sendo discutidas durante anos. Esta seção provê uma visão geral das abordagens precursoras das redes SDN as quais contribuíram para o amadurecimento das ideias que sustentam o paradigma atual das SDNs. As principais iniciativas que contribuíram para o surgimento das SDNs são o grupo de trabalho *Open Signaling* (OPENSIG) [Campbell et al. 1999], IETF *Network Configuration* (NETCONF) [R. Enns 2006], *Active Networking* [Tennenhouse et al. 1997], *Devolved Control of ATM Networks* (DCAN) [Leslie et al. 2015], Projeto 4D [Greenberg et al. 2005] e *Ethane* [Casado et al. 2007]. O grupo OPENSIG iniciou em 1995 promovendo uma série de *workshops* dedicados a tornar as redes ATM, Internet e as redes móveis mais abertas, extensíveis e programáveis [Campbell et al. 1999]. Eles acreditaram na necessidade de implantar a comunicação com o hardware através do controle baseado em software, mas que esta tarefa apresentaria desafios em termos práticos. O núcleo da proposta do OPENSIG era prover acesso ao hardware de rede através de interfaces de rede programáveis visando à criação de novos tipos de serviços baseados em um ambiente distribuído pro-

gramável. O protocolo *General Switch Management Protocol* (GSMP) consistiu em um dos principais resultados desta iniciativa. Esta iniciativa encontra-se oficialmente concluída e a última versão do protocolo GSMP foi publicada em Junho de 2002.

A iniciativa *Active Networking* também iniciou em 1995 com objetivo de proporcionar infraestruturas de rede que poderiam ser programáveis para melhor se adaptar a serviços específicos [Tennenhouse et al. 1997, Moore et al. 2001]. Para alcançar este objetivo, o *Active Networking* considerava duas abordagens: (1) os *switches* programáveis pelos usuários; e (2) cápsulas. A primeira abordagem previa o gerenciamento de canais para transferir as bandas de entrada e saída. A segunda abordagem advogava que programas poderiam ser fragmentados e carregados nas mensagens dos usuários para serem interpretados e executados pelos roteadores. Apesar dos diversos esforços desta iniciativa, o *Active Networking* nunca chegou a ser considerado para a adoção pela indústria devido às limitações de desempenho e segurança [Tennenhouse and Wetherall 2002].

Ainda na metade dos anos 90 surgiu a iniciativa DCAN visando projetar e desenvolver uma infraestrutura necessária para gerenciamento e controle escalável das redes ATM [Leslie et al. 2015]. A principal premissa do DCAN consistia em desacoplar o controle e o gerenciamento das funções de encaminhamento dos pacotes, este consiste basicamente no conceito fundamental por trás das redes SDN. O DCAN também assumia um protocolo minimalista entre o gerenciador e a rede, o que de modo geral consiste no papel desempenhado pelo *OpenFlow*. Esta iniciativa foi oficialmente encerrada em 1998.

O projeto 4D iniciou em 2004 e enfatizava a separação entre a decisão lógica do roteamento de dados e os protocolos que governavam a interação entre os dispositivos de rede [Rexford et al. 2004, Greenberg et al. 2005]. Este projeto defendia que o controle das redes deveria ser separado em três planos: decisão, disseminação e descobrimento. Através desta organização o plano de decisão possuía uma visão geral da rede, contando com serviços oferecidos pelo plano de disseminação e descobrimento. Estes três planos controlariam o plano de dados, responsável por encaminhar o tráfego de rede.

O NETCONF surgiu em 2006 tendo como objetivo melhorar o ***S**imple **N**etwork **M**anagement **P**rotocol* (SNMP) para configuração de dispositivos da rede [R. Enns 2006]. O protocolo SNMP foi proposto na década de 80 e se mostrou muito popular como um protocolo de gerenciamento de rede ao usar a ***S**tructured **M**anagement **I**nterface* (SMI) para encontrar e alterar os dados contidos no ***M**anagement **I**nformation **B**ase* (MIB) [J. D. Case and M. Fedor and M. L. Schoffstall and J. Davin 1990]. Através do SNMP era possível alterar variáveis na MIB para modificar aspectos de configuração. Com o passar do tempo, o SNMP se proporcionou contribuições mais eficazes quando utilizado como uma ferramenta de monitoramento de desempenho e gerenciamento de faltas. Além disto, foram detectadas limitações na concepção do SNMP, principalmente referente a aspectos de segurança. O NETCONF visava corrigir as limitações do SNMP ao prover uma simplificação da configuração e reconfiguração dos dispositivos, atuando como um bloco de gerenciamento, onde não existia a separação entre o plano de dados e o plano de controle. Atualmente esta iniciativa continua ativa.

O *Ethane* também foi iniciado em 2006 e consiste no predecessor imediato do *OpenFlow* [Nunes et al. 2014]. O *Ethane* focou na utilização de um controlador centralizado para gerenciar a segurança na rede [Casado et al. 2007]. Esta iniciativa empregava

dois componentes principais: um controlador e *switch ethane*. O componente controlador era responsável por decidir se um pacote deveria ser encaminhado e o *switch ethane* mantinha uma tabela de fluxos e um canal seguro para o controlador. Uma das características mais marcantes desta iniciativa consistiu no paradigma de controle de acesso baseado em identidades, o qual contribuiu para a implementação de controladores SDN largamente utilizados atualmente, tais como o *NOX* [Gude et al. 2008], *Maestro* [Ng 2010] e *Beacon* [Erickson 2013].

1.3.2. Principais arquiteturas SDN

Esta seção revisa as arquiteturas *ForCES* e *OpenFlow*, as quais seguem o princípio básico da separação entre o plano de controle e o plano de dados. Além disto, ambas as arquiteturas padronizam a troca de informação entre estes planos. No entanto, elas são diferentes em termos de projeto, modelo de encaminhamento e interface de protocolo.

ForCES

Consiste na abordagem proposta pelo grupo de trabalho da *IETF Forwarding and Control Element Separation* (ForCES) para refinar a arquitetura interna dos dispositivos de rede tendo o controle separado dos elementos de encaminhamento de dados [Nunes et al. 2014]. Nesta arquitetura, o dispositivo de rede permanece representado como uma única entidade. O principal objetivo deste consistia em combinar novos hardwares para encaminhamento de dados com um controle provido por uma terceira parte dentro de um único dispositivo de rede. Assim, os planos de controle e de dados são mantidos próximos, por exemplo, na mesma caixa ou sala. Em contraste, o plano de controle é tirado inteiramente do dispositivo de rede, seguindo o mesmo estilo dos sistemas SDN baseados em *OpenFlow*. O ForCES define duas entidades lógicas denominadas *Forwarding Element* (FE) e *Control Element* (CE), sendo que ambas se comunicam através do protocolo *ForCES*. O FE é responsável por prover a manipulação dos pacotes. O CE executa o controle, sinaliza funções e emprega o protocolo *ForCES* para instruir os FEs em como manipular pacotes. O protocolo funciona de acordo com o modelo mestre/escravo, onde os FEs atuam como escravos e CEs atuam como mestres. Um dos principais componentes da arquitetura *ForCES* consiste no *Logical Function Block* (LFB). O LFB consiste em um componente armazenado nos FEs que é controlado pelos CEs através do protocolo *ForCES*. O LFB possibilita que os CEs controlem a configuração dos FEs, especificando como os FEs devem processar os pacotes. O grupo de trabalho do *ForCES* foi oficialmente declarado como concluído em 2015 e contribuíram com uma variedade de documentos, por exemplo, um arcabouço definindo as principais entidades e suas interações, um modelo de linguagem que define funções lógicas dentro do dispositivo de encaminhamento e um protocolo para comunicação entre o controlador e os elementos de encaminhamento.

OpenFlow

O *OpenFlow* padroniza a troca de informação entre os planos de controle e de dados [McKeown et al. 2008]. Na arquitetura *OpenFlow* ilustrada na Figura 1.9 o dispositivo de encaminhamento, ou *switch OpenFlow*, contém uma ou mais tabelas de fluxos e uma camada de abstração que possibilita a comunicação segura com o controlador através do

protocolo *OpenFlow*. As tabelas de fluxos consistem de fluxos de entrada, onde cada uma determina como os pacotes pertencentes a um fluxo serão processados e encaminhados. Os fluxos de entrada consiste tipicamente de (1): campos de correspondência, ou regras de correspondência; campos de correspondência podem conter informações encontradas nos cabeçalhos dos pacote, porta de ingresso e metadados; (2) contadores, usados para coletar estatísticas para um fluxo particular, tal como o número de pacotes recebidos, número de *bytes* e a duração do fluxo; e (3) um conjunto de instruções, ou ações para serem aplicadas sobre uma correspondência, elas ditam como manipular pacotes correspondentes. Quando um pacote chega a um *switch OpenFlow*, os cabeçalhos do pacote são extraídos e comparados com os campos de correspondência da tabela de fluxos. Se existir uma correspondência, o *switch* aplica o conjunto de instruções ou ações associadas com a entrada do fluxo correspondente. Se não existir uma correspondência na tabela de fluxos, a ação tomada pelo *switch* dependerá das instruções definidas nas entradas de falhas de correspondência de fluxos. Estas entradas especificam o conjunto de ações a serem tomadas quando nenhuma correspondência para um dado fluxo é encontrada para um pacote de entrada, por exemplo, o descarte do pacote, o encaminhamento para outra tabela de fluxos ou o encaminhamento do pacote para o controlador *OpenFlow*.

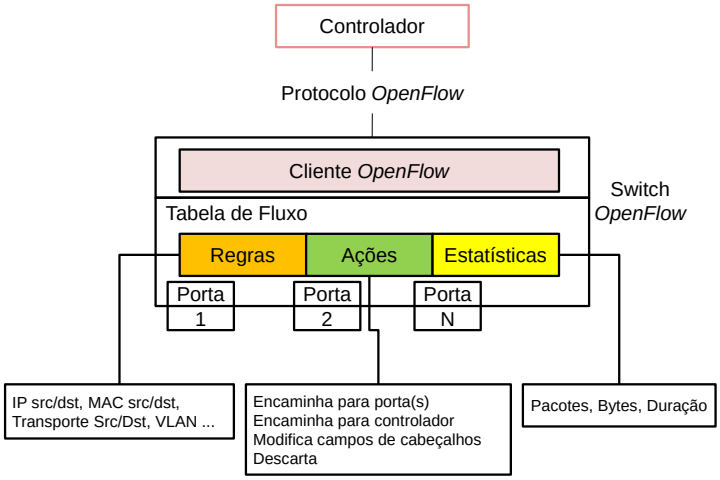


Figura 1.9: Arquitetura *OpenFlow*. Adaptado de [Nunes et al. 2014].

A comunicação entre o controlador e o *switch* acontece via protocolo *OpenFlow*, o qual define um conjunto de mensagens que podem ser trocadas entre estas entidades através de um canal seguro. Usando o protocolo *OpenFlow* um controlador remoto pode executar operações de gerenciamento, como por exemplo adicionar, atualizar ou remover fluxos de entrada das tabelas de fluxos dos *switches*. Estas operações podem ocorrer de modo reativo ou proativo. No modo proativo, as operações de gerenciamento são executadas em resposta a chegada de um pacote. Enquanto que o modo reativo compreende na predefinição das operações de gerenciamento para fluxos previamente esperados. Embora protocolos como o *OpenFlow* especifiquem que um *switch* é controlado por um controlador, implicando em uma centralização, as redes SDN podem possuir um plano de controle centralizado ou distribuído. Como a comunicação entre controladores não é definida pelo *OpenFlow*, torna-se necessário algum tipo de distribuição ou redundância

no plano de controle. O controlador fisicamente centralizado representa um ponto único de falhas para toda a rede. Portanto, o *OpenFlow* permite a conexão de múltiplos controladores para um *switch* o qual deveria possibilitar a utilização de controladores backups para serem ativados em caso de uma falha. Muitos pesquisadores adotam manter o plano de controle logicamente centralizado, mas fisicamente distribuído [Nunes et al. 2014].

1.3.3. IoT baseada em SDN

Os princípios das redes SDNs podem ser direcionados para diferentes aspectos das estruturas de provimento de serviços na IoT [Bera et al. 2017]. A Figura 1.10 demonstra o funcionamento de uma rede IoT baseada em SDN considerando a divisão entre os planos de gerência, controle, dados, serviço e IoT. O plano de gerência compreende sistemas responsáveis por controlar as operações de rede em nível de aplicação. O plano de controle atua como o cérebro da estrutura, sendo composto pelos controladores SDN. Os controladores são dispositivos logicamente posicionados no centro da estrutura e realizam as principais abstrações para que o plano de gerência se relacione com os dispositivos do plano de dados. O plano de dados agrega os dispositivos responsáveis pelo encaminhamento de dados. Esses dispositivos detêm tabelas de regras responsáveis por controlar os fluxos de tráfego da rede. Seguindo esta organização, sempre que um comutador do plano de dados não souber lidar com um fluxo ele solicita auxílio ao plano de controle. O plano de serviço representa a computação em nuvem e fog, são serviços para IoT fornecidos através da Internet. Por fim, o plano IoT envolve os dispositivos com capacidade de se conectar com a Internet. É importante destacar que os planos que representam as SDNs (Planos de gerência, controle e dados) estão presentes nos planos de serviço e IoT, e eles realizam as tarefas de distribuição e controle do tráfego de rede. A arquitetura representada na Figura 1.10 mostra que o modelo SDN pode estar presente na expectativa do servidor online (Nuvem), da disseminação dos dados desses serviços (fog) ou IoT.

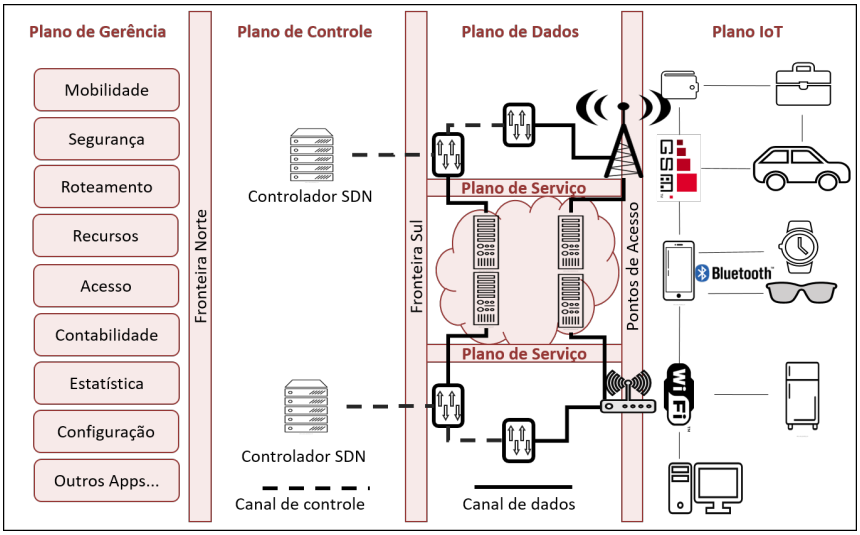


Figura 1.10: Exemplo de IoT Baseada em SDN

Devido ao rápido crescimento no número de dispositivos conectados a Internet, a gerência de redes se torna um fator importante para as estruturas IoT. Administrar redes com grandes quantidades de dispositivos gerando dados massivos, somado com a escas-

sez de recursos, trazem a necessidade de processamento de dados remotos. Isso exige da estrutura mecanismos para administrar o tráfego da rede de forma eficiente e rápida. As SDNs oferecem benefícios para a gerência de redes de computadores. Centralizar o plano de controle da rede permite uma visão global da estrutura, o que traz agilidade para mecanismos que realizam atividades como distribuição, controle de tráfego e alocação de recursos. Outro ponto em que as SDNs beneficiam a IoT consiste na simplificação para implementar soluções baseadas em virtualização de funções de rede (do inglês, *Network Function Virtualization* - NFV), que habilitam os dispositivos, originalmente projetados para executar uma única função, para desempenhar múltiplas funções [Bera et al. 2017].

Ao agregar estas tecnologias (SDN, Nuvem e IoT), novas funções para esse conjunto vêm sendo desenvolvidas. O modelo tradicional para as SDNs implicam em um controlador fixo para cada sub-rede, ou seja, um dispositivo servidor exclusivo para controle das regras de rede. No entanto, no modelo IoT as sub-redes são móveis, sem fio e muitas vezes são interconectados por conexões de um único salto e organizadas através de um *gateway* (não existe nenhum de dispositivo de encaminhamento entre o *gateway* e o dispositivo final); ou seja, não necessitam de regras de roteamento. Por outro lado, devido a característica de escassez de recursos, precisam de sistemas de administração de recursos mais eficientes. Levando em consideração esses parâmetros, na literatura existem trabalhos que implementam o modelo de controladores SDN no dispositivo final. Lee *et al.* propõem administrar o consumo dos recursos através de uma hierarquia de controladores composta por controladores móveis e um controlador global [Lee et al. 2014]. O controlador global desempenha atividades administrativas nos controladores móveis, sendo posicionado de forma fixa. Os controladores móveis administram o consumo dos recursos de acesso considerando a disponibilidade dos canais sem fio através do tempo de transmissão para a comunicação direta entre os dispositivos, controlam as especificações de qualidade de serviço para o padrão IEEE 802.11 por meio das regras instaladas sob demanda pelo controlador global. Esta organização permite que as SDNs possam se tornar um serviço para gerência de redes IoT ofertados como um serviço de nuvem, demonstrando a flexibilidade oferecida por este conjunto de tecnologias.

1.4. Principais Ameaças

A segurança em redes vem sendo um dos principais focos para o desenvolvimento de pesquisas em IoT e SDN na atualidade[Bera et al. 2017]. Estes problemas muitas vezes comprometem a segurança dos dados trafegando na rede. Os três principais pilares da segurança da informação consistem na confidencialidade, integridade e disponibilidade. Uma estrutura de rede considerada segura deve manter estes pilares consonância com seus objetivos. Os usuários mal-intencionados (atacantes) realizam análises para encontrar vulnerabilidades do sistema podendo resultar em vazamento de dados, acesso não autorizado nos serviços e operações da rede, modificação de dados [Scott-Hayward et al. 2016]. Existem ameaças de segurança para cada plano da arquitetura de IoT baseada em SDN apresentada na Figura 1.10. Estas ameaças são classificadas de acordo com o tipo de rede alvo, podendo compreender redes estruturadas (SDN) e redes não estruturadas (IoT e IoT Baseada em SDN). As ameaças de segurança podem ser classificadas em intrusões e ataques. Esta seção aborda estas ameaças.

1.4.1. Intrusões e Ataques

Um dos principais problemas da IoT baseada em SDN é a segurança contra intrusões e ataques [Farris et al. 2018]. As intrusões visam explorar vulnerabilidades nos sistemas que garantem a confidencialidade dos dados na rede. Os usuários mal-intencionados usualmente falsificam ou clonam dados de identificação e por meio de um acesso não autorizado a rede (intrusão), remoto ou físico, capturam e manipulam os dados que trafegam na rede (ataque). Existem diversos tipos de ataques, cada um atinge diferentes planos da IoT baseada em SDN (Tabela 1.1). As aplicações tanto da IoT como das SDNs e o plano de controle são atingidas por ataques que manipulam, falsificam ou negam o fornecimento de informações. O plano de rede e a camada de rede são prejudicados por ataques que exploram os pontos de acesso e a transmissão dos dados. A camada de percepção é atingida por ataques que envolvem a falsificação de dispositivos, encaminhamento seletivo de informações ou esgotamento dos recursos.

Para realizar uma intrusão, o atacante pode personificar usuários, administradores, dispositivos ou até aplicações. Para realizar a intrusão ao sistema, o atacante pode utilizar as técnicas de análise ou força bruta. As técnicas de análise consistem em adquirir informações da estrutura de rede. As mais comuns em estruturas de rede baseadas em SDNs são os escaneadores de WLAN, *sniffers*, escaneadores de portas, *phishing* e engenharia social. Os escaneadores de WLAN são dispositivos equipados com placas de redes sem fio que buscam pontos de acesso sem fio adaptando a frequência de transmissão, essa técnica de escaneamento se torna útil para descobrir a localização dos dispositivos e consequentemente informações sobre a topologia. Os *sniffers* capturam todo o tráfego de rede que não é criptografado e realizam análises para descobrir os serviços ativos na rede. Os escaneadores de portas se conectam a servidores e exploram o TCP e o UDP para realizar um mapeamento das portas, através disso eles conseguem planejar quais serviços podem ser atacados. No *fishing* o atacante explora a inocência dos usuários legítimos da rede, com o objetivo de adquirir os dados de acesso, para isso ele insere um código malicioso nos dispositivos das vítimas através de links de acesso, imagens ou páginas web falsificadas. Os métodos de engenharia social também exploram a inocência dos usuários, então os atacantes tentam adquirir informações dos usuários como cargos, horários de ponto, datas, ou seja, qualquer informação que revele senhas ou oportunidades de acesso físico à estrutura. Na força bruta, os atacantes testam todas as possibilidades de combinações de credenciais de acesso na tentativa de burlar as defesas da rede. Além disso, o atacante pode combinar as técnicas utilizando as informações adquiridas através das análises para direcionar os dados da força bruta. Por exemplo, utilizar uma lista dos nomes, cargos e datas de nascimento dos funcionários de uma empresa e programar um *script* para testar todas as combinações possíveis entre esses dados.

Ao conseguir penetrar as defesas da rede, o atacante pode desempenhar ataques considerando especificidades das SDNs e IoT. Nas SDNs, a personificação pode ocorrer no plano de controle ou no plano de dados. A personificação de um controlador (plano de controle) pode comprometer toda a arquitetura de rede, pois por meio deste dispositivo o atacante consegue manipular todas as operações de rede. Ao personificar um servidor provedor de serviços, o atacante pode capturar uma quantidade significativa de dispositivos que utilizam o serviço e desempenhar ataques de negação de serviço contra outro

alvo através do redirecionamento das requisições. Considerando uma rede IoT, a personificação pode ocorrer no dispositivo ou nos serviços. A personificação de um dispositivo (camada de percepção) implica no comprometimento do dispositivo em si e pode abrir brechas para outros ataques. Em nível de serviço, este ataque permite aos atacantes o controle dos dispositivos para prejudicar a rede ou divulgar dados indevidamente. As principais variações deste ataque estão classificados na Tabela 1.1.

Ataques	**SDN**			**IoT**		
	Aplicação	*Plano de controle*	*Plano de Rede*	*Aplicação*	*Camada de Rede*	*Camada de Percepção*
Sybil	•	•		•		•
Homem no Meio			•		•	
Buraco Negro		•	•		•	•
Falsificação e Manipulação dos dados	•	•	•	•	•	•
Negação de Serviço	•	•	•	•	•	•

Tabela 1.1: Categorização dos Ataques por Camada/Plano

Os ataques de homem no meio podem ocorrer em infraestruturas SDN de redes IoT que empregam a computação em névoa [Li et al. 2017]. A computação em névoa resulta em benefícios para IoT ao proporcionar uma etapa preliminar de processamento dos dados antes de os enviar para a nuvem e ao reduzir a latência na troca de dados de dispositivos IoT. As funcionalidades da computação em névoa e das redes SDN são integradas ao representar os nós fog como *switches SDN*. Todavia, esta combinação de soluções em rede torna-se vulnerável à ataques homem no meio. Neste cenário, atacantes interceptam o canal de comunicação entre o controlador e os *switches* do protocolo *OpenFlow*. Este canal possibilita o envio de comandos e requisições do controlador, bem como as estatísticas coletadas dos *switches*. A ocorrência de um ataque homem no meio neste canal ocasiona circunstâncias desastrosas para a rede e para seus usuários. Considerando a perspectiva os usuários, um atacante pode coletar informações sensitivas dos seus usuários, resultando no vazamento de dados confidenciais. Do ponto de vista da rede, o atacante pode enviar pacotes falsos para o controlador se passando pelos *switches*, contaminando a visão global que o controlador possui da rede. Como consequência, o controlador pode configurar de maneira equivocada os demais *switches*, gerando problemas de conectividade na rede.

Os ataques Sybil prejudicam as redes IoT através da inclusão de um dispositivo falso na estrutura para manipular regras de roteamento ou opiniões coletivas [Rajan et al. 2017]. A principal característica do ataque Sybil consiste em falsificar a identidade dos dispositivos. Nesse ataque o atacante utiliza dispositivos com falsas credenciais para acessar uma rede IoT. Para isso o atacante precisa conhecer as características das aplicações e dispositivos que utilizam a rede. Com objetivo de obter descobrir como funcionam as aplicações de autenticação, o atacante pode utilizar técnicas como *sniffing*, *phishing* e engenharia social. Ao descobrir detalhes sobre o método de autenticação do dispositivo em uma aplicação, o atacante falsifica os dados de um dispositivo e tenta conectá-lo

à estrutura. Ao conseguir, o dispositivo falso passa a simular o comportamento de um nó legítimo para agregar confiança e por fim inserir novos dispositivos maliciosos. Este tipo de ataque fere diretamente os pilares de confidencialidade e integridade da rede. O objetivo desse ataque normalmente consiste em adquirir dados confidenciais e manipular a opinião geral inserindo dados falsos na estrutura de rede. Em uma estrutura de IoT baseada em SDN, caso o atacante consiga incorporar um controlador, ele consegue manipular o tráfego de rede e direcionar para um alvo, a fim de realizar um ataque DDoS, transformando todos os dispositivos subordinados ao controlador em uma rede zumbi. A integração da infraestrutura SDN em uma rede IoT alvo deste ataque pode facilitar a proposição de mecanismos para identificar dispositivos com comportamento malicioso [Bull et al. 2016a].

Os ataques buraco de negro e de falsificação e manipulação de dados se aproveitam de regras de roteamento da rede. Para desempenhar estes ataques, o atacante utiliza os escaneadores de rede, de portas e *sniffers* para identificar os dispositivos e serviços em conjunto com seus respectivos dados temporais e informações de roteamento. Os dados temporais são importantes para identificar quando um dispositivo para de transmitir dados. Então o atacante se aproveita deste intervalo para transmitir dados maliciosos ou instalar regras de roteamento. No ataque buraco de minhoca o atacante cria um tunelamento para armazenar o tráfego da rede através de um dispositivo de rede capturado ou falsificado. Ao conseguir capturar registros suficientes do tráfego da rede o atacante pode desempenhar o ataque *replay*. Nessa variação do ataque buraco de minhoca o atacante consegue fazer com que a rede fique repetindo as ações passadas e burlando os sistemas de detecção de intrusão. Ao falsificar os dados de roteamento o atacante pode desempenhar atividades como camuflar um dispositivo malicioso ou instalar um *looping* de roteamento. O atacante também pode desempenhar o ataque do sumidouro, onde ele manipula os protocolos de roteamento para atrair todo o tráfego da rede, prejudicando a recepção dos dados no ponto de coleta. Uma variação dos ataques de falsificação consiste no ataque do buraco negro, onde o atacante instala uma regra em um dispositivo de encaminhamento capaz de ocasionar a eliminação de toda nova entrada de fluxos para isolar parte da rede dos sistemas de segurança.

1.4.2. Ataques de Negação de Serviço

Um ataque de negação de serviço (DoS) visa tornar indisponível um serviço ou informação da rede aos usuários legítimos. A abordagem mais tradicional para alcançar este objetivo consiste em sobrecarregar intencionalmente recursos da rede ou dos servidores, tais como a largura de banda da rede, processamento ou memória dos equipamentos de rede, a fim de impedir que usuários legítimos acessem determinados recursos computacionais [Bartholemy and Chen 2015]. Em sua abordagem tradicional, um atacante fabrica uma carga maliciosa e a dispara contra um serviço alvo com objetivo de comprometer seu desempenho e/ou torná-lo indisponível. A característica mais marcante deste ataque consiste na exaustão dos recursos computacionais do serviço ou sistema alvo. Dentre os principais alvos destes ataques encontram-se as infraestruturas de bancos, as empresas que oferecem serviços *on-line* e as organizações governamentais. Iniciativas vêm sendo realizadas para tratar a ocorrência este ataque no contexto de redes IoT baseadas em SDN [Wani and Revathi 2018]. Nestes ataques, o atacante consegue atingir todas os

planos da arquitetura da IoT baseada em SDN. Nas aplicações. tanto da IoT como das SDNs, o atacante explora as vulnerabilidades dos protocolos, através da falsificação de requisições. No plano de controle o atacante pode exaurir os recursos do controlador, negando o serviço de instalação de regras e consequentemente inviabilizando o funcionamento do plano de rede. Os ataques DoS que objetivam prejudicar a camada de rede da IoT, sobrecarregam o canal físico [Sonar and Upadhyay 2014].

O ataque de negação de serviço distribuído, comumente referenciado como DDoS (*Distributed Denial of Service*), é uma variação do ataque tradicional de negação de serviço. Esta variação do ataque emprega de forma coordenada uma grande quantidade de dispositivos previamente infectados para intensificar o poder disruptivo do ataque e dificultar a identificação do atacante [Wang et al. 2015, Paxson 2001]. Usualmente, os atacantes utilizam dispositivos interconectados através de infraestruturas conhecidas como *botnets* para executar ataques DDoS. Essas redes são compostas por computadores infectados, normalmente referenciados como escravos, *bot* ou zumbis, os quais atuam como geradores de carga maliciosa para um sistema alvo [Zargar et al. 2013].

Os ataques de negação de serviço foram observados pela comunidade acadêmica no início dos anos 80. No verão de 1999, o *Computer Incident Advisory Capability* (CIAC), grupo de especialistas responsável por relatar incidentes de segurança, reportou a primeira ocorrência deste ataque [Criscuolo 2000]. Desde então, muitos ataques DDoS vêm sendo disparados contra diferentes organizações com o objetivo de tornar indisponível determinados serviços alvos e gerar prejuízos financeiros ao incrementar custos com a restauração de serviços. Por exemplo, no mês de Fevereiro de 2000, a empresa *Yahoo!* vivenciou um dos primeiros ataques DDoS de inundação, o qual tornou indisponível os serviços prestados pela companhia durante cerca de duas horas, ocasionando uma perda financeira significativa [Wired 2000].

Em Outubro de 2002, um ataque DDoS tornou indisponível por uma hora 9 dos 13 servidores que proviam o serviço *Domain Name System* (DNS) para os usuários da Internet [Naraine 2002]. Outro grande ataque DDoS ocorreu em Fevereiro de 2004, tornando o site da *SCO Group* inacessível para usuários legítimos [Vijayan 2004]. Este ataque foi disparado através de sistemas que foram previamente infectados pelo vírus *Mydoom*. O vírus continha um código que instruiu milhares de computadores infectados para acessar o site da SCO no mesmo instante. O código do vírus *Mydoom* foi reutilizado para disparar ataques DDoS de inundação contra os maiores sites de finanças e redes de notícias dos governos da Coreia do Sul e dos Estados Unidos em Julho de 2009 [Myd 2009, Sudworth 2009]. Em Dezembro de 2010, um grupo auto intitulado como "*Anônimos*" orquestrou ataques DDoS responsáveis por inviabilizar o acesso aos sites das organizações *Mastercard*, *PostFinance* e *Visa* [Wik 2010]. Em 2013, o serviço de banco online dos 9 maiores bancos dos Estados Unidos (*Bank of America*, *Cititgroup*, *Wells Fargo*, *U.S. Bancorp*, PNC, *Capital One*, *Fifth Third Bank*, BB&T e HSBC) foram alvos de uma série de poderosos ataques DDoS disparados por um grupo denominado "*Izz ad-Din al-Qassam Cyber Fighters*" [Kitten 2013].

Em 2013 ocorreu o ataque DDoS contra a *Spamhaus*, uma organização anti-spam sem fins lucrativos [Prince 2013]. No dia 18 de Março de 2013, a empresa identificou um tráfego de entrada massivo em seus servidores de aproximadamente 10 Gbps como

pertencente a um ataque DDoS. No dia seguinte, os relatórios indicaram que o tráfego de entrada tinha aumentado, atingindo picos de 90 Gbps. Nos dias seguintes o ataque desapareceu por um período curto de tempo, retornando no dia 22 de Março com uma intensidade ainda maior, alcançando 120 Gbps. Através de investigações envolvendo peritos da área de segurança computacional, foi descoberto que o autor do ataque consistia em um adolescente residente na cidade de Londres.

Um ataque motivado por razões políticas atingiu 500 Gbps e foi denominado como Occupy Central [Olson 2014]. ELe teve como objetivo remover o poder eleitoral de um pequeno grupo leal aos interesses de Pequim, favorecendo os cidadãos de Hong Kong. Durante este acontecimento, foi realizada uma pesquisa de opinião pública através da Internet para realizar uma reforma constitucional. O ataque teve como objetivo prejudicar a consolidação da democracia em Hong Kong ao impedir o acesso ao sistema de votação do grupo pró-democracia. Alguns indícios mostram que o ataque foi originado internamente no país. Em 2016 o blog KrebsOnSecurity [2] sofreu um ataque que atingiu cerca de 620 Gbps. O ataque foi motivado pelo fato de o blog ter divulgado informações de um serviço que cobrava para realizar ataques DDoS. Os donos do serviço foram presos logo em seguida. A empresa de Internet Akamai que mitigou o ataque descobriu que a botnet era composta por câmeras de segurança. Até então o maior ataque registrado tinha atingido cerca de 363 Gbps. Ainda em 2016 um *malware* que explora dispositivos com usuário e senha padrão foi descoberto ao realizarem análises em um ataque realizado contra empresa francesa de serviços em nuvem a OVH [Goodin 2016]. O ataque atingiu cerca de 1.1 Tbps de tráfego gerado por mais de 145 mil câmeras de segurança. Segundo estudos realizados pela Akamai [Mckeay 2016] os dispositivos desta botnet estão espalhados em cerca de quatro países.

O maior ataque DDoS registrado até o momento atingiu 1.35 Tbps e foi contra o servidor de um dos maiores serviços de hospedagem de códigos fontes do munto, o *GitHub* [Newman 2018]. Os atacantes utilizaram servidores de memória distribuída (*memcache*) para amplificar a potência do ataque. Os autores e os motivos não foram revelados, no entanto o ataque conseguiu deixar offline os servidores da empresa por 10 minutos. Como defesa a empresa utilizou os serviços de mitigação do provedor de internet Akamai, o Akamai Prolexic. Esse tipo de ataque utiliza de técnicas de falsificação de cabeçalhos para realizar requisições aos servidores *memcache*, assim as respostas são direcionadas para o alvo.

Categorização da Motivação dos Atacantes

Existem diversas razões motivadoras para os atacantes desencadearem ataques DDoS. Estas razões podem ser resumidas em cinco categorias, ganho econômico, vingança, crença ideológica, desafio intelectual e guerras cibernéticas [Zargar et al. 2013]. Os atacantes motivados por razões econômicas têm como objetivo obter ganhos financeiros de corporações através de extorsão. Devido à natureza desta motivação, os atacantes desta categoria apresentam usualmente alto nível de habilidades técnicas e muita experiência. Os ataques motivados por ganhos financeiros causam os maiores danos e visam desencadear falhas de funcionamento em um curto espaço de tempo nos sistemas alvo.

[2] https://krebsonsecurity.com/

Os atacantes cuja principal razão consiste na vingança são geralmente indivíduos frustrados, possivelmente com baixo nível de habilidade técnica e usualmente executam ataques DDoS em resposta a uma aparente injustiça. Outros atacantes são inspirados por suas convicções ideológicas para escolher seus alvos [Prasad et al. 2014]. Atualmente, a maioria dos atacantes é incentivada por estas razões. Dentre os principais motivos encontrados encontram-se as opiniões políticas, religiosas e aspectos morais. A principal característica dos atacantes motivados pelo desafio intelectual compreende em colocar em prática e aprender como executar vários ataques. Estes indivíduos usualmente são hackers iniciantes que querem mostrar suas capacidades. Atualmente, existem várias ferramentas fáceis de usar e *botnets* que podem ser alugadas, possibilitando que até mesmo amadores podem ser capazes de executar ataques DDoS bem-sucedidos.

Os atacantes motivados por guerras cibernéticas geralmente pertencem a organizações militares ou terroristas e estão politicamente encorajados para atacar uma larga faixa de serviços críticos um grupo, região ou país rival. Estes atacantes empregam uma grande porção de tempo e recursos para planejar quais os serviços essenciais de uma região causam maior impacto econômicos em situações de falha ou que podem paralisar um país. Os alvos em potencial destes ataques podem incluir departamentos civis executivos, serviços críticos conectados com a Internet, tais como as redes elétricas inteligentes, os sistemas de monitoramento de tráfego e/ou os serviços de provimento de gás. Estes atacantes podem ser considerados muito bem treinados equipados com amplos recursos.

Escopo e Classificação

Os ataques DDoS são classificados de acordo com a camada onde ocorrem. De modo geral, estes ataques ocorrem na camada de rede/trasporte ou em nível de aplicação. A seguir são descritos os principais ataques DDoS nestas camadas. Os ataques DDoS em nível de rede/transporte podem ser volumétricos, em nível de sistema operacional, reflexão e amplificação. Os ataques DDoS volumétricos focam em interromper a conectividade dos usuários legítimos ao exaurir a largura de banda da rede da vítima, por exemplo, gerando inundações, fabricando tráfego através de protocolos como *Transmission Control Protocol* (TCP), *User Datagram Protocol* (UDP) ou *Internet Control Message Protocol* (ICMP). Os ataques DDoS em nível de sistema operacional se aproveitam da forma como os protocolos são implementados no sistema operacional para ocasionar uma sobrecarga, um exemplo muito conhecido deste tipo de ataque consiste no *ping* da morte [Douligeris and Mitrokotsa 2004]. Nos ataques DDoS baseados em reflexão, os atacantes usualmente enviam requisições forjadas (tais como, *ICMP echo request*) ao invés de direcionar para os refletores, assim esses refletores enviam suas respostas para a vítima [Paxson 2001], exaurindo seus recursos (por exemplo, ataques *Smurf* e *Fraggle*). Os ataques de amplificação são conhecidos por empregar serviços ou dispositivos para aumentar a intensidade dos efeitos de negação de serviço. As principais técnicas utilizadas consistem em responder requisições com mensagens muito maiores ou com múltiplas respostas, visando amplificar o tráfego que a vítima processará. As *botnets* têm sido constantemente usadas para propósitos de reflexão e amplificação, principalmente considerando as *botnets* móveis formadas por dispositivos da IoT [Santos et al. 2017]. As técnicas de reflexão e amplificação são empregadas usualmente em casos de ataques *Smurf*, onde os atacantes

enviam requisições com endereço *Internet Protocol* (IP) de origem falsificado (reflexão) para um grande número de refletores através de mensagens *broadcast* (amplificação).

Os ataques DDoS em nível de aplicação focam na interrupção dos serviços oferecidos para usuários legítimos ao exaurir os recursos dos dispositivos, por exemplo, *sockets*, CPU, memória, disco/base de dados e largura de banda de entrada e/ou saída. Quando comparados com os ataques DDoS em nível de rede/transporte, os ataques DDoS executados em nível de aplicação geralmente consomem menos largura de banda, sendo mais fácil de ocultar no tráfego de rede legítimo. Entretanto, os ataques DDoS de inundação em nível de aplicação usualmente possuem o mesmo impacto nos serviços alvos, pois eles são projetados considerando características específicas dos protocolos de aplicação, tais como o HTTP, MQTT ou DNS. Os ataques DDoS em nível de aplicação são categorizados como baseados em amplificação/reflexão e baseados em geração de carga explorando características do protocolo HTTP. Os ataques em nível de aplicação baseados em amplificação/reflexão usam as mesmas técnicas aplicadas nos níveis de rede/transporte, tais como, enviar requisições forjadas de protocolos em nível de aplicação para um grande número de refletores. Por exemplo, a amplificação de ataque DNS emprega técnicas de reflexão e amplificação. Os atacantes geram pequenas consultas DSN forjando o endereço de IP de origem no qual podem gerar um grande volume de tráfego de resposta. Na sequência, este grande volume de tráfego é redirecionado para o sistema alvo com o objetivo de paralisá-lo.

Existem quatro principais tipos de ataques DDoS baseados em geração de carga HTTP, inundação de sessão, inundação de requisições, assimétricos e com requisições e/ou respostas lentos [Wang et al. 2015]. Nos ataques DDoS de inundação de sessão as taxas de requisição de conexão de sessão dos atacantes são maiores que as requisições dos usuários legítimos. Assim, as requisições maliciosas exaurem os recursos dos servidores. Um dos mais famosos ataques desta categoria consiste no ataque de inundação GET/POST, o qual gera uma grande quantidade de requisições HTTP do tipo GET/POST para o servidor web vítima [Nazario 2015]. Os métodos GET e POST estão previstos no protocolo HTTP para possibilitar a interação dos clientes e servidores [Fielding et al. 1999]. Através do método GET os clientes recuperam conteúdos dos servidores e através do método POST os clientes enviam dados para os servidores. Os atacantes exploram estes métodos para sobrecarregar um servidor HTTP vítima usualmente empregam *botnets*. Através destes ataques, cada *bot* gera uma grande quantidade de requisições HTTP GET/POST válidas, sem necessariamente falsificar endereços IP.

Nos ataques de inundação de requisições, os atacantes enviam sessões que contém um número maior de requisições do que o usual, gerando uma inundação no servidor. Um dos ataques mais conhecidos desta categoria consiste na inundação de requisições HTTP GET/POST com sessão única. Este ataque consiste em uma variação do ataque de inundação de requisições HTTP GET/POST que utiliza características da versão 1.1 do HTTP para permitir múltiplas requisições dentro de uma única sessão HTTP. Assim, o atacante pode limitar a taxa de sessão de um ataque HTTP e ultrapassar mecanismos de defesa que limitam a taxa da sessão de muitos sistemas de segurança.

Nos ataques assimétricos os atacantes enviam sessões que contem requisições com alta carga de trabalho. Os ataques mais famosos desta categoria compreendem na inunda-

ção através de múltiplas requisições HTTP *GET/POST* e na injeção de códigos *Structured Query Language* (SQL). No primeiro caso o atacante cria múltiplas requisições HTTP ao formar um único pacote. Desta forma, o atacante pode manter altas cargas de trabalho no servidor vítima com uma baixa taxa de pacotes, tornando o atacante quase invisível para técnicas de detecção de anomalias no tráfego de rede. Através da injeção de código SQL, os usuários maliciosos tiram vantagem de sites Web com um projeto pobre ou integração imprópria com a base de dados. Quando uma vítima é identificada com estas características, o atacante injeta códigos SQL para executar consultas complexas no banco de dados, consumindo uma grande quantidade de recursos do servidor, tais como memória e CPU.

Os ataques DDoS com requisição/resposta lentos objetivam iniciar diversas sessões em um servidor alvo, obrigando-o a mantê-las. Os ataques desta categoria contam com rotinas programadas para realizar requisições periódicas explorando o limite máximo de tempo de vida das sessões. Ao incrementar significativamente o número de sessões em um sistema vítima, um usuário malicioso obriga sua vítima a consumir grande parcela de seus recursos computacionais, resultando na eventual indisponibilidade dos serviços prestados para usuários legítimos. Os ataques mais conhecidos desta categoria são o *Slowloris*, a fragmentação HTTP, *Slowpost* e *Slowreading* [Zargar et al. 2013].

1.5. Principais Defesas

Manter um sistema em rede estável e seguro compreende na maior motivação que levam os profissionais em redes a desenvolverem sistemas de defesa. As principais linhas de defesa ao desenvolver um sistema de segurança em redes são prevenção, reação e tolerância [Lima et al. 2009]. Esta seção apresenta as principais defesas para os ataques contra redes IoT baseadas em SDN obedecendo esta organização lógica. A Subseção 1.5.1 descreve as defesas preventivas. A Subseção 1.5.2 aborda as defesas reativas. A Subseção 1.5.3 detalha as defesas capazes de tolerar os ataques.

1.5.1. Prevenção

As técnicas prevenção procuram se antecipar aos ataques, ou seja, implementam soluções que impedem ou predizem o acontecimento de ataques. Esta subseção apresenta as principais técnicas utilizadas para sistemas de prevenção e os principais trabalhos que propõe a utilização dessas técnicas para IoT baseada em SDN.

Autenticação e Criptografia

Os dispositivos precisam se autenticar para estabelecer uma comunicação confiável em nível de segurança e estabilidade da conexão. Essa autenticação identifica o dispositivo ou a origem dos dados e define os parâmetros da criptografia para as partes obterem acesso às informações íntegras e privadas, garantindo os pilares da confidencialidade, integridade e disponibilidade. A maior parte dos sistemas de autenticação compreendem a troca de chaves de criptografia entre os dispositivos. No entanto, essas técnicas normalmente exigem dos dispositivos muito poder computacional (capacidade de processamento) e energia. Isso ocorre porque para gerar uma chave de criptografia os dispositivos realizam cálculos de tal forma que os dados calculados só podem ser acessados através de uma chave. Essa chave consiste em uma sequência de caracteres e pode ser uma informação única do

dispositivo, um código de acesso ou uma informação que as partes compartilham. A principal motivação que leva ao desenvolvimento de soluções de autenticação compreende na possibilidade do vazamento das chaves de autenticação, pois a obtenção da chave por parte do atacante implica no acesso legítimo das informações de um sistema até mesmo da estrutura da rede.

As soluções capazes de definir como os dispositivos devem se organizar durante a autenticação são interessantes para garantir a autenticação segura dos dispositivos e usuários. Seguindo esta linha, Sciancalepore *et al.* desenvolveram o OAuth-IoT, um *framework* para controle de acesso baseado em padrões abertos [Sciancalepore et al. 2017]. Neste trabalho a rede IoT compreende diversos dispositivos inteligentes interligados por uma conexão sem fio de baixa potência. Nesta rede, um dispositivo coordenador desempenha as atividades de *gateway* e coordena os clientes (por exemplo, estabelecimento de canal TLS com o cliente, autenticação e controle de acesso) em conjunto com um servidor de autorização, como proposto pelo *framework* OAuth2.0. Então, as tabelas do *gateway* são atualizadas como consequência da mútua autenticação dos os dispositivos. A partir desta etapa, o servidor de autenticação realiza o controle dos recursos disponíveis e instala regras de roteamento para o *gateway* acessar os recursos. Neste *framework* o usuário também se autentica para poder solicitar um dos recursos disponíveis. Estas autenticações são realizadas através de *tokens*, assim, quando um usuário solicitar recursos, ele recebe um *token* do servidor para acessar os recursos disponíveis através do *gateway*.

[Bernabe et al. 2016] propõem um sistema de controle de acesso flexível e confiável para IoT (TACIoT). O sistema é baseado em um *framework* de segurança para manutenção da privacidade [Bernabe et al. 2014]. Este *framework*, propõe uma versão melhorada do Modelo de Referência Arquitetural (ARM) [Bassi et al. 2013], onde inclui um gerenciador de contexto, um gerenciador de identidades e um gerenciador de grupos. Através desta estrutura, o TACIoT propõe um controle de acesso flexível e leve, com um modelo de confiança multidimensional capaz de quantificar as dimensões de forma eficiente para aplicar a lógica *fuzzy*. Os nós utilizam como parâmetro de autenticação informações contextuais, como a quem eles pertencem.

Análise de Dados e Predição dos Ataques

A análise de dados consiste no processo de identificação e avaliação dos dados capazes de descrever as características mais relevantes sobre o comportamento do sistema alvo de um ataque DDoS. Dessa forma, a análise de dados compreende as seguintes etapas: (*i*) medição dos dados, (*ii*) identificação das características e (*iii*) aplicação de artefatos (ex. estatísticos). A etapa de medição ou registro dos dados extraí os dados da rede, sendo executada no início do processo de análise. A etapa de identificação das características depende da definição de objetivos de análise e foca em características diretamente relacionadas com aspectos referentes à ataques DDoS como exemplo, tamanho de pacotes, volume de requisições, entre outras. A etapa de aplicação de artefatos consiste em selecionar as técnicas como exemplos modelos estatísticos ou matemáticos para compor o *board* de artefatos aplicados na análise de dados. Durante a condução destas etapas torna-se crucial desempenhar de modo satisfatório o registro, a extração, a manipulação dos dados e a seleção das ferramentas adequadas no processo de predição de ataques DDoS.

A predição de ataques DDoS se apropria de técnicas utilizadas para a predição

do comportamento de sistemas complexos e dinâmicos. Em geral, esses sistemas são voláteis, metaestáveis e também dispõem de significativos volumes de dados. As redes de dados se assemelham aos sistemas complexos devido à alta dinamicidade do seu fluxo de dados. Diversas pesquisas estão em andamento para desenvolver a predição de ataques DDoS. Estas pesquisas geralmente exploram técnicas estatística, inteligência artificial ou aprendizagem de máquina para antecipar o comportamento do fluxo dos dados em redes de computadores. O estudo de [Xiao et al. 2006] propõem um sistema para alertar a ocorrência de um ataque DDoS em seus estágios iniciais. O sistema possui um módulo cliente e outro servidor. Nesta pesquisa, o sistema emprega um filtro de *Bloom* modificado para monitorar *handshakes* anormais e utiliza uma estrutura de dados baseada em *hash*. O filtro de *Bloom* identifica se determinado elemento pertence a um conjunto de dados e armazena a informação de forma probabilística emitindo o alerta do ataque DDoS em seus estágios iniciais. Outra proposta observada em [Tsai et al. 2010] apresenta um sistema de alertas antecipados de multicamadas para a geração de alertas antecipados baseado em redes neurais e implementado sobre um IDS. Neste sistema, os computadores atuam de forma colaborativa para monitorar seus nós vizinhos. Cada nó da rede envia os dados coletados para um módulo que analisa e compara combinações com os padrões de DDoS.

O trabalho de [Kwon et al. 2017] sugere um método para estimar a quantidade de ataques DDoS. Os pesquisadores visam superar os limites impostos pelos sistemas de segurança reativos na detecção de intrusão e avaliar a necessidade de implantação de sistemas IPSs na rede. Os autores estimam o número de *bots* com base no número de usuários da rede em associação com os dados disponibilizados por uma pesquisa que informa a porcentagem de *bots* estimada para o país. O método proposto estima a quantidade de ataques DDoS na rede por meio de regressão e de correlação ao considerar as características do tráfego da rede, o número de usuários e a estimativa de *bots*. A pesquisa de [Nijim et al. 2017] propõe um sistema para predizer e previnir os ataques na camada de aplicação. O sistema proposto prioriza requisições legítimas em detrimento do tráfego de ataque através de um mecanismo automático de priorização baseado na classificação das requisições e no histórico do uso dos recursos como memória, espaço em disco, tempo de CPU e tráfego da rede.

Com a aplicação de modelos orientados a dados que capturam o comportamento temporal e espaço-temporal dos ataques DDoS, caracterizados por seus comportamentos e dinâmicas, o estudo de [Wang et al. 2017] aplica modelos estatísticos para predizer as características comportamentais de *botnets* e ocorrência de ataques DDoS. A predição é realizada por meio de análises de engenharia reversa sobre traços de ataques DDoS observados a partir de operações de mitigação calculam correlações temporais, espaciais e espaço-temporais das propriedades de ataques (ex. número de *bots* e duração do ataque). Outra técnica passível de ser aplicada na predição de ataques DDoS é exposta por [Azzouni and Pujolle 2017] e é caracterizada pelo uso de rede neural sobre um modelo de séries temporais (*Long Short-Term Memory*) para processar, classificar e prever a matriz de tráfego em grandes redes. Como a matriz de tráfego tem entre suas aplicações favorecer o gerenciamento da rede, esta coopera com a detecção de anomalias. Assim sendo, é possível predizer a matriz de tráfego aplicando-a na predição de ataques DDoS.

Nas pesquisas de [Zan et al. 2009, Holgado et al. 2017], são propostos métodos baseados em Modelos Ocultos de Markov (*HMM - Hidden Markov Model*) para predi-

zer ataques de múltiplas etapas, como ataques DDoS. As múltiplas etapas compreendem as fases realizadas durante um ataque, como busca de vulnerabilidade e de preparação. Dessa forma, os métodos objetivam prever os próximos passos dos ataques com base nos passos anteriores e dados históricos. Para esse fim, o processo estocástico oculto do modelo de Markov foi demonstrado pela sequência de diferentes etapas de ataques observados nos alertas emitidos por IDSs. Esses alertas são transformados em observações e agrupados em *clusters* conforme a intensidade. Uma vez treinado, o modelo é capaz de predizer a probabilidade dos passos dos ataques.

[Nanda et al. 2016] adotaram uma abordagem baseada em algoritmos de aprendizagem de máquina para predizer padrões de ataques em redes SDN. Os algoritmos foram treinados com base em dados históricos de ataques a rede para identificar conexões com potencial malicioso e potenciais alvos do ataque. Entre os algoritmos aplicados no estudo estão *C4.5, Bayesian Network, Decision Table e Naive-Bayes* para predizer o *host* que será atacado com base nos dados históricos da rede. Na pesquisa de [Mousavi and St-Hilaire 2015], a proposta é detectar ataques DDoS contra controladoras SDN em seus estágios iniciais. Dessa forma, aplicam o conceito de entropia em que define um valor limite com base em simulação. Na etapa seguinte é calculado um valor de entropia baseando-se em uma janela de 50 pacotes. Se o resultado da entropia calculada for menor que o valor limite estipulado, um alerta é enviado indicando a detecção do ataque.

As soluções propostas por estes estudos têm características que demandam conhecimento prévio estado da rede, de registros históricos do fluxo de dados ou ainda de treinamento prévio de algoritmos de aprendizagem de máquina. Assim, inviabiliza a predição de ataques DDoS desconhecidos ou que ainda não foram identificados seus padrões. Os dispositivos que compõem a IoT geram volumes de fluxo de dados significativos. Dessa forma a técnica de predição de ataques apresentada por meio do sistema STARK (*Prediction **S**ys**T**em against DDoS **A**ttack on Netwo**RK***) [Pelloso et al. 2018], contribui com a prevenção de ataques DDoS em redes IoT através da identificação antecipada da iminência de ataques, minimizando ou evitando os seus impactados.

O sistema STARK segue três etapas: (*i*) medições e preparação dos dados, (*ii*) cálculo dos indicadores estatísticos, e (*iii*) análise dos indicadores e emissão de alertas. As entradas do sistema compreendem dados de medições da rede contendo características (*features*), como por exemplo tamanho dos pacotes, número de requisições. Com base nas medições, o sistema extrai as *features* relevantes para o cálculo dos indicadores a fim de compor as séries temporais. A composição das séries temporais se dá através dos dados brutos do fluxo da rede, ou seja, não se limitam de maneira específica ao fluxo proveniente da vítima, para a vítima, ou por um endereço IP e porta específicos. Após a composição das séries temporais, o sistema analisa os valores dos indicadores.

A etapa de **medições e preparação dos dados** engloba *(i) a coleta do fluxo de dados da rede*, *(ii) o estabelecimento do tamanho da janela de tempo* e *(iii) a filtragem dos dados* (extração da característica). Dessa forma, o projeto do sistema STARK requer que a interface de rede do *hardware* subjacente opere em modo promíscuo. As análises ocorrem sobre janelas de coleta de tamanho N. O tamanho da janela de dados deve ser definido por tempo, por exemplo, X segundos (ou minutos). O tamanho da janela se ajusta automaticamente, uma vez que pode sofrer variações conforme o volume de dados

do fluxo da rede.

De acordo com os dados contidos na janela de tamanho N , o sistema STARK efetua o processo de cálculo dos indicadores estatísticos. Desta forma, se o sistema emitir a mensagem que indica ausência de recurso computacional para calcular sobre os dados que compõem a janela ou ainda que os dados são insuficientes para alimentar os indicadores, o STARK descarta o conjunto de dados dessa janela e submete uma nova janela de tamanho N+P ou N-P (onde P é o valor percentual sobre N), ampliando ou reduzindo o tamanho da janela conforme a demanda, auto ajustando o valor de N. Dessa forma, a filtragem dos dados ocorre sobre cada janela de onde são extraídas as características desejadas (ex. tamanho do pacote). As séries temporais são compostas com base nos dados contidos na janela de tamanho N e as amostras são utilizadas como entrada para a função F. A função F extrai as características desejadas através de parâmetros específicos, neste estudo, o tamanho dos pacotes. A saída da função F gera séries temporais evidenciando o tamanho dos pacotes (em *bytes*) relacionado ao respectivo tempo T. Assim, essas séries temporais são as saídas da etapa de medição e preparação dos dados e servem como entrada para a etapa de cálculo dos indicadores estatísticos.

Cada série temporal resultante da etapa anterior é utilizada como entrada da etapa de **cálculo dos indicadores estatísticos**. Nesta etapa são calculados os indicadores *taxa de retorno, auto correlação, coeficiente de variação e assimetria*. O processo de cálculo ocorre com a conversão dos valores amostrados na série temporal em um vetor (V). O sistema faz uso de um parâmetro W para expressar em porcentagem uma janela de rolamento. Essa janela tem a função de indicar a fração do conjunto de dados contido no vetor V que é utilizado como carga para o início do cálculo dos respectivos indicadores estatísticos. Em geral, é utilizado como valor padrão para a janela de rolamento (W) cinquenta por cento das amostras contidas no vetor. Assim, o sistema submete as observações contidas no vetor v e o valor W que determina a janela de rolamento a cada um dos indicadores. O resultado desta etapa mostra a tendência de comportamento de cada um dos indicadores, que pode ser crescente ou decrescente, bem como a intensidade desta tendência através do respectivo *Kendall tau* do indicador estatístico. Tais parâmetros são descritos por kTR (*Kendall tau* da taxa de retorno), kAC (*Kendall tau* da auto correlação), kCV (*Kendall tau* do coeficiente de variação), kAS (*Kendall tau* da assimetria).

Na etapa de **análise dos indicadores e emissão de alerta** o sistema STARK avalia se os indicadores estatísticos apresentam determinado comportamento para emissão do alerta. Quando a taxa de retorno apresenta uma tendência decrescente, a auto correlação e o coeficiente de variação demonstram uma tendência crescente em associação com a assimetria. Esta, ao revelar tanto uma tendência crescente ou decrescente ao deslocar as médias para os extremos, indica que a rede está com dificuldades de ser recuperar de uma perturbação, neste caso, perturbações provocadas por atividades pré-ataque, como exemplo, varreduras na rede. Ao observar este comportamento associado dos indicadores o sistema emite um alerta para informar que a rede está sofrendo ruptura no seu estado metaestável, ou seja, saindo de um estado metaestável para outro, e que por isso há a aproximação de um ataque DDoS. O sistema avalia o *Kendalltau* dos indicadores para avaliar a intensidade da tendência da ruptura. Os indicadores são avaliados em conjunto conforme a função limiar resultante da Equação 1. Quando valor resultante da função limiar FL for maior ou igual a 0,7 o sistema emite o alerta.

$$FL = \frac{-(kTR) + (kAC) + (kCV) + \sqrt{(kAS^2)}}{nTI} \quad (1)$$

Dessa forma, o sistema STARK reconhece tendências de aproximação de ataques DDoS com base no comportamento de indicadores estatísticos genéricos, ou seja, sem o conhecimento prévio do estado da rede. Por meio da antecipação de um ataque DDoS, ferramentas de monitoramento da rede, podem disparar gatilhos para ajustes das configurações em recursos específicos como exemplo um *firewall*, com objetivo de parametrizar a rede para contingenciar, evitar ou conter o ataque.

1.5.2. Reação

Os sistemas de reação a ataques compreendem a detecção do ataque e a reação do sistema contra ele. Para isso, esses sistemas realizam verificações para a detecção de anomalias e assinaturas. Estas verificações compreendem a análise do fluxo de rede, identificação das fontes do tráfego malicioso e a realização do fechamento das portas de acesso ou bloqueio das sessões. Esta subseção apresenta um estudo sobre os sistemas de identificação de dispositivos e do tráfego malicioso junto com as formas de impedir eventuais ameaças.

[Bull et al. 2016b] propuseram o uso de um *gateway* SDN como meio distribuído de monitoramento do tráfego originado e direcionado para dispositivos baseados em IoT. Esse *gateway* detecta comportamentos anômalos e executa uma resposta apropriada, tais como bloquear, encaminhar ou aplicar a Qualidade de Serviço. Essa abordagem abrange ataques baseados em inundações TCP e ICMP. [Nobakht et al. 2016] propõem o IoT-IDM, um *framework* de identificação de intrusão e mitigação para IoT. Este *framework* utiliza a SDN para fornecer proteção em nível de rede para dispositivos inteligentes implantados em ambientes domésticos. O IoT-IDM monitora as atividades de rede dos dispositivos inteligentes planejados e investiga se há alguma atividade suspeita ou mal-intencionada. Uma vez que uma intrusão é detectada, o conjunto de funções propostos pelo *framework* bloqueia o invasor em tempo real. Os autores empregam técnicas personalizadas de aprendizado de máquina para detecção com base em padrões de assinatura de ataques conhecidos. Em [Le et al. 2011], os autores propuseram outra abordagem baseada em especificação, focada na detecção de ataques contra o RPL. O comportamento do RPL foi modelado em uma máquina de estados finitos para monitorar a rede e detectar ações maliciosas. Uma extensão desta proposta foi apresentada em [Le et al. 2016] com o objetivo de suportar arquivos de rastreamento de simulação e gerar a máquina de estados finitos para o protocolo RPL. Além disso os autores também apresentaram uma técnica estatística para traçar perfis de comportamento da rede. Por fim, o autor une todos os estados e transições com as estatísticas correspondentes e implementa como um conjunto de regras para detectar dispositivos atuando maliciosamente. [Cervantes et al. 2015] desenvolveu o sistema de detecção de invasão INTI (**I**nstrusion detection for Si**N**khole attacks over 6LoWPAN for Interne**T** of Th**I**ngs). O INTI visa prevenir, detectar e isolar os efeitos dos ataques de roteamento sumidouro. Para isso, o autor utilizou a combinação de uma estratégia de monitoramento com técnicas de reputação e confiança. Esse conjunto de técnicas oferecem propriedades de auto-organização e auto-reparo. A auto-organização visa a coordenação e cooperação dos dispositivos para a configuração da rede. A se-

gunda propriedade permite a detecção de nós suspeitos e o reagrupamento para manter a estabilidade da rede.

1.5.3. Tolerância

Os sistemas desenvolvidos para tolerar ataques utilizam técnicas de mitigação e redistribuição de tráfego para aliviar os efeitos dos ataques. Essas técnicas de mitigação compreendem o controle de portas de acesso, das rotas em rede e agrupamento entre servidores em rede. Esta subseção descreve um estudo sobre sistemas de segurança que implementam soluções tolerantes à os tráfegos maliciosos que um ataque pode causar. Em [Zhang et al. 2016], os autores contribuiram com um esquema para manter o alvo em movimento utilizando salto de portas. Através do *port hopping based DoS mitigation* (PH-DM), o controlador troca as portas de acesso dos serviços em determinados intervalos de tempo, ele também assume o papel de *gateway*, descartando as conexões com as portas dessincronizadas. O esquema implementa um algoritmo de sincronização das portas de acesso do serviço com o usuário.

[Sattar et al. 2016] calculam o limite de tráfego que cada servidor pode suportar e o tráfego que todos os servidores podem suportar juntos. Quando um servidor atinge seu limite de acessos, o controlador envia um sinal de *reset* para os *switches* responsáveis pela rota, que removem todas as tabelas de fluxo, isso força os fluxos a voltarem para o controlador, que vai reinstalá-los de forma adaptativa. [Belyaev and Gaivoronski 2014] propõem um modelo de distribuição de carga na camada 4 do modelo OSI. Visto que uma rede pode ter mais de um caminho para um mesmo destino, os autores mapearam toda a rede em canais e conforme esses canais fossem atingindo um limite de tráfego, o controlador distribui os fluxos pelos demais canais. Esta abordagem, pode solucionar o problema estrangulamento de banda que um ataque pode causar. Porém, esse modelo de distribuição também gera uma grande carga no canal de controle. [Macedo et al. 2016] desenvolveram um protocolo de mitigação dos efeitos dos ataques DDoS contra redes SDN com múltiplos controladores. O protocolo compreende três fases. A primeira fase consiste no monitoramento entre os dispositivos. Nesse monitoramento os controladores trocam mensagens a fim de identificar controladores sobrecarregados por ataques DDoS. Ao identificar, a segunda fase é iniciada, realizando uma eleição entre os controladores com o objetivo de eleger controladores mais ricos em recursos disponíveis a serem compartilhados, esses controladores são organizados em uma hierarquia; o líder, o vice-líder e o grupo de elite. Em seguida, na terceira fase o controlador líder assume parte da carga de trabalho do servidor sob ataque. Caso o controlador líder também entre em estado de sobre carga, o vice-líder realiza uma nova eleição.

1.6. Prática

Esta seção detalha um estudo de caso prático envolvendo a criação das topologias de redes SDN e a simulação de ataques DoS contra estas infraestruturas de redes. Os ataques são gerados por meio de *scripts* em *Python* existentes na literatura [Mousavi 2014]. As ferramentas utilizadas no estudo de caso consistem no *Mininet*[3], no *POX*[4] e no *Wireshark*[5].

[3]http://mininet.org

[4]https://openflow.stanford.edu/display/ONL/POX+Wiki

[5]https://www.wireshark.org/

O *Mininet* é um emulador/simulador de rede em modo texto usado na criação de topologias customizadas, sendo compatível com o protocolo *OpenFlow*. O *MiniEdit* consiste em uma interface gráfica para o *Mininet*. O POX *Controller* implementa um controlador SDN por meio da linguagem *Python*, sendo largamente utilizado na academia. O *software Wireshark* possibilita a análise do tráfego de rede gerado entre os *switches* e o controlador SDN. Estas ferramentas possibilitam criar topologias de redes SDN usando uma interface gráfica, gerar carga de trabalho e analisar o tráfego gerado.

1.6.1. Simulando uma SDN

Testar soluções para SDNs envolve atividades como simular/emular uma rede, pois experimentações evolve muitas vezes um alto investimento financeiro e de tempo. O *Mininet* foi criado por professores da Universidade de Stanford[6] para a realização de simulações de SDNs e se tornou popular por suportar o protocolo *OpenFlow* e pela eficiência nas configurações, possibilitando chegar mais próximo da realidade. O *Mininet* pode ser utilizando pela linha de comandos ou pela interface gráfica (*Miniedit*). Inicialmente, vamos analisar o *Miniedit* bem como as suas principais funcionalidades. Em seguida, vamos utilizar o Mininet para simular uma SDN.

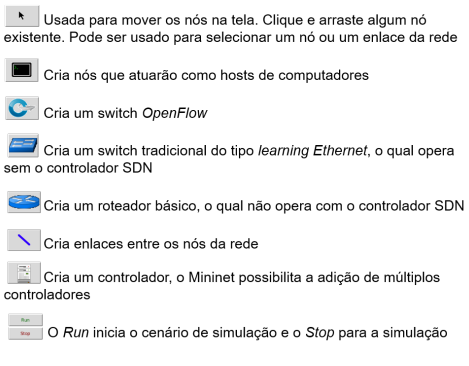


Figura 1.11: Funcionalidades do Miniedit

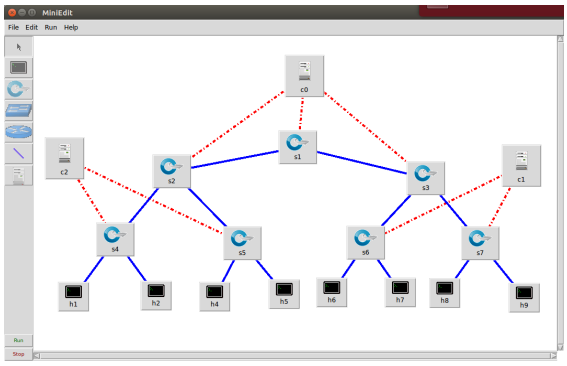


Figura 1.12: Topologia SDN no Miniedit

Executando o Miniedit

Para iniciar o *Miniedit* digite o seguinte comando:

```
$ sudo ./mininet/examples/miniedit.py
```

Este comando abrirá a interface do *Miniedit*. O *MiniEdit* tem uma interface de usuário simples. No lado esquerdo da janela, encontra-se uma lista de ícones referentes as funcionalidades e uma barra de menu na parte superior. A Figura 1.11 explica cada funcionalidade da interface inicial.

[6] http://mininet.org/

Criando uma Topologia

Primeiro, vamos criar uma topologia ao adicionar alguns *hosts* ao cenário de rede conforme a Figura1.12. Para isto, clique no ícone do *host*, mova o ponteiro para o local na tela do *MiniEdit* onde deseja que o *host* apareça e clique novamente. Um *host* aparecerá na tela. Para adicionar mais *hosts*, com o ícone selecionado, basta continuar clicando na posição desejada. Adicione oito *switches* e três controladores usando o mesmo método: Clique na ferramenta *Switch* e adicione *switches*, clique na ferramenta controlador e adicione os controladores. Em seguida, adicione *links* entre os nós na tela. Para realizar esta tarefa, clique na ferramenta *NetLink* (representada por um traço), clique em um nó e arraste o *link* para outro nó. Por exemplo: conecte um *host* a um *switch* ou um *switch* a outro *switch*. Conecte também cada *host* em pelo menos um *switch*. Em seguida, conecte os *switches* juntos para criar uma rede. Por fim, conecte cada switch a um dos controladores. No exemplo mostrado na Figura 1.12 inserimos três controladores. Então, usaremos o controlador de referência *OpenFlow* padrão que vem embutido no *Mininet*. No entanto, precisamos configurar cada controlador para que use uma porta diferente, como mostra a Figura 1.13. Clique com o botão direito do mouse em cada controlador e selecione *Propriedades* no menu exibido. O número da porta padrão para cada controlador é 6633. Altere este valor para que os números das portas usadas pelos controladores c0, c1 e c2 sejam 6633, 6634 e 6635, respectivamente.

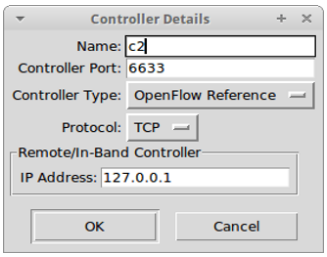


Figura 1.13: Configurando o Controlador

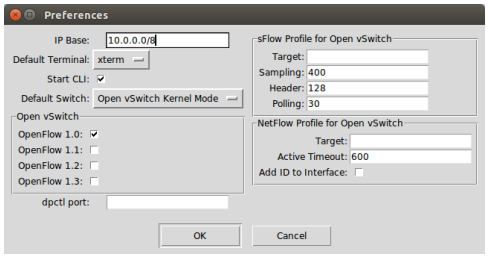


Figura 1.14: Menu *Preferences* do *Mininedit*

Ativando o CLI

O *Mininet* possibilita a realização de testes via linha de comando (CLI) durante a simulação. Usaremos esta opção em nosso estudo de caso e portanto precisamos habilitá-la. Para definir as preferências do *MiniEdit*, use o comando do menu superior como segue, *Edit* $\rightarrow$ *Preferences*. Abrirá uma caixa de diálogo como mostra na Figura1.14, marque a opção "*Start CLI*", então você terá uma CLI tal como a ilustrada na Figura 1.15.

Salvando a Topologia e Testando a Simulação

Antes de executar a simulação salve a topologia. Para salvar o arquivo *Mininet Topology* (.mn), clique em *File* na barra de menu superior e selecione *Save* no menu suspenso. Digite um nome de arquivo e salve o arquivo. Por fim, execute o cenário a ser simulado. Para desempenhar esta tarefa, clique no botão *Executar*. Na janela do terminal a partir da qual você iniciou o *MiniEdit*, você verá algumas mensagens mostrando o progresso da inicialização da simulação e em seguida, o terminal na linha de comandos do *Mininet*. Para

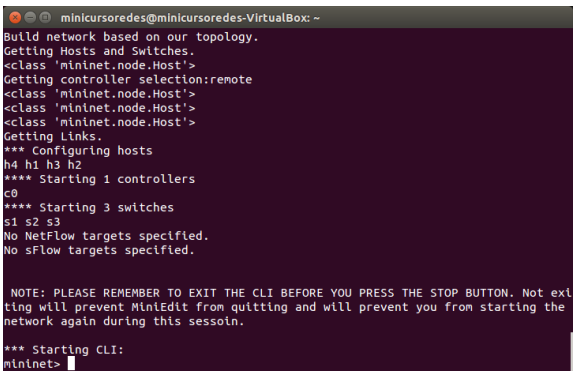


Figura 1.15: CLI do *Mininet*

verificar o funcionamento do *Mininet* vamos executar alguns comandos. Primeiramente vamos executar o comando "*pingall*", como segue:

```
mininet> pingall
```

Através deste comando todos os *hosts* enviam *pings* entre si. Através dos *pings*, os *switches* que, até então não tinham recebido nenhuma regra de fluxo, solicitam a instalação de regras para o controlador. Para verificar as regras instaladas nos *switches* é necessária a abertura de terminal virtual da máquina raiz da simulação. Para abrir o terminal o *miniedit* disponibiliza uma opção no menu superior em "*Run*"→ "*Root Terminal*". Ao executar o terminal, para solicitar a tabela de regras do *switch* 1 execute o comando abaixo.

```
# ovs-ofctl dump-flows s1
```

Simulando a Quebra de um Link

O *Miniedit* permite também a iteração com a interface gráfica durante a simulação. Para simular a quebra de um *link*, volte para a interface gráfica. A seguir com o botão direito do mouse selecione um *link* e clique na opção "*link down*", como mostra a Figura 1.16. Volte na CLI e execute um teste através do *ping*. Para executar o comando *ping* de forma individual, execute o seguinte comando:

```
mininet> h1 ping h8
```

Terminal Virtual dos *Hosts*

As redes simuladas pelo *Mininet* criam *hosts* reais com a capacidade de executar códigos e aplicativos de rede padrão *Unix/Linux*, pois ele virtualiza o *kernel* e a pilha de rede reais desses sistemas. Para acessar terminal virtual do *host* 1 basta executar o comando abaixo:

```
mininet> xterm h1
```

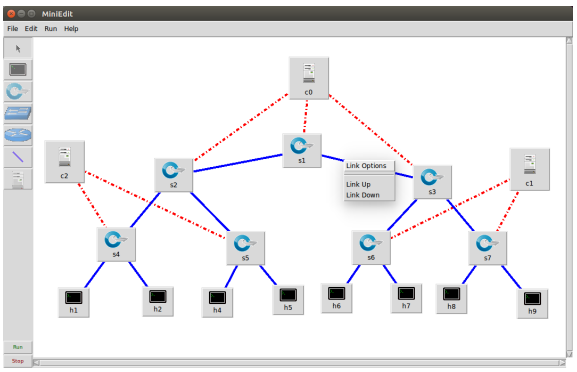


Figura 1.16: Simulando a Quebra de um *Link*

No terminal, verifique as placas de rede do *host* com o comando "*ifconfig*". Note que o *host* contém as configurações de rede semelhantes a de um sistema Linux normal. Isso capacita a realização de simulações realistas através do *Mininet*.

Topologias Customizadas

Apesar do *Miniedit* oferecer uma interface gráfica que simplifica muitas das atividades, ele não oferece todas as funcionalidades do *Mininet*. Nesta parte da simulação vamos encerrar a arquitetura criada. Para isso vamos encerrar a linha de comandos do *Mininet* com o comando abaixo. Em seguida, vamos parar a simulação através da interface gráfica, clicando no "*Stop*"(canto inferior esquerdo). Por fim, clique no menu superior em "*File*"→ "*New*".

```
mininet> exit
```

Abra novamente a topologia salva no momento anterior ("*File*"→ "*Open*"). O *Mininet* permite a criação de *scripts* customizados que permitem um nível de customização mais elevado. Com a topologia sugerida criada, ative a linha de comandos como anteriormente (1.14). Mova o mouse até o menu superior, clique nas opções "*File*"→ "*Save Level 2 Script*". Digite o nome do arquivo como "teste.py"no diretório "*home/minicursoredes/*". Por fim, vamos abrir um novo terminal, dar as devidas autorizações para o *script* salvo e executar a simulação. Os comandos ficam como segue:

```
$ sudo chmod +x teste.py
$ sudo ./teste.py
```

Note que o *script* salvo configura a topologia e o terminal do *Mininet*. A vantagem de executar um *script* de topologia personalizado *Mininet* é que você pode editar o *script* originalmente criado pelo *MiniEdit* para criar cenários mais complexos e usar os recursos do *Mininet* não suportados pelo *MiniEdit*. Execute os testes sugeridos anteriormente como o comando "*pingall*". Por fim, encerre a topologia com o comando abaixo:

```
mininet> exit
```

Topologias Instantâneas

Outra funcionalidade do *Mininet* é a possibilidade de criar topologias instantâneas. Isso permite a realização de testes rápidos que compreendam topologias simples. O comando abaixo representa uma topologia em árvore com profundidade três e dois filhos por nó. Por padrão ele inicializa a linha de comandos e o controlador embutido no *Mininet*. A opção "*–controller remote*"significa que a topologia vai esperar um controlador se conectar no *ip* 127.0.0.1, na porta 6633 (lembre-se que a barra invertida significa um espaço).

```
$ sudo mn --topo tree,depth=3,fanout=2 --controller
\ remote,ip=127.0.0.1,port=6633
```

Controlador

Os controladores desempenham o papel mais importante na estrutura de rede. No entanto existem diversas propostas para sistemas controladores SDN. No primeiro momento da parte prática utilizamos o controlador nativo do *Mininet*. Nesta seção, para tornar a simulação mais realista, vamos simular uma arquitetura com o controlador externo. Para isso, vamos utilizar o controlador POX. O Controlador POX[7] oferece uma interface de comunicação para SDN. Para isso, ele suporta os protocolos *OpenFlow* e OVSDB. O objetivo geral dos controladores SDN, incluindo o Controlador POX, é permitir o desenvolvimento de aplicações para SDN. Neste caso, aplicações desenvolvidas em *Python*. O controlador POX, vem com programas adicionais que implementam funcionalidades de redes SDN, como a instalação de regras de encaminhamento de pacotes em *switches* SDN.

Executando o Controlador

Nesta parte vamos simular uma topologia sob controle do controlador POX. Primeiramente inicie uma topologia instantânea como na demonstrado anteriormente. Execute alguns testes com o comando *ping*. Note que o comando não funciona, devido a falta do controlador. Verifique também que as tabelas de fluxos dos *Switches* estão vazias. Para executar o controlador com o componente de instalação de regras dos *switches*, inicie um terminal em paralelo e digite o comando abaixo (Note que é um único comando, a barra invertida significa um espaço). O comando executa o controlador POX com o os componentes de encaminhamento, incluindo a demonstração detalhada dos *logs* no terminal. A tela fica como mostra a Figura 1.17. Em seguida, volte ao terminal que está executando a topologia e realize os testes novamente. Note que o comando "*ping*" voltou a funcionar, também que as tabelas de fluxos dos *switches* foram preenchidas. Verifique também o terminal do controlador mostrando as ações que estão sendo executadas.

```
$ sudo ~/pox/pox.py forwarding.l3_learning info.packet_dump
\samples.pretty_log log.level --DEBUG
```

[7] https://noxrepo.github.io/pox-doc/html/

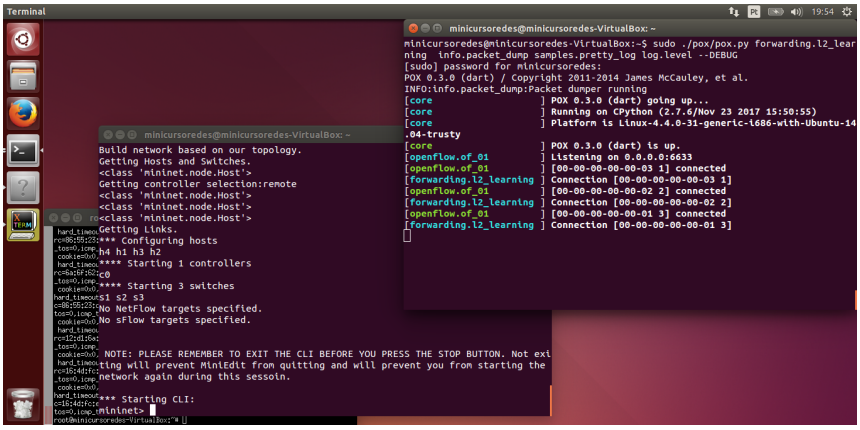


Figura 1.17: Terminal do Controlador

Wireshark

A simulação de uma rede também envolve a captura e análise dos pacotes que trafegam na rede simulada. O *Wireshark*consiste em um analisador de protocolos capaz de capturar de pacotes *OpenFlow*. Esta ferramenta também possibilita salvar capturas para a realização de análises futuras. Primeiramente vamos iniciar uma simulação através da topologia instantânea de rede e o controlador POX conforme explicado anteriormente. Em seguida, abra um terminal virtual para o *host "h1"* (*xterm h1*) e digite o comando:

```
# wireshark
```

A interface gráfica do *Wireshark* inicializa. O Wireshark oferece diversas opções de captura. Para iniciar uma captura, selecione a interface de rede "*h1-eth0*" e clique na opção "*Start*". Por fim, realize alguns testes na topologia com o comando *ping*. Note que os pacotes são capturados pela linha do tempo do *Wireshark*[8].

1.6.2. Simulando e Analisando os Dados de um Ataque de Negação de Serviço

O desenvolvimento de soluções de segurança para redes de computadores envolve também o desenvolvimento e simulação dos ataques almejados pelo desenvolvedor. Um ataque DoS em um sistema simples pode ser desenvolvido em poucas linhas de código. Nesta seção vamos simular uma versão simples de um DoS. O *script* do ataque está disponível na pasta "Documentos"do usuário "minicursoredes"na máquina virtual disponibilizada. A função deste *script* consiste basicamente em falsificar os *IPs* do cabeçalho de pacotes UDP e os enviar repetidamente. Os *IPs* falsificados geram um trafego desconhecido pelo *switches*, forçando-os a enviarem requisições ao controlador. Para simular este ataque primeiramente inicie uma topologia instantânea, o controlador POX e o *Wireshark* como na simulação anterior. Em seguida, inicie um terminal virtual para o *host "h2"* (*xterm h2*). Por fim, inicie o *script* do ataque através do comando abaixo no terminal virtual aberto. Note o aumento na atividade dos *logs* do controlador. Execute alguns testes com comando "*ping*"e você perceberá a queda de desempenho nas respostas, indicando os efeitos do ataque DoS.

[8]Não encerre a execução para a próxima simulação

```
# ./Documentos/ataque.py
```

Analisando Dados da Rede em Busca de Padrões

A análise de dados da rede em busca de padrões é determinada pelas etapas de (*i*) medições e preparação dos dados, do (*ii*) cálculo dos indicadores estatísticos, e da (*iii*) análise dos indicadores e emissão de alertas [Pelloso et al. 2018]. Na primeira etapa o *TcpDump* efetua o registro do fluxo de dados bruto da rede, resultando em conjuntos de dados definidos em janelas de tempo T pré-determinadas.Portanto, assume-se a interface de rede operando em modo promíscuo. A janela de tempo tem como valor padrão 60 segundos, porém o sistema avalia o volume de dados gerado pelo fluxo da rede, ajustando-se de acordo com a demanda. Se as amostras de dados da janela for insuficiente o sistema expande o tempo. Contudo, se a quantidade de observações disponível na janela for grande e comprometer a capacidade de análise, o sistema realiza o ajuste, diminuindo o tempo da janela, conforme exposto na Subseção 1.6.2. Após o registro das amostras em janelas, o sistema STARK realiza a extração da característica sobre a qual os dados são analisados, utilizando o *tshark*. Neste caso, representado pelo tamanho do pacotes *versus* o tempo. Este processo resulta em uma estrutura contendo a respectiva série temporal. Com base nas séries temporais geradas, a etapa de cálculo dos indicadores estatísticos efetua o cálculo da **taxa de retorno, autocorrelação, coeficiente de variação** e **assimetria**, aplicando a biblioteca *Early Warning Signals Toolbox - EWS*, implementada sobre a ferramenta R. Para calcular os indicadores, a EWS tem como parâmetro a janela de rolamento, neste caso representada por W. Em geral, a janela W é fixada em cinquenta por cento. Esta indica a quantidade de amostras da série temporal usada como carga do indicador. O resultado desta etapa expõe a tendência e intensidade do comportamento dos indicadores estatísticos. O resultado dessa etapa é submetida à análise dos indicadores e emissão de alertas. Nesta etapa o comportamento dos indicadores é avaliado.

A operacionalização das etapas do sistema ocorre ao assumir a interface de rede em modo promíscuo, assim os dados do fluxo da rede são registrados em formato pcap. A etapa (*i*) medições e preparação dos dados consiste no registro dos dados do fluxo da rede. Para fins didáticos assumimos que o resultado dessa etapa é um *log* do fluxo da rede em formato *pcap*, conforme instruções a seguir:

```
# tcpdump -i eth0 -w capture_file
```

Ainda nesta etapa é realizada a preparação dos dados, que inclui o fracionamento do *log* em arquivos de tamanhos adequados à extração da característica a ser avaliada pelos indicadores estatísticos. Neste caso, o tamanho dos pacotes foi definido em 60 segundos. O fracionamento do *log pcap* é realizado aplicando a ferramenta *editcap* e a extração da característica para cada pacotes para submeter aos indicadores é realizada pelo *tshark*.

```
# editcap -i 60 -F pcap capture_file output-packets-XX.pcap

# tshark -r output-packets-XX.pcap -t e -T fields -e frame.time_relative -e frame.len
-E header=n -E quote=n -E occurrence=f > output.dat
```

A etapa de análise dos indicadores consiste em submeter os pacotes nomeados como .dat resultantes da etapa anterior aos indicadores estatísticos submetendo a biblioteca EWS por meio do software R, como segue:

```
data <- read.table(output.dat, header = FALSE, sep = "", col.names = c('time', 'frame_size'))
timeseries <- ts(data)
generic_ews(timeseries, winsize = 50)
```

A Figura 1.18 ilustra o resultado desta etapa demonstrando o comportamento esperado dos indicadores estatísticos indicando a aproximação de ruptura nos dados, ou seja, a aproximação de um ataque DDoS. Na etapa de (*iii*) análise dos indicadores e emissão de alertas, o sistema calcula o limiar estipulado para a emissão do alerta com base nos valores do *Kendall tau* de cada um dos indicadores e, ao superar o limiar o alerta é emitido.

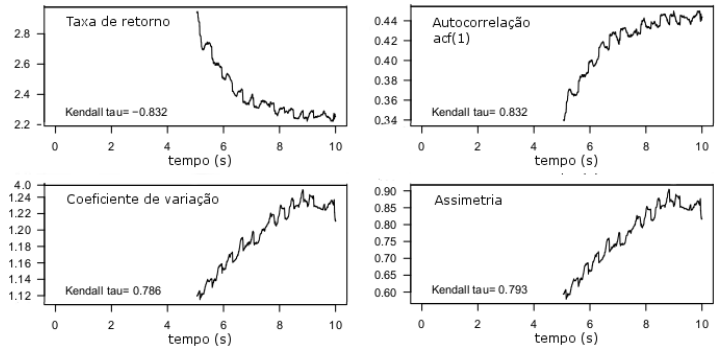


Figura 1.18: Comportamento esperado demonstrando a predição no conjunto de dados

1.7. Conclusões

As redes IoT baseadas em SDN vêm se consolidando como uma solução dos principais desafios característicos da IoT tradicional e da IoE, tais como heterogeneidade, escalabilidade e *Big Data*. No entanto, esta integração torna-se suscetível a ameaças contra a confidencialidade, a integridade e a disponibilidade dos dados, serviços e das aplicações em rede. Este capítulo apresentou as principais ameaças de segurança e defesas em redes IoT baseadas em SDN. Para alcançar este objetivo, o capítulo seguiu uma abordagem teórico/prática. Na parte teórica foram apresentados os conceitos, terminologias das redes IoT e SDN. Além disso, o capítulo detalhou as principais ameaças e defesas das redes IoT baseadas em SDN. Estes ataques foram apresentados seguindo as camadas da arquitetura das IoTs baseadas em SDNs, destacando suas especialidades. As soluções de defesa foram divididas em sistemas de prevenção, reação e tolerância, as quais apresentam diferentes técnicas para evitar, identificar ou mitigar os ataques. Na parte prática foi realizado um estudo de caso envolvendo a simulação de uma SDN sob ataque de negação de serviço, a captura e análise do tráfego gerado pelo ataque. O simulador *Mininet* foi usado para criação das topologias, juntamente com a ferramenta *MiniEdit*. Os *scripts* de ataque foram extraídos de trabalhos encontrados na literatura da área. A análise de dados foi conduzida por meio do sistema STARK em combinação com funções do *software* R.

Referências

[Myd 2009] (2009). Lazy Hacker and Little Worm Set Off Cyberwar Frenzy. https://www.computerworld.com/article/2574799/security0/mydoom-lesson--take-proactive-steps-to-prevent-ddos-attacks.html. Último Acesso: Agosto de 2018.

[Wik 2010] (2010). Wikileaks supporters disrupt Visa and MasterCard sites in 'Operation Payback'. http://www.theguardian.com/world/2010/dec/08/wikileaks-visa-mastercard-operation-payback. Último Acesso: Maio de 2018.

[Akamai 2016] Akamai (2016). Akamai's [state of the internet]/security q3 2016 report. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q3-2016-state-of-the-internet-security-report.pdf. Último Acesso: Agosto de 2018.

[Alaba et al. 2017] Alaba, F. A., Othman, M., Hashem, I. A. T., and Alotaibi, F. (2017). Internet of Things security: A survey. *Journal of Network and Computer Applications*, 88:10 – 28.

[Alkhatib et al. 2015] Alkhatib, H., Faraboschi, P., Frachtenberg, E., Kasahara, H., Lange, D., Laplante, P., Merchant, A., Milojicic, D., and Schwan, K. (2015). What Will 2022 Look Like? the IEEE CS 2022 Report. *IEEE Computer*, 48(3):68–76.

[Alrawais et al. 2017] Alrawais, A., Alhothaily, A., Hu, C., and Cheng, X. (2017). Fog computing for the Internet of Things: Security and privacy issues. *IEEE Internet Computing*, 21(2):34–42.

[Atzori et al. 2017] Atzori, L., Girau, R., Martis, S., Pilloni, V., and Uras, M. (2017). A SIoT-aware approach to the resource management issue in mobile crowdsensing. In *Innovations in Clouds, Internet and Networks*, Páginas 232–237. IEEE.

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

[Azzouni and Pujolle 2017] Azzouni, A. and Pujolle, G. (2017). A long short-term memory recurrent neural network framework for network traffic matrix prediction.

[Bandyopadhyay and Sen 2011] Bandyopadhyay, D. and Sen, J. (2011). Internet of Things: Applications and challenges in technology and standardization. *Wireless Personal Communications*, 58(1):49–69.

[Banks and Gupta 2014] Banks, A. and Gupta, R. (2014). Mqtt version 3.1. 1. *OASIS standard*, 29.

[Bartholemy and Chen 2015] Bartholemy, A. and Chen, W. (2015). An Examination of Distributed Denial of Service Attacks. In *IEEE International Conference on Electro/Information Technology*, Páginas 274–279.

[Bassi et al. 2013] Bassi, A., Bauer, M., Fiedler, M., Kramp, T., Van Kranenburg, R., Lange, S., and Meissner, S. (2013). *Enabling things to talk*. Springer.

[Belyaev and Gaivoronski 2014] Belyaev, M. and Gaivoronski, S. (2014). Towards load balancing in SDN-networks during DDoS-attacks. In *Science and Technology Conference (Modern Networking Technologies)*, Páginas 1–6. IEEE.

[Bera et al. 2017] Bera, S., Misra, S., and Vasilakos, A. V. (2017). Software-Defined Networking for Internet of Things: A survey. *IEEE Internet of Things Journal*, 4(6):1994–2008.

[Bernabe et al. 2014] Bernabe, J. B., Hernández, J. L., Moreno, M. V., and Gomez, A. F. S. (2014). Privacy-preserving security framework for a social-aware Internet of Things. In *International conference on ubiquitous computing and ambient intelligence*, Páginas 408–415. Springer.

[Bernabe et al. 2016] Bernabe, J. B., Ramos, J. L. H., and Gomez, A. F. S. (2016). TACIoT: multidimensional trust-aware access control system for the Internet of Things. *Soft Computing*, 20(5):1763–1779.

[Bizanis and Kuipers 2016] Bizanis, N. and Kuipers, F. A. (2016). SDN and virtualization solutions for the Internet of Things: A survey. *IEEE Access*, 4:5591–5606.

[Botta et al. 2016] Botta, A., De Donato, W., Persico, V., and Pescapé, A. (2016). Integration of cloud computing and Internet of Things: a survey. *Future Generation Computer Systems*, 56:684–700.

[Bradley et al. 2013] Bradley, J., Reberger, C., Dixit, A., Gupta, V., and Macaulay, J. (2013). Internet of Everything (IoE): Top 10 insights from cisco's IoE value at stake analysis for the public sector. Analysis, Cisco. Último Acesso: Agosto de 2018.

[Bull et al. 2016a] Bull, P., Austin, R., Popov, E., Sharma, M., and Watson, R. (2016a). Flow based security for IoT devices using an SDN gateway. In *International Conference on Future Internet of Things and Cloud*, Páginas 157–163.

[Bull et al. 2016b] Bull, P., Austin, R., Popov, E., Sharma, M., and Watson, R. (2016b). Flow based security for IoT devices using an SDN gateway. In *Future Internet of Things and Cloud*, Páginas 157–163. IEEE.

[Campbell et al. 1999] Campbell, A. T., Katzela, I., Miki, K., and Vicente, J. (1999). Open Signaling for ATM, Internet and Mobile Networks. *ACM SIGCOMM Computer Communication Review*, 29(1):97–108.

[Casado et al. 2007] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking Control of the Enterprise. *ACM SIGCOMM Computer Communication Review*, 37(4):1–12.

[CERT.br 2018] CERT.br (2018). Estatísticas do cert.br – incidentes de segurança. `https://www.cert.br/stats/incidentes/`. Último Acesso: Agosto de 2018.

[Cervantes et al. 2015] Cervantes, C., Poplade, D., Nogueira, M., and Santos, A. (2015). Detection of sinkhole attacks for supporting secure routing on 6lowpan for internet of things. In *IM*, Páginas 606–611.

[Cisco 2018] Cisco (2018). Cisco annual security report. `https://www.cisco.com/c/dam/assets/global/DE/unified_channels/partner_with_cisco/newsletter/2015/edition2/download/cisco-annual-security-report-2015-e.pdf`. Último Acesso: Agosto de 2018.

[Cisco 2016] Cisco, C. V. N. I. (2016). Cisco visual networking index: Global mobile data traffic forecast update, 2016-2021 white paper. Último Acesso: Agosto de 2018.

[Criscuolo 2000] Criscuolo, P. (2000). Distributed Denial of Service, Tribe Flood Network 2000, and Stacheldraht CIAC-2319, Department of Energy Computer incident Advisory Capability (CIAC). Technical report, UCRL-ID-136939, Rev. 1., Lawrence Livermore National Laboratory.

[Douligeris and Mitrokotsa 2004] Douligeris, C. and Mitrokotsa, A. (2004). DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643 – 666.

[Erickson 2013] Erickson, D. (2013). The Beacon OpenFlow Controller. In *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, Páginas 13–18, New York, NY, USA. ACM.

[Evans 2012] Evans, D. (2012). The Internet of Everything: How more relevant and valuable connections will change the world. *Cisco IBSG*, 2012:1–9.

[Farhady et al. 2015] Farhady, H., Lee, H., and Nakao, A. (2015). Software-Defined Networking: A Survey. *Computer Networks*, 81:79 – 95.

[Farris et al. 2018] Farris, I., Taleb, T., Khettab, Y., and Song, J. S. (2018). A survey on emerging SDN and NFV security mechanisms for IoT systems. *IEEE Communications Surveys & Tutorials*.

[Feamster et al. 2014] Feamster, N., Rexford, J., and Zegura, E. (2014). The Road to SDN: An Intellectual History of Programmable Networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98.

[Feng et al. 2017] Feng, S., Setoodeh, P., and Haykin, S. (2017). Smart home: Cognitive interactive people-centric Internet of Things. *IEEE Communications Magazine*, 55(2):34–39.

[Fielding et al. 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1.

[Flauzac et al. 2015] Flauzac, O., González, C., Hachani, A., and Nolot, F. (2015). SDN based architecture for IoT and improvement of the security. In *International Conference on Advanced Information Networking and Applications Workshops*, Páginas 688–693.

[Goodin 2016] Goodin, D. (2016). Record-breaking ddos reportedly delivered by >145k hacked cameras. `https://arstechnica.com/information-technology/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/`. Último Acesso: Agosto de 2018.

[Greenberg et al. 2005] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A Clean Slate 4D Approach to Network Control and Management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54.

[Gude et al. 2008] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: Towards an Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, 38(3).

[Holgado et al. 2017] Holgado, P., VILLAGRA, V. A., and Vazquez, L. (2017). Real-time multistep attack prediction based on hidden markov models. *IEEE Transactions on Dependable and Secure Computing*.

[Iannacci 2018] Iannacci, J. (2018). Internet of Things (IoT); Internet of Everything (IoE); tactile Internet; 5g – a (not so evanescent) unifying vision empowered byEH-MEMS (energy harvesting MEMS) and RF-MEMS (radio frequency MEMS). *Sensors and Actuators A: Physical*, 272:187 – 198.

[J. D. Case and M. Fedor and M. L. Schoffstall and J. Davin 1990] J. D. Case and M. Fedor and M. L. Schoffstall and J. Davin (1990). Simple Network Management Protocol (SNMP).

[Khan et al. 2012] Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future Internet: the Internet of Things architecture, possible applications and key challenges. In *Frontiers of Information Technology*, Páginas 257–260. IEEE.

[Kitten 2013] Kitten, T. (2013). DDoS: Lessons from Phase 2 Attacks. `http://www.bankinfosecurity.com/ddos-attacks-lessons-from-phase-2-a-5420`. Último Acesso: Maio de 2018.

[Kwon et al. 2017] Kwon, D., Kim, H., An, D., and Ju, H. (2017). DDoS attack volume forecasting using a statistical approach. In *IFIP/IEEE Symposium on Integrated Network and Service Management*, Páginas 1083–1086. IEEE.

[Lamaazi et al. 2014] Lamaazi, H., Benamar, N., Jara, A. J., Ladid, L., and Ouadghiri, D. E. (2014). Challenges of the Internet of Things: IPv6 and Network Management. In *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, Páginas 328–333.

[Le et al. 2016] Le, A., Loo, J., Chai, K. K., and Aiash, M. (2016). A specification-based IDS for detecting attacks on RPL-based network topology. *Information*, 7(2):25.

[Le et al. 2011] Le, A., Loo, J., Luo, Y., and Lasebae, A. (2011). Specification-based IDS for securing RPL from topology attacks. In *IFIP Wireless Days*, Páginas 1–3. IEEE.

[Lee et al. 2014] Lee, J., Uddin, M., Tourrilhes, J., Sen, S., Banerjee, S., Arndt, M., Kim, K.-H., and Nadeem, T. (2014). mesdn: Mobile extension of sdn. In *International workshop on Mobile cloud computing & services*, Páginas 7–14. ACM.

[Leslie et al. 2015] Leslie, I., Crosby, S., and Rooney, S. (2015). Devolved Control of ATM Networks. `https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/dcan/#pub`. Último Acesso: Agosto de 2018.

[Li et al. 2017] Li, C., Qin, Z., Novak, E., and Li, Q. (2017). Securing SDN Infrastructure of IoT–Fog Networks From MitM Attacks. *IEEE Internet of Things Journal*, 4(5):1156–1164.

[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.

[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 *Global Communications Conference (GLOBECOM), 2016 IEEE*, Páginas 1–6. IEEE.

[Mason 2018] Mason, J. (2018). Cyber security statistics. `https://thebestvpn.com/cyber-security-statistics-2018/`. Último Acesso: Agosto de 2018.

[Mckeay 2016] Mckeay, M. (2016). 620 gbps attack post mortem. `https://blogs.akamai.com/2016/10/620-gbps-attack-post-mortem.html`. Último Acesso: Maio de 2018.

[McKeown et al. 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

[Miraz et al. 2015] Miraz, M. H., Ali, M., Excell, P. S., and Picking, R. (2015). A review on Internet of Things (IoT), Internet of Everything (IoE) and Internet of Nano Things (IoNT). In *Internet Technologies and Applications (ITA)*, Páginas 219–224.

[Mitchell et al. 2013] Mitchell, S., Villa, N., Stewart-Weeks, M., and Lange, A. (2013). The Internet of Everything for cities: connecting people, process, data and things to improve the livability of cities and communities. *San Jose: Cisco*.

[Moore et al. 2001] Moore, J. T., Hicks, M., and Nettles, S. (2001). Practical programmable packets. In *IEEE INFOCOM*, volume 1, Páginas 41–50 vol.1.

[Mousavi 2014] Mousavi, S. M. (2014). Early detection of DDoS Attacks in Software Defined Networks Controller. Master's thesis, Carleton University, Ottawa, Ontario, Canada. Último Acesso: Agosto de 2018.

[Mousavi and St-Hilaire 2015] Mousavi, S. M. and St-Hilaire, M. (2015). Early detection of DDoS attacks against SDN controllers. In *Computing, Networking and Communications*, Páginas 77–81. IEEE.

[Nanda et al. 2016] Nanda, S., Zafari, F., DeCusatis, C., Wedaa, E., and Yang, B. (2016). Predicting network attack patterns in SDN using machine learning approach. In *IEEE Conference on Network Function Virtualization and Software Defined Networks*, Páginas 167–172. IEEE.

[Naraine 2002] Naraine, R. (2002). Massive DDoS Attack Hit DNS Root Servers. `http://www.cs.cornell.edu/people/egs/beehive/rootattack.html`. Último Acesso: Agosto de 2018.

[Nazario 2015] Nazario, J. (2015). BlackEnergy DDoS Bot Analysis - Arbor Networks. `http://atlas-public.ec2.arbor.net/docs/BlackEnergy+DDoS+Bot+Analysis.pdf`. Último Acesso: Maio de 2018.

[Newman 2018] Newman, L. (2018). Github survived the biggest DDoS attack ever recorded. `https://www.wired.com/story/github-ddos-memcached/`. Último Acesso: Agosto de 2018.

[Ng 2010] Ng, E. (2010). Maestro: A System for Scalable OpenFlow Control. Technical report, TSEN Maestro-Technical Report TR10-08, Rice University.

[Nieminen et al. 2015] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and Gomez, C. (2015). Ipv6 over bluetooth(r) low energy. RFC 7668, IETF. Último Acesso: Agosto de 2018.

[Nijim et al. 2017] Nijim, M., Albataineh, H., Khan, M., and Rao, D. (2017). Fastdetict: A Data Mining Engine for predecting and preventing DDoS attacks. In *Technologies for Homeland Security*, Páginas 1–5. IEEE.

[Nobakht et al. 2016] Nobakht, M., Sivaraman, V., and Boreli, R. (2016). A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow. In *Availability, Reliability and Security*, Páginas 147–156. IEEE.

[Nunes et al. 2014] Nunes, B., Mendonca, M., Nguyen, X.-N., Obraczka, K., and Turletti, T. (2014). A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys Tutorials*, 16(3):1617–1634.

[Olson 2014] Olson, P. (2014). The largest cyber attack in history has been hitting hong kong sites. `https://www.forbes.com/sites/parmyolson/2014/11/20/the-largest-cyber-attack-in-history-has-been-hitting-hong-kong-sites/#782acbf638f6`. Último Acesso: Agosto de 2018.

[Open Networking Foundation 2012] Open Networking Foundation (2012). Software-Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, Palo Alto, CA, USA.

[Palattella et al. 2013] Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., and Dohler, M. (2013). Standardized protocol stack for the Internet of (important) Things. *IEEE communications surveys & tutorials*, 15(3):1389–1406.

[Paxson 2001] Paxson, V. (2001). An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks. *ACM SIGCOMM Computer Communication Review*, 31(3):38–47.

[Pelloso et al. 2018] Pelloso, M., Vergu, A., Santos, A., Nogueira, M., et al. (2018). Um sistema autoadaptável para predição de ataques ddos fundado na teoria da metaestabilidade. In *Simpósio Brasileiro de Redes de Computadores (SBRC)*, volume 36.

[Pradhan et al. 2018] Pradhan, M., Fuchs, C., and Johnsen, F. T. (2018). A survey of applicability of military data model architectures for smart city data consumption and integration. In *IEEE World Forum on Internet of Things*, Páginas 129–134. IEEE.

[Prasad et al. 2014] Prasad, K. M., Reddy, A. R. M., and Rao, K. V. (2014). DoS and DDoS Attacks: Defense, Detection and Traceback Mechanisms - A Survey. *Global Journal of Computer Science and Technology*, 14(7).

[Prince 2013] Prince, M. (2013). The DDoS That Almost Broke the Internet. `https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet/`. Último Acesso: Agosto de 2018.

[R. Enns 2006] R. Enns (2006). NETCONF Configuration Protocol. rfc 4741. obsoleto pela rfc 6241.

[Rajan et al. 2017] Rajan, A., Jithish, J., and Sankaran, S. (2017). Sybil attack in IoT: Modelling and defenses. In *Advances in Computing, Communications and Informatics*, Páginas 2323–2327. IEEE.

[Rexford et al. 2004] Rexford, J., Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Xie, G., Zhan, J., and Zhang, H. (2004). Network-wide Decision Making: Toward a Wafer-thin Control Plane. In *Proceedings of HotNets*, Páginas 59–64.

[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).

[Sattar et al. 2016] Sattar, D., Matrawy, A., and Adeojo, O. (2016). Adaptive Bubble Burst (ABB): Mitigating DDoS attacks in Software-Defined Networks. In *Telecommunications Network Strategy and Planning Symposium*, Páginas 50–55. IEEE.

[Schatten et al. 2016a] Schatten, M., Ševa, J., and Tomičić, I. (2016a). A roadmap for scalable agent organizations in the Internet of Everything. *Journal of Systems and Software*, 115:31–41.

[Schatten et al. 2016b] Schatten, M., Ševa, J., and Tomičić, I. (2016b). A roadmap for scalable agent organizations in the Internet of Everything. *Journal of Systems and Software*, 115:31 – 41.

[Sciancalepore et al. 2017] Sciancalepore, S., Piro, G., Caldarola, D., Boggia, G., and Bianchi, G. (2017). OAuth-IoT: An access control framework for the Internet of Things based on open standards. In *IEEE Symposium on Computers and Communications*, Páginas 676–681. IEEE.

[Scott-Hayward et al. 2016] Scott-Hayward, S., Natarajan, S., and Sezer, S. (2016). A survey of security in Software Defined Networks. *IEEE Communications Surveys & Tutorials*, 18(1):623–654.

[Sezer et al. 2013] Sezer, S., Scott-Hayward, S., Chouhan, P., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., and Rao, N. (2013). Are We Ready for SDN? Implementation Challenges for Software-Defined Networks. *IEEE Communications Magazine*, 51(7):36–43.

[Shelby et al. 2014] Shelby, Z., Hartke, K., and Bormann, C. (2014). The constrained application protocol (coap). Technical report.

[Sonar and Upadhyay 2014] Sonar, K. and Upadhyay, H. (2014). A survey: DDoS attack on Internet of Things. *International Journal of Engineering Research and Development*, 10(11):58–63.

[Sudworth 2009] Sudworth, J. (2009). New 'cyber attacks' hit s Korea. `http://news.bbc.co.uk/2/hi/asia-pacific/8142282.stm`. Último Acesso: Agosto de 2018.

[Symatec 2018] Symatec (2018). 2018 Internet Security Threat Report. `https://www.symantec.com/security-center/threat-report`. Último Acesso: Agosto de 2018.

[Tayyaba et al. 2017] Tayyaba, S. K., Shah, M. A., Khan, O. A., and Ahmed, A. W. (2017). Software Defined Network (SDN) Based Internet of Things (IoT): A Road Ahead. In *Proceedings of the International Conference on Future Networks and Distributed Systems*, page 10. ACM.

[Tennenhouse et al. 1997] Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., and Minden, G. J. (1997). A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86.

[Tennenhouse and Wetherall 2002] Tennenhouse, D. L. and Wetherall, D. J. (2002). Towards an Active Network Architecture. In *DARPA Active NEtworks Conference and Exposition*, Páginas 2–15.

[Thubert et al. 2012] Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., and Pister, K. (2012). Rpl: Ipv6 routing protocol for low power and lossy networks. RFC 6550, IETF. Último Acesso: Agosto de 2018.

[Tsai et al. 2010] Tsai, C.-L., Chang, A. Y., and Ming-Szu, H. (2010). Early Warning System for DDoS Attacking Based on Multilayer Deployment of Time Delay Neural Network. *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*.

[Vijayan 2004] Vijayan, J. (2004). Mydoom lesson: Take proactive steps to prevent DDoS attacks. https://www.wired.com/2009/07/mydoom/. Último Acesso: Agosto de 2018.

[Vlacheas et al. 2013] Vlacheas, P., Giaffreda, R., Stavroulaki, V., Kelaidonis, D., Foteinos, V., Poulios, G., Demestichas, P., Somov, A., Biswas, A. R., and Moessner, K. (2013). Enabling smart cities through a cognitive management framework for the Internet of Things. *IEEE communications magazine*, 51(6):102–111.

[Wang et al. 2017] Wang, A., Mohaisen, A., and Chen, S. (2017). An adversary-centric behavior modeling of DDoS attacks. In *International Conference on Distributed Computing Systems*, Páginas 1126–1136. IEEE.

[Wang et al. 2015] Wang, Y., Liu, L., Sun, B., and Li, Y. (2015). A Survey of Defense Mechanisms against Application Layer Distributed Denial of Service Attacks. In *IEEE International Conference on Software Engineering and Service Science*, Páginas 1034–1037.

[Wani and Revathi 2018] Wani, A. and Revathi, S. (2018). Analyzing Threats of IoT networks using SDN Based Intrusion Detection System (SDIoT-IDS). In *Communications in Computer and Information Science*.

[Wired 2000] Wired (2000). Yahoo on Trail of Site Hackers. http://www.wired.com/2000/02/yahoo-on-trail-of-site-hackers/. Último Acesso: Maio de 2018.

[Xia et al. 2015] Xia, W., Wen, Y., Foh, C. H., Niyato, D., and Xie, H. (2015). A survey on Software-Defined Networking. *IEEE Communications Surveys Tutorials*, 17(1):27–51.

[Xiao et al. 2006] Xiao, B., Chen, W., and He, Y. (2006). A novel approach to detecting DDoS attacks at an early stage. *J Supercomput*.

[Zan et al. 2009] Zan, X., Gao, F., Han, J., and Sun, Y. (2009). A hidden markov model based framework for tracking and predicting of attack intention. In *International Conference on Multimedia Information Networking and Security*, volume 2, Páginas 498–501. IEEE.

[Zargar et al. 2013] Zargar, S., Joshi, J., and Tipper, D. (2013). A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys Tutorials*, 15(4):2046–2069.

[Zhang et al. 2016] Zhang, L., Guo, Y., Yuwen, H., and Wang, Y. (2016). A Port Hopping Based DoS Mitigation Scheme in SDN Network. In *International Conference on Computational Intelligence and Security*, Páginas 314–317. IEEE.

[Zhao et al. 2017] Zhao, M., Kumar, A., Chong, P. H. J., and Lu, R. (2017). A comprehensive study of RPL and P2P-RPL routing protocols: Implementation, challenges and opportunities. *Peer-to-Peer Networking and Applications*, 10(5):1232–1256.

Capítulo 2

Criptografia Assimétrica para Programadores – Evitando Outros Maus Usos da Criptografia em Sistemas de Software

Alexandre Braga (UNICAMP) e Ricardo Dahab (UNICAMP)

Abstract

The widespread misuse of cryptography in software systems is the most frequent source of cryptography-related security problems. Several misuses of cryptography have been found to be recurrent in software in general, resulting in vulnerabilities exploitable in real attacks. There is a huge gap between what cryptologists see as misuses of cryptography and what developers see as unsafe use of cryptographic technology. This chapter contributes to fill this gap by addressing the programmatic use of asymmetric (public key) cryptography by software developers with little or no experience in information security and cryptography. The text is introductory and aims at showing to software programmers, through actual examples and code snippets, the gooduses and misuses of asymmetric cryptography and facilitate further studies.

Resumo

O mau uso generalizado da criptografia em sistemas software é a fonte mais frequente de problemas de segurança relacionados à criptografia. Diversos maus usos de criptografia já são considerados recorrentes em softwares em geral, resultando em vulnerabilidades exploráveis em ataques reais. Percebe-se uma grande lacuna entre o que os criptologistas veem como maus usos de criptografia e aquilo que os desenvolvedores veem como uso inseguro da tecnologia criptográfica. Este texto contribui para preencher essa lacuna, abordando a utilização programática de criptografia assimétrica (de chave pública) por desenvolvedores de software com pouca ou nenhuma experiência em segurança da informação e criptografia. O texto é introdutório e tem o objetivo de mostrar aos programadores de software, por meio de exemplos reais e trechos de código, os bons e maus usos da criptografia assimétrica e, assim, facilitar o aprofundamento em estudos futuros.

2.1. Introdução

Ao longo dos anos, estudos [Anderson 1993, Schneier 1998, Gutmann 2002, Marlinspike 2009] têm revelado que vulnerabilidades em softwares criptográficos são causadas principalmente por defeitos de implementação e pela má gestão de parâmetros criptográficos. Assim, parece ser mais fácil e prático para os atacantes cibernéticos procurarem por falhas não apenas nas implementações em software de algoritmos criptográficos, mas também, e às vezes principalmente, nas camadas de software que encapsulam, circundam ou utilizam as implementações criptográficas, em vez de tentarem encontrar falhas nos algoritmos criptográficos.

Atualmente, o desenvolvimento de sistemas de software é caraterizado por ecossistemas de aplicativos maciçamente disponíveis, desenvolvidos, implantados e utilizados de modo distribuído, por uma população geograficamente dispersa, mas unida socialmente por interesses em comum. Neste ambiente, a criptografia surge muitas vezes como uma tecnologia habilitadora de relações (sociais e de negócios), influenciando a lógica das aplicações, resultando, por um lado, em aplicações criptograficamente seguras e, por outro lado, em maus usos criptográficos cada vez mais comuns. Sabe-se, por vários trabalhos relacionados [Akhawe et al. 2013, Alashwali 2013, Egele et al. 2013, Georgiev et al. 2012, Fahl et al. 2012, Shuai et al. 2014], da presença recorrente de diversas práticas ruins de criptografia em softwares diversos. Possivelmente, estas vulnerabilidades criptográficas foram incluídas sem intenção.

Hoje, percebe-se o aumento de interesse, tanto da indústria [Bleichenbacher et al. 2017] quanto da academia [IACR 2012, Braga 2017, Acar et al. 2017], pelos aspectos práticos, do mundo real, relacionados aos maus usos da tecnologia criptográfica que levam a vulnerabilidades exploráveis em sistemas de software. À medida que a segurança dos sistemas de software torna-se transparente para os usuários e a criptografia de qualidade está disponível para todos os desenvolvedores de software, a fonte mais comum de vulnerabilidades deixa de ser a infra-estrutura criptográfica, para se tornar o software em torno desta, escrito por desenvolvedores não especialistas no assunto. Por isto, o mau uso generalizado da criptografia em software é, possivelmente, a fonte mais frequente de problemas de segurança relacionados à criptografia [Braga 2017].

Quando estudamos [Braga 2017] como a criptografia pode ser mal utilizada em Java, mesmo com o uso correto da API criptográfica, resultando em vulnerabilidades identificáveis diretamente no código fonte, as seguintes observações emergiram empiricamente:

- Apenas um quarto das ferramentas de verificação de softwares criptográficos pode ser usado por programadores não especialistas em criptografia [Braga and Dahab 2015a].
- O mau uso de criptografia é muito frequente em comunidades de programação online, com padrões recorrentes de mau uso de APIs criptográficas [Braga and Dahab 2016].
- As ferramentas de análise estática de código fonte são capazes de detectar apenas pouco mais de um terço dos maus usos criptográficos catalogados [Braga et al. 2017a].
- Os desenvolvedores podem aprender a usar APIs criptográficas sem realmente aprender criptografia, enquanto alguns maus usos criptográficos persistem ao longo do tempo e independem da experiência dos desenvolvedores com as APIs [Braga and Dahab 2017].

Neste contexto, este novo capítulo dá continuidade ao texto anterior intitulado "Introdução à criptografia para programadores" [Braga and Dahab 2015b], desta vez com o objetivo de mostrar aos programadores de software os bons e maus usos da criptografia assimétrica, por meio de exemplos reais, contra-exemplos, trechos de código e programas ilustrativos em Java.

Este capítulo busca atender à demanda crescente por material didático voltado para a utilização correta de implementações criptográficas por programadores não especialistas em criptografia. O texto aborda a utilização programática de criptografia assimétrica por desenvolvedores de software com pouca experiência em segurança da informação e criptografia. Java é a linguagem de programação escolhida porque possui uma API padronizada e estável [Oracle a] que vem sendo usada por desenvolvedores de software por muito tempo, tendo sido adotada também pela plataforma Android [Google].

O escopo deste texto é a utilização correta de implementações criptográficas prontas, não cobrindo a implementação de algoritmos criptográficos. Ainda, o texto não oferece um tratamento exaustivo ao tema da programação criptográfica; por outro lado, ele tem a finalidade de fomentar o interesse pelo assunto e ajudar na formação de profissionais qualificados tanto na academia quanto na indústria. Todo o código fonte utilizado neste texto e no texto anterior [Braga and Dahab 2015b] pode ser encontrado no repositório indicado pela URL:

`https://bitbucket.org/alexmbraga/crypto4developers`

O restante do texto está organizado da seguinte forma. A Seção 2.2 revisita os conceitos e serviços criptográficos relacionados à criptografia assimétrica. A Seção 2.3 ilustra, com programas em Java, casos de uso comuns da criptografia assimétrica. A Seção 2.4 explica de forma programática os maus hábitos de programação insegura associados à criptografia assimétrica. A Seção 2.5 aborda as configurações inseguras do TLS do ponto de vista dos maus usos criptográficos. A Seção 2.6 conclui o capítulo.

2.2. Conceitos de criptografia assimétrica

Esta seção revisita os conceitos de serviços criptográficos assimétricos necessários para o entendimento do restante do capítulo. Os seguintes assuntos são tratados: criptografia de chave pública, encriptação assimétrica para sigilo e não-repúdio (ou irrefutabilidade) de mensagens integras e autênticas com assinaturas digitais. A seção também oferece exemplos de sistemas criptográficos assimétricos como RSA, acordos de chaves com Diffie-Hellman (DH), Criptografia de Curvas Elípticas (*Elliptic Curve Cryptography - ECC*), distribuição de chaves públicas com certificação digital e noções gerais sobre o protocolo *Transport Layer Security* (TLS).

2.2.1. Criptografia de chave pública

Há dois tipos de sistemas criptográficos que são comumente conhecidos como criptografia de chave secreta (ou simétrica) e criptografia de chave pública (ou assimétrica). Este texto trata a criptografia assimétrica (de chave publica), enquanto que a criptografia simétrica (de chave secreta) foi discutida em um texto anterior [Braga and Dahab 2015b]. Brevemente, na criptografia de chave secreta, uma única chave (um segredo compartilhado) é usada para encriptar e decriptar a informação. A criptografia de chave pública utiliza duas chaves, que são relacionadas matematicamente e construídas para trabalharem juntas. Uma das chaves do par é dita a chave privada (pessoal) e é mantida em segredo, sendo conhecida apenas pelo dono do par de chaves. A outra chave do par é dita a chave pública, porque é conhecida publicamente.

Devido à maior complexidade algébrica das operações matemáticas envolvidas, a criptografia de chave pública tradicional (associada ao algoritmo RSA) possui em geral um desempenho inferior se comparada à criptografia de chave secreta.

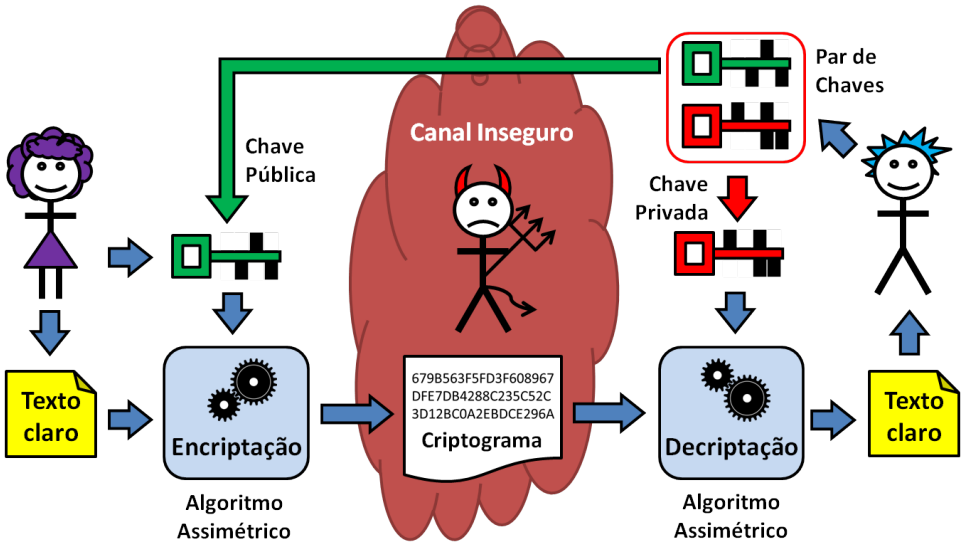


Figura 2.1. Sistema criptográfico assimétrico para sigilo (de [Braga and Dahab 2015b]).

2.2.2. Encriptação assimétrica para sigilo

A criptografia de chave pública pode ser usada para obter sigilo. Neste caso, a encriptação com a chave pública torna possível que qualquer um envie criptogramas (textos cifrados) para o dono da chave privada. A Figura 2.1 ilustra um sistema criptográfico assimétrico para sigilo e seus elementos básicos. Na figura, Ana, Beto e Ivo são os personagens. As mensagens de Ana para Beto são transmitidas por um canal inseguro, controlado por Ivo. Beto possui um par de chaves, uma chave pública e outra privada. Ana conhece a chave pública de Beto, mas somente o dono do par de chaves (Beto) conhece a chave privada (não há segredo compartilhado). A Figura 2.1 contém os passos a seguir:

1. Ana configura o algoritmo de encriptação com a chave pública de Beto;
2. Ana alimenta o algoritmo com a mensagem original (o texto claro);
3. O texto claro é encriptado e transmitido para Beto pelo canal inseguro;
4. Beto configura o algoritmo de decriptação com a sua própria chave privada;
5. O criptograma é decriptado e o texto claro original é finalmente obtido por Beto.

Analisando a Figura 2.1, observa-se que Ana envia uma mensagem privada para Beto. Para fazer isso, Ana encripta a mensagem usando a chave pública de Beto. Ana conhece a chave pública de Beto, porque ela foi divulgada por Beto. Porém, o criptograma só pode ser decriptado pela chave privada de Beto; nem Ana pode fazê-lo. Para obter comunicação segura bidirecional, basta acrescentar ao sistema criptográfico o mesmo processo no sentido oposto (de Beto para Ana), com outro par de chaves. Isto é, Beto usa a chave pública de Ana para enviar mensagens encriptadas para ela. Ana, ao receber a mensagem, usa sua própria chave privada pessoal para decriptar a mensagem enviada por Beto. A criptografia de chave pública é indispensável para o sigilo e a privacidade na Internet, pois torna possível a comunicação privada em uma rede pública.

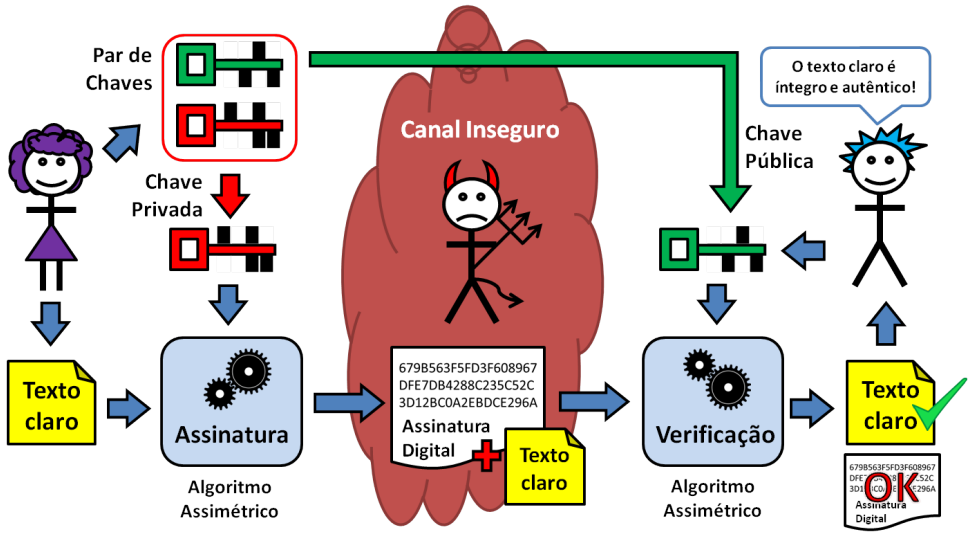


Figura 2.2. Sist. criptográfico assimétrico para autenticação (de [Braga and Dahab 2015b]).

2.2.3. Não-repúdio de mensagens integras e autênticas

A criptografia de chave pública é a base para outros dois serviços criptográficos: a autenticação das partes comunicantes e a verificação de integridade das mensagens. O procedimento operacional de utilização da criptografia de chave pública para autenticação de mensagens, em certa medida, é o oposto do uso para sigilo. A criptografia de chave pública para assinatura digital é usada para obter integridade, autenticidade e irrefutabilidade.

Chama-se assinatura digital ao resultado de uma certa operação criptográfica com a chave privada sobre o texto claro. Neste caso, o dono da chave privada pode gerar mensagens assinadas, que podem ser verificadas por qualquer um que conheça a chave pública correspondente, portanto, sendo capaz de verificar a autenticidade da assinatura digital. Do ponto de vista das implementações criptográficas subjacentes, nem sempre a operação de assinatura é uma encriptação e nem a sua verificação é sempre uma decriptação.

Visto que qualquer um de posse da chave pública pode "decriptar" a assinatura digital, ela não é mais secreta, mas possui outra propriedade: a irrefutabilidade. Isto é, quem quer que verifique a assinatura com a chave pública, sabe que ela foi produzida por uma chave privada exclusiva; logo, a mensagem não pode ter sido gerada por mais ninguém além do proprietário da chave privada.

Na Figura 2.2, Ana usa sua chave privada para assinar digitalmente uma mensagem para Beto. O texto claro e a assinatura digital são enviados por um canal inseguro (sem sigilo) e podem ser lidos por todos, por isso a mensagem não é secreta. Qualquer um que conheça a chave pública de Ana (todo mundo, inclusive Beto), pode verificar a assinatura digital. Ivo pode ler a mensagem, mas não consegue falsificá-la de maneira indetectável, pois não conhece a chave privada de Ana.

2.2.4. Exemplos de sistemas criptográficos assimétricos

Esta subseção descreve dois sistemas criptográficos assimétricos em suas formas clássicas, como descritos nos livros textos: o algoritmo RSA canônico e o acordo de chaves Diffie-Hellman sem autenticação. Além disso, a criptografia de curvas elípticas é tratada do ponto de vista funcional. A subseção termina mostrando uma comparação entre os níveis de segurança dos criptossistemas assimétricos.

2.2.4.1. O algoritmo RSA canônico

Tradicionalmente, o algoritmo criptográfico assimétrico mais conhecido e utilizado para encriptação é o RSA, cujo nome é formado pelas letras iniciais dos sobrenomes dos autores Ron Rivest, Adi Shamir e Leonard Adleman. O RSA foi publicado em 1978 [Rivest et al. 1978]. Um par de chaves do RSA é gerado da seguinte forma: (*i*) Dois números primos muito grandes, p e q, são escolhidos aleatoriamente, com $(p \neq q)$, para calcular $n = p \times q$. (*ii*) Um inteiro e é escolhido tal que $1 < e < \phi(n)$, onde e é coprimo de $\phi(n)$, com $\phi(n) = (p-1) \times (p-1)$. (*iii*) O inteiro d é calculado tal que seja o inverso multiplicativo de e ($d \times e \equiv 1 \bmod \phi(n)$). Assim, a chave pública é o par de números (e,n) e a chave privada é o par (d,n).

A encriptação com o RSA é realizada com a chave pública pela fórmula $c = m^e \bmod n$, em que m é o texto claro e c é o criptograma. A decriptação é obtida com a chave privada pela fórmula $m = c^d \bmod n$. A operação de assinatura digital é obtida com a chave privada pela fórmula $s = m^d \bmod n$, onde m é a mensagem e s é a assinatura. A verificação é realizada com a chave pública pela fórmula $m' = s^e \bmod n$, quando $m' = m$.

A segurança do RSA é baseada na dificuldade em se fatorar n em seus fatores primos quando p e q são muito grandes, hoje em dia, com no mínimo 1024 bits cada (*Integer Factorization Problem* - IFP). Um valor comumente usado para e é $2^(16)+1 = 65537$. Várias aplicações usam valores pequenos de e (tais como 3, 5 ou 35) para aumentar o desempenho da cifração e a verificação de assinaturas. Na prática, esta implementação canônica nunca deve ser utilizada. Implementações com preenchimento e aleatorização devem ser preferidas.

Para promover segurança e interoperabilidade, o uso e a implementação do RSA devem obedecer aos padrões internacionais. O documento PKCS#1v2 [Jonsson and Burt Kaliski 2003] especifica o *Optimal Asymmetric Encryption Padding* (OAEP) como um mecanismo de preenchimento (*padding*), que transforma o RSA em um mecanismo de encriptação assimétrica aleatorizado chamado RSA-OAEP. Já o *Probabilistic Signature Scheme* (PSS) é um esquema de assinaturas digitais probabilísticas com RSA (RSA-PSS) também padronizado no PKCS#1v2 [Jonsson and Burt Kaliski 2003] para substituir o esquema de assinatura do PKCS#1v1, considerado inseguro.

2.2.4.2. Acordos de chaves com Diffie-Hellman (DH)

Há ocasiões em que entidades que nunca tiveram a oportunidade de compartilhar chaves criptográficas (por exemplo, nunca se encontram, não se conhecem ou querem permanecer anônimas) precisam se comunicar em sigilo. Nestes casos, uma chave efêmera, usada apenas para algumas encriptações e decriptações decorrentes de uma conversa, pode ser gerada momentos antes do

início da conversa. Os métodos de acordo de chaves são utilizados para combinar ou negociar uma chave secreta entre dois ou mais participantes usando um canal público. Uma característica interessante destes métodos é que o segredo compartilhado (a partir do qual a chave será derivada) é combinado pela troca de informações públicas por meio de um canal inseguro.

O protocolo de acordo de chaves Diffie-Hellman (DH) foi publicado em 1976 no artigo que lançou a criptografia de chave pública [Diffie and Hellman 1976]. Com o DH, o acordo de um segredo compartilhado entre duas parte, Alice e Bob, ocorre da seguinte forma. (*i*) Alice escolhe um primo grande p e um gerador g) do corpo finito F_p. (*ii*) Alice compartilha estes valores com Bob. (*iii*) Alice escolhe um número aleatório secreto a e calcula $A = g^a \bmod p$. (*iv*) Bob também escolhe um número aleatório secreto b e calcula $B = g^b \bmod p$. (*v*) Alice e Bob trocam entre si os valores A e B. (*vi*) Alice calcula $B^a \bmod p$ enquanto Bob calcula $A^b \bmod p$. Pelas propriedades comutativas da exponenciação, $B^a = (g^b)^a = (g^a)^b = A^b = s$, onde s é o segredo compartilhado entre Alice e Bob.

A segurança do DH vem da dificuldade do adversário em resolver o problema Diffie-Hellman, variação do problema do logaritmo discreto (*Discrete logarithm Problem* - DLP): calcular o valor $(g^a)^b$ para inteiros muitos grandes e sen conhecer a e nem b. Atualmente, um primo p seguro (*safe prime*) tem pelo menos 2048 bits e deve obedecer a propriedade $p = 2q - 1$, onde q também é primo. Os parâmetros g e p podem ser estáticos (reusados em várias execuções do protocolo) ou dinâmicos (descartáveis ou efêmeros). Além disso, para evitar ataques de personificação causados por um homem no meio (*man-in-the-middle attack*) que se passa tanto por Alice quanto por Bob, apenas o DH autenticado deve ser usado.

Finalmente, o segredo compartilhado s não deve ser usado diretamente como chave secreta, mas sim ser usado na derivação de uma chave secreta segura, por exemplo, com o auxilio de uma função de resumo criptográfico.

2.2.4.3. Criptografia de Curvas Elípticas

Esta subseção oferece uma breve descrição sobre como a criptografia de curvas elíticas é usada na prática [Bos et al. 2014, Mart and Hern 2013], do ponto de vista de uma API criptográfica. O leitor interessado em aprofundar os estudos nos aspectos matemáticos e de implementação interna, deve procurar a literatura especializada [Menezes et al. 1996, Hankerson et al. 2004].

Os sistemas criptográficos baseados em curvas elípticas foram propostos por volta de 1985 de forma independente por dois pesquisadores [Miller 1985, Koblitz 1987], e se baseiam na dificuldade em se calcular o logaritmo discreto de um número inteiro longo sobre a estrutura algébrica de uma curva elíptica. Este problema é conhecido como *Elliptic Curve Discrete Logarithm Problem* (ECDLP). Em geral, a curva elíptica tem a forma $y^2 = x^3 + ax + b$ definida sobre um corpo finito primo F_p ou binário F_{2^m}. O ECDLP é considerado mais difícil que o IFP e o DLP, associados ao RSA, ao DH e ao DSA. Por este motivo, o tamanho das chaves criptográficas nos criptossistemas de curvas elípticas é consideravelmente menor que no RSA ou no DH.

Aplicações da criptografia de curvas elípticas incluem troca de chaves, assinaturas digitais e encriptação. Os esquemas e protocolos criptográficos sobre curvas elípticas mais comumente utilizados atualmente são o protocolo de acordo de chaves *Elliptic Curve Diffie Hellman* (ECDH), o esquema de assinaturas digitais *Elliptic Curve Digital Signature Algo-*

Tabela 2.1. Níveis de segurança e tamanhos de chave (de [BlueKrypt]).

Nível de segurança	Fatoração (IFP)	DLP		Curva Elíptica	Hash	
		chave	Corpo		Assinatura	KDF/HMAC/PRNG
80	1024	160	1024	160–163	SHA1 (160)	–
112	2048	224	2048	224–233	SHA-224/512 SHA3-224	–
128	3072	256	3072	256–283	SHA-256/512 SHA3-256	SHA1(160)
192	7680	384	7680	384–409	SHA-384 SHA3-384	SHA-224 SHA-512
256	15360	512	15360	512–571	SHA-512 SHA3-512	SHA-256/384/512 SHA-512/SHA3-512

rithm (ECDSA) [NIST 2013] e a encriptação *Elliptic Curve Integrated Encryption Scheme* (ECIES) [Hankerson et al. 2004].

O NIST [NIST 2013] recomenda cinco curvas elípticas sobre corpos primos para serem utilizadas no ECDSA, em cinco níveis de segurança, a saber: P-192, P-224, P-256, P-384 e P-521 (também conhecidas [SEC 2010] como secp192r1, secp224r1, secp256r1, secp384r1, secp521r1). Existe ainda uma variedade de outras curvas definidas por diversas instituições. Muitas já são consideradas inseguras para os padrões atuais enquanto outras emergem como padrões de fato em substituição a padrões obsoletos.

Em se tratando de padrões emergentes, a curva 25519 [Bernstein 2006] tem sido considerada um novo padrão de fato na indústria de segurança criptográfica por ser uma boa substituta às curvas padronizadas pelo NIST [NIST 2013], porque é mais rápida, possivelmente mais segura, e alegadamente sem a influência de agências de governo sobre seu projeto. O seu autor, Daniel Bernstein, mantém um site sobre a curva `https://cr.yp.to/ecdh.html`. Infelizmente, nenhum dos provedores criptográficos Java adotados neste texto oferece a curva 25599 diretamente pela API.

2.2.4.4. Comparação de tamanhos de chave e níveis de segurança

Sistemas criptográficos implementados em software geralmente combinam diversas funções criptográficas, cujas configurações devem ser escolhidas de modo a manter as funções no mesmo nível de segurança. A Tabela 2.1 mostra uma comparação entre os níveis de segurança de tipos de primitivas criptográficas e os tamanhos de chaves correspondentes. O nível de segurança é uma medida relativa e pode ser interpretado como sendo a força de um algoritmo criptográfico simétrico com chave de n bits.

A tabela foi adaptada do web site `keylength.com` e considera as recomendações do NIST(2016) [BlueKrypt]. Cada linha da tabela representa um nível de segurança, indo de 80 bits (para sistemas legados) até 256 bits. Por exemplo, no nível de 80 bits, atualmente útil apenas para sistemas legados, um RSA de 1024 bits é comparável ao DH de 1024 bits com chave de 160 bits, uma curva elíptica de 160 bits e hashes de 160 bits também. Já no nível de segurança mais alto, 256 bits, o RSA se torna inviável na prática com chaves de 15360 bits, enquanto as curvas elípticas despontam com chaves de 512 bits e hashes de 512 bits para assinaturas e de até 384 bits para geração de chaves.

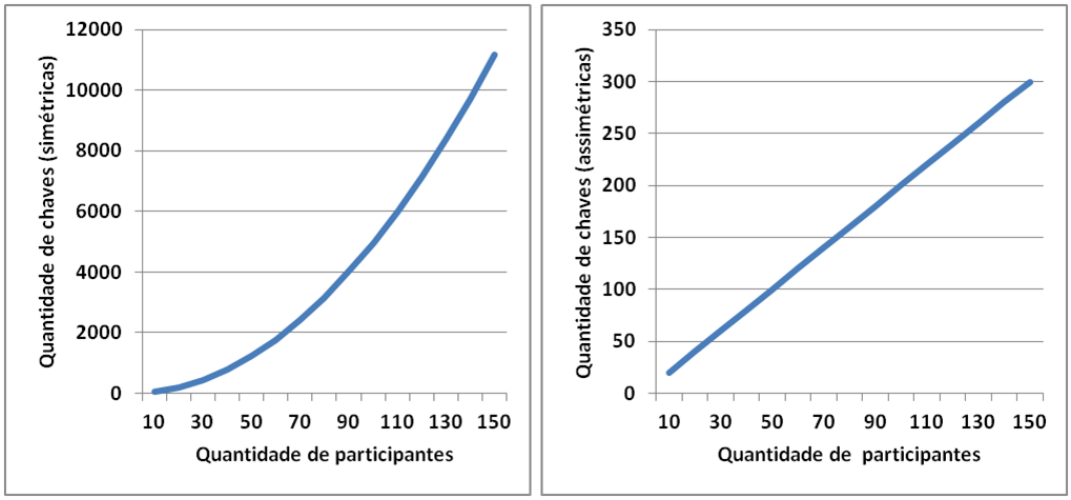


Figura 2.3. Quantidades de chaves assimétricas e simétricas para até 150 participantes.

2.2.5. Distribuição de chaves públicas com certificação digital

A criptografia assimétrica de chaves públicas tem a propriedade de simplificar o problema de distribuição de chaves criptográficas. Por exemplo, cada um dos participantes do sistema criptográfico só precisa ter o seu par de chaves, manter em segredo a sua chave privada e divulgar a chave pública. Pode haver um repositório global para todas as chaves públicas ou um mecanismo acessível para divulgá-las. A Figura 2.3 ilustra a distribuição de chaves públicas para um grupo de quatro indivíduos. O gráfico da direita na Figura 2.3 mostra o crescimento da quantidade de chaves (públicas e privadas) como uma função afim da quantidade de participantes. A quantidade de chaves é o dobro da quantidade de participantes. Já a distribuição de chaves simétricas (gráfico da esquerda) obedece a uma função quadrática.

O principal problema administrativo associado a distribuição de chaves públicas é justamente a confiança depositada na chave distribuída publicamente. Se a chave pública de alguém pode ser facilmente encontrada em qualquer lugar, então é difícil assegurar com alto grau de certeza que esta chave não foi corrompida ou substituída. O problema de garantir a integridade e a autenticidade da chave pública é muito importante e, se não for solucionado satisfatoriamente, pode comprometer a confiança no sistema criptográfico inteiro. Uma maneira de validar chaves públicas é fazer com que elas sejam emitidas por Autoridades Certificadoras (AC) de uma Infraestrutura de Chaves Públicas (ICP), do inglês *Public-Key Infrastrucuture* - PKI, que torne possível a verificação da autenticidade de tais chaves.

Certificação digital e os aspectos de validação de certificados digitais possuem atualmente boas práticas bem documentadas [Gutmann b, Gutmann a, NIST 2012] e literatura especializada na implantação de tais infraestruturas [Chandra et al. 2002]. Um certificado digital de chave pública, ou simplesmente certificado, é uma estrutura de dados que dá como verdadeiro o vínculo entre uma chave pública autêntica e uma entidade cujo nome está no certificado. A veracidade do certificado é garantida por uma terceira parte confiável, emissora do certificado, chamada de Autoridade Certificadora (CA). O certificado contém a assinatura digital da CA emissora e várias informações, tais como a chave pública, a identidade da entidade reconhecida pela CA, datas de início de uso e de validade (final de uso), etc. Por isto, o

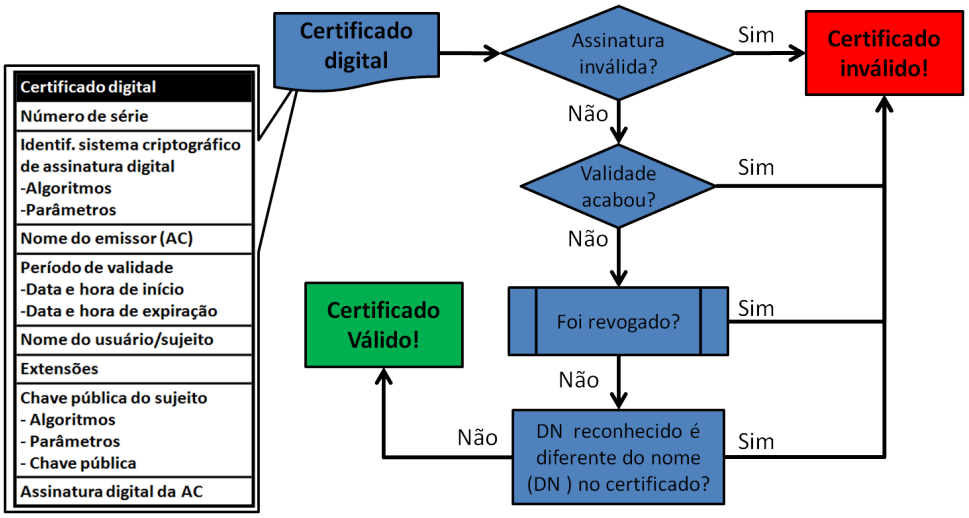


Figura 2.4. Fluxograma de validação de um certificado digital (de [Braga and Dahab 2015b]).

certificado é usado na verificação de que uma chave pública válida pertence a uma entidade e serve também como meio confiável para distribuição de chaves públicas.

A validação do certificado é realizada toda vez que a autenticidade da chave pública contida nele deve ser garantida. A assinatura da AC pode ser verificada por qualquer um com acesso à chave pública da AC, cujo certificado é amplamente disponível. Por exemplo, num caso bastante comum, uma AC pode emitir certificados SSL/TLS para servidores web que se comunicam por meio do protocolo HTTP sobre TLS (HTTPS). Quando um software cliente HTTPS (o navegador ou *browser*) faz uma requisição para um servidor web protegido, a resposta do servidor inclui o seu certificado digital. O software cliente HTTPS valida o certificado do servidor verificando a assinatura da AC emissora sobre a chave pública do servidor e outros parâmetros do certificado. Se o cliente já não possuir a chave pública da AC, ele vai buscá-la (em um repositório de chaves públicas da AC). Se o certificado é verificado como válido, então o cliente sabe que o servidor é autêntico.

A validação do certificado (e da chave pública contida nele) abrange mais etapas do que somente a verificação da assinatura da AC. Além da verificação da assinatura, o software cliente precisa verificar se o certificado não atingiu o final de seu período de validade, se não foi revogado e se o nome constante no certificado é o mesmo da parte que alega ser a dona da chave pública. O nome também deve ser obtido de um terceiro confiável, por exemplo, de um serviço de DNS seguro, no caso de nomes de domínio. A validação do certificado é ilustrada no fluxograma da Figura 2.4.

A verificação da assinatura da AC em um certificado digital exige a chave pública da AC. Para ser confiável, a chave pública da AC deve estar contida em um certificado assinado por outra AC ou autoassinado, se for uma CA raiz. As verificações sucessivas de uma sequência de assinaturas constroem uma cadeia ou hierarquia de certificados. Os certificados na base da hierarquia são assinados pelas ACs de mais baixo nível, cujos certificados são assinados pelas ACs intermediárias, que têm seus certificados assinados pelas ACs de alto nível, cujos certificados são assinados pela AC raiz, que tem seus certificados autoassinados. A Figura 2.5

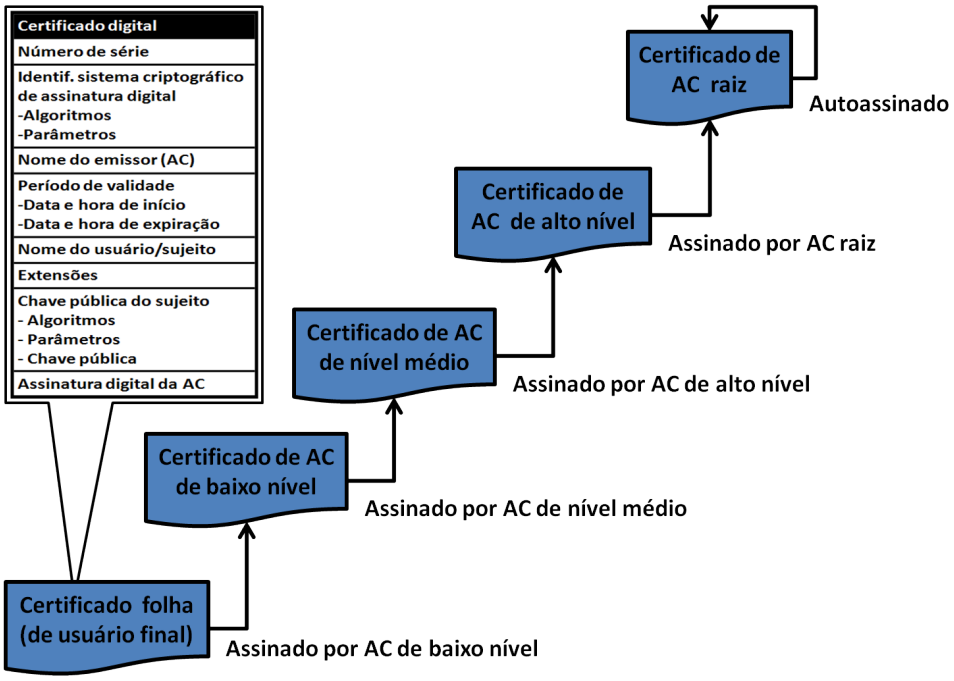


Figura 2.5. Cadeia hierárquica de certificação digital (de [Braga and Dahab 2015b]).

ilustra esta cadeia de certificação.

Uma AC revoga um certificado nas seguintes situações: quando ocorrem erros na emissão do certificado (nome grafado errado, por exemplo), ou o certificado foi emitido para uso de um serviço e o portador não tem mais acesso a ele (demissão de um funcionário), ou a chave privada do portador foi comprometida (um smartcard foi perdido), ou ainda a chave privada da AC foi comprometida (uma situação extrema). Uma Lista de Certificados Revogados (*Certificate Revocation List* - CRL) é o documento assinado digitalmente pela AC que lista o número de série de todos os certificados, ainda não expirados, que perderam a utilidade por algum dos motivos acima. Um software criptográfico pode consultar um serviço de CRL para receber atualizações periódicas, em intervalos regulares definidos por procedimentos, ou ainda consultar em tempo real se um certificado foi revogado ou não. Porém, a instabilidade do canal de comunicação pode causar indisponibilidade do serviço de validação em tempo real.

2.2.6. O protocolo SSL/TLS

Security Sockets Layer - SSL (camada de segurança de *sockets*, em tradução livre) é o protocolo para transporte de dados protegidos com criptografia sobre os *sockets* TCP. O SSL permite que a comunicação pela Internet entre aplicações cliente/servidor possa ser protegida contra monitoramento, adulteração de conteúdo e falsificação de mensagens. O objetivo principal do SSL é proporcionar um canal de comunicação seguro entre partes comunicantes. Tradicionalmente, livros de segurança em redes [Stallings 2003] são boas fontes para estudo do SSL, havendo também literatura específica [Chandra et al. 2002, Ristic 2015] sobre a implementação de código aberto mais conhecida deste protocolo, o OpenSSL [OpenSSL.org].

A terceira versão do SSL (SSLv3) serviu de base para o *Transport Layer Security* (TLS).

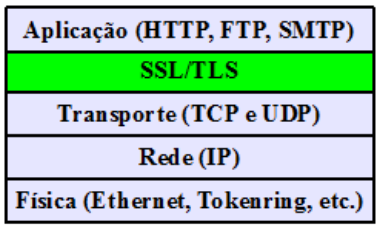


Figura 2.6. Localização do Protocolo SSL/TLS na Pilha TCP/IP.

O TLS é um protocolo padronizado pelo *Internet Engineering Task Force* (IETF) com o objetivo de substituir o formato proprietário do SSL. O SSLv3 oferece os seguintes características de segurança disponíveis até hoje no TLS:

- Autenticação do servidor com assinaturas digitais e certificados X.509, por meio dos algoritmos RSA, DSA e ECDSA, ou até por meio de uma chave simétrica pré-configurada;
- Autenticação (opcional) do cliente com assinaturas digitais e certificados X.509;
- Sigilo com encriptação da informação trocada entre cliente e servidor;
- Integridade (com uso de MACs) da informação trocada entre cliente e servidor.

Por razões históricas, o protocolo TLS também é referido como SSL/TLS. A Figura 2.6 posiciona o SSL/TLS em relação a pilha de protocolos TCP/IP. A camada de *socket* seguro fica logo acima da camada de transporte. De fato, as aplicações (na camada de aplicação) podem usar os *sockets* seguros como se fossem um serviço oferecido pela camada de transporte.

Inicialmente, o SSL foi usado para garantir o sigilo, autenticação e integridade dos dados transmitidos via uma conexão HTTP. Quando uma conexão HTTP é protegida com SSL, o protocolo é chamado HTTPS. Na autenticação do servidor SSL/TLS, para iniciar a canal seguro, o servidor HTTPS primeiro envia seu próprio certificado digital. O cliente faz verificações no certificado (estas verificações são descritas em outra seção) e indica que o certificado é válido e confiável. O canal seguro é estabelecido e o servidor envia o documento requisitado para o cliente. Já na autenticação do cliente SSL/TLS, o servidor requisita o certificado digital do cliente, que o envia ao servidor. O servidor faz verificações no certificado do cliente e indica que o certificado é válido e confiável. Em seguida, o servidor envia o conteúdo requisitado para o cliente autenticado. Este tipo de autenticação do cliente pode substituir o uso de senhas em sites web.

O SSL/TLS possui dois sub-protocolos componentes: a saudação (*handshake*), que autentica as partes e negocia os parâmetros de segurança criptográfica da comunicação, e o protocolo de registro, que usa os parâmetros escolhidos na saudação para proteger o tráfego entre as partes comunicantes. A Figura 2.7 ilustra o funcionamento da saudação (*handshake*) do SSL/TLS v1.2. As etapas do diálogo de saudação entre cliente e servidor são detalhadas e enumeradas na Figura 2.7. O objetivo principal deste diálogo é o estabelecimento de uma chave de sessão (uma chave criptográfica secreta, simétrica e temporária). O diálogo de saudação pode ser dividido em quatro partes:

1. Negociação de parâmetros (versão, ID de sessão, algoritmos criptográficos e compressão);
2. Envio do certificado do servidor e solicitação opcional do certificado do cliente;

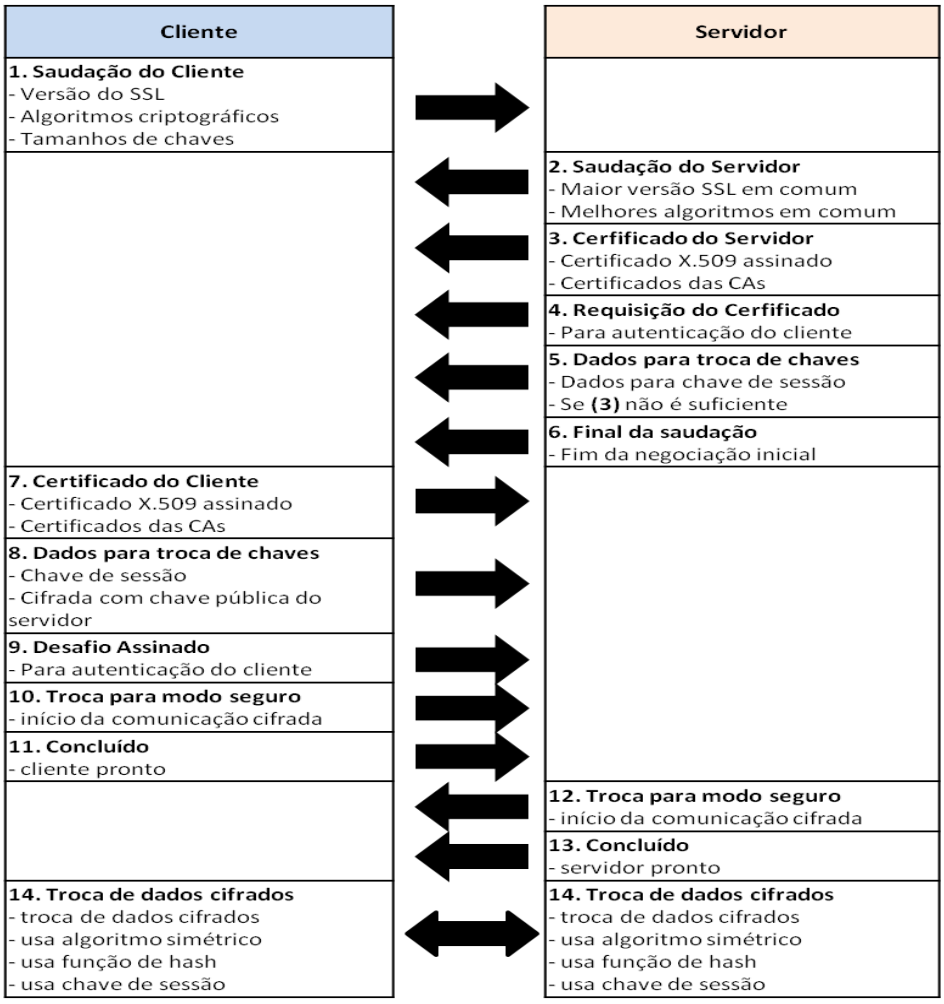


Figura 2.7. Estabelecimento de sessão e envio de mensagens SSL/TLS v1.2.

3. Envio do certificado do cliente, se solicitado;
4. Escolha de algoritmos criptográficos e finalização;

Esta implementação do *handshake* em duas rodadas é considerada [Sullivan 2018] uma causa de perda de desempenho na execução do protocolo e foi otimizada na versão 1.3. Após a realização do protocolo de saudação (*handshake*), entra em ação o protocolo de registro que realiza a troca, entre cliente e servidor, de informação encriptada segmentada em blocos chamados de registros. Este texto não trata o protocolo de registro.

O SSL/TLS é independente de aplicação e, como tal, não especifica como ele deve ser adicionado a um protocolo de aplicação. Por isto, decisões de projeto importantes são deixadas a cargo dos implementadores de protocolos de aplicação, tais como em que momento o *handshake* deve ser iniciado ou como interpretar a autenticação dos certificados digitais trocados entre as partes. Esta situação abre espaço para a inclusão de defeitos de implementação que levam a vulnerabilidades exploráveis.

2.2.7. Versões do SSL/TLS e suas vulnerabilidades

Além do SSLv3 que deu origem ao TLSv1, vale ainda mencionar outras versões do SSL/TLS que se destacam por possuírem características específicas e vulnerabilidades conhecidas. Em linhas gerais, existem seis versões em uso do SSL/TLS que podem ser encontradas em implantações reais: SSLv2, SSLv3, TLSv1.0, TLSv1.1, TLSv1.2 e TLSv1.3. As características gerais de cada uma delas são descritas a seguir:

- SSLv2 é considerado inseguro e não deve mais ser usado. Esta versão é vulnerável a ataques conhecidos, tais como o BEAST (Browser Exploit against SSL/TLS) e o DROWN (Decrypting RSA with Obsolete and Weakened eNcryption) [Aviram et al. 2016], e outras vulnerabilidades como *Cipher Suite Rollback* e *ChangeCipherSpec Message Drop*.
- SSLv3 é obsoleto e não deve mais ser utilizado. O HTTPS sobre SSLv3 é vulnerável aos ataques POODLE (Padding Oracle on Downgraded Legacy Encryption) [Möller et al. 2014] e *Lucky 13* [Fardan and Paterson 2013] e sofre de vulnerabilidades como o *Version Rollback* e o *Key Exchange Algorithm Confusion*.
- TLSv1.0 (RFC 2246) ainda é usada em sistemas legados. TLSv1.0 é vulnerável aos ataques BEAST e *Lucky 13* [Fardan and Paterson 2013]. Esta versão inicial do TLS também é vulnerável a ataques de negação de serviço e que exploram falhas na renegociação. Padrões de segurança como o PCI-DSS recomendam que esta versão seja abandonada.
- TLSv1.1 (RFC 4346) é uma versão relativamente recente e que não apresenta vulnerabilidades conhecidas sem mitigação, mas ainda oferece algoritmos criptográficos antigos.
- TLSv1.2 (RFC 5246) era a versão atual até Agosto (acabou de ser substituída pela versão 1.3) e já oferece esquemas criptográficos novos com encriptação autenticada.
- TLSv1.3 (RFC 8446) foi lançado na forma final em Agosto de 2018 [Rescorla 2018] e representa o rompimento definitivo com o passado pela eliminação de diversas características inseguras ou obsoletas mantidas até hoje por compatibilidade com legados.

Há testes específicos para implantações do SSL/TLS [Ristic 2015, Eldewahi et al. 2015, OWASP 2015]. Outros ataques contra SSL/TLS bastante conhecidos são o CRIME (*Compression Ratio info-leak Made Easy*)[CRI 2012], o BREACH (*Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext*) [BRE 2013, Gluck et al. 2013], o *Bleichenbacher Attack on PKCS#l* [Bleichenbacher 1998], os ataques ao RC4 [AlFardan et al. 2013], o Heartbleed, que afeta implementações como o OpenSSL, e o LogJam [Adrian et al. 2015, Log 2016], que afeta as implementações dos protocolos de acordo de chaves.

2.2.8. A versão 1.3 do SSL/TLS

Uma vez que a versão 1.3 do TLS [Rescorla 2018] foi lançada de fato na forma final em Agosto deste ano (2018), faz sendo incluir neste texto um breve resumo dos avanços apresentados por esta nova versão em relação a sua antecessora. Já há análises detalhadas do TLSv1.3 [Sullivan 2018] disponíveis para o público em geral. Grosso modo, a nova versão 1.3 atende a quatro objetivos de projeto: redução da latência do *handshake*, encriptação de partes do *handshake*, aumento da robustez contra ataques e remoção de funções legadas. A lista a seguir contém as principais diferenças entre o TLSv1.2 e o TLSv1.3 [Rescorla 2018]:

- A lista de opções de algoritmos simétricos foi bastante reduzida e só contém encriptação autenticada (*authenticated encryption with additional data* - AEAD). Cifras de bloco em modo CBC e a cifra RC4 foram removidos desta versão.
- O conceito de *suite* criptográfica foi simplificado e agora separa os mecanismos de autenticação e troca de chaves dos mecanismos de encriptação, *hash* usado na derivação de chaves e MACs.
- O *handshake* simplificado, em uma única rodada, acelera a saudação. Além disso, o *handshake* em zero rodadas (sem negociações, só comunicação de escolhas pré-definidas) foi incluído para atender às aplicações com restrições de tempo ainda maiores.
- RSA e DH com parâmetros estáticos foram removidos, de modo que todas as *suites* criptográficas agora possuem *forward secrecy*. Porém, só o DHE/ECDHE é usado para troca de chaves.
- Todas as mensagens do *handshake* depois do "ServerHello"agora são encriptadas e assinadas pelo servidor. esta medida protege contra os ataques práticos (FREAK, LogJam) e outros mais teóricos, como a troca maliciosa de curvas elípticas (*curveswap*).
- A criptografia de curvas elípticas faz parte da especificação principal protocolo, mas negociação de formatos de representação de pontos da curva foi removida, para incluir apenas um formato.
- Em termos de criptografia de chaves públicas, o RSA-PSS é o único *padding* do RSA permitido. O *padding* inseguro do padrão PKCS#1 v1.5 foi removido, assim como também foram removidos o DSA e a compressão.
- DH com parâmetros efêmeros customizados não é mais permitido. Há agora um número fixo de opções sabidamente seguras. Isto evita a parametrização fraca do DH que leva, por exemplo, ao ataque LogJam [Log 2016].
- A Mensagem "ChangeCipherSpec"foi removida do *handshake*, simplificando o protocolo e aumentando a robustez contra ataques. Já o *handshake* com chave pré-compartilhada *pre-shared key* (PSK) foi simplificado para aumentar a robustez contra ataques.

2.2.9. Testando uma conexão TLS com OpenSSL

O software OpenSSL [OpenSSL.org] vem com uma ferramenta cliente TLS que pode ser usada para estabelecer conexões com um servidor. O comando a seguir mostra como o cliente `s_client` usa a opção `-connect`, o nome do servidor e a porta 443 (padrão do SSL) para estabelecer uma conexão segura (na versão 1.2 `-tls1.2`) com um servidor.

```
$openssl s_client -connect exemplo.com:443 -tls1_2
```

O comportamento padrão do comando é tentar uma conexão na versão mais alta do protocolo disponível na instalação do OpenSSL. Versões específicas podem ser usadas explicitamente com as opções `-ssl2`, `-ssl3`, `-tls1`, `-tls1_1`, `-tls1_2` e `-tls1_3` (somente em distribuições novas). Além disso, versões podem ser excluídas com as opções `-no_ssl2`, `-no_ssl3`, `-no_tls1`, `-no_tls1_1`, `-no_tls1_2`.

O suporte a algoritmos específicos pode ser testado pela opção `-cipher` do comando `s_client` seguido do nome da *suite* criptográfica. Já o comando `-ciphers -s`

lista os algoritmos suportados na instalação local no OpenSSL. Os comandos a seguir listam os algoritmos disponíveis e depois testam uma conexão com `AES128-SHA`, uma configuração fraca. A opção `-cipher kECDHE` valida o uso do ECDH efêmero no acordo de chaves.

```
$openssl -ciphers -s
$openssl s_client -connect exemplo.com:443 -cipher AES128-SHA
$openssl s_client -connect exemplo.com:443 -cipher kECDHE
```

2.2.10. Considerações finais sobre SSL/TLS

A História do protocolo SSL se confunde com a história do comércio eletrônico na Internet. Na última década do milênio passado, a empresa Netscape [Wikipedia 2018] dominava tanto o mercado de softwares navegadores web (*browsers*) como o de servidores Web. Em um modelo de negócios inovador para a época, a Netscape distribuía gratuitamente o *browser*, mas ganhava muito dinheiro com a venda do seu servidor web, o único na época a possuir proteções de sigilo contra monitoramento e interceptação da comunicação. A segurança da comunicação era garantida por um protocolo criptográfico na camada de transporte do TCP/IP que ficou conhecido como SSL.

O SSL tornou possíveis as transações sigilosas entre o *browser* na máquina do cliente e o servidor web (do lojista). O uso do SSL é praticamente transparente para o cliente, o que facilitou muito a ampla adoção deste protocolo como padrão de fato para segurança na Internet. Com o SSL, os números de cartões de crédito puderam transitar pelas redes abertas com sigilo, viabilizando o comércio eletrônico como acontece hoje em dia.

Entretanto, o SSL não resolve todos os problemas de segurança. Por exemplo, uma vez que a informação cifrada do cartão de crédito chega ao lojista, ela precisa ser decifrada para que o pagamento seja efetivado junto à instituição financeira (banco ou operadora de cartão de crédito). Neste momento, as informações do cartão ficam expostas ao lojista e aos seus funcionários. O SSL não resolve este problema, pois ele trata apenas da comunicação entre *browser* e servidor web. Outros protocolos de segurança devem ser usados para proteger as informações de pagamento.

Finalmente, hoje em dia, surgem modalidades de ataques que exploram a confiança já estabelecida em sites web que possuem conexões SSL/TLS legítimas. O abuso da confiança depositada por usuários na conexão SSL/TLS entre *browser* e servidor web ocorre quando uma conexão SSL/TLS legítima é usada em ataques. Atualmente, mesmo sites web maliciosos, utilizados por exemplo em ataques de *phishing*, podem ter certificados SSL legítimos em seus servidores. Isto se deve ao fato de autoridades certificadores emitirem, de forma simplificada na Internet, certificados gratuitos e de validade curta. Estes certificados, mesmo com validade reduzida, ainda podem ser usados por tempo suficiente para viabilizar ataques de engenharia social [Calvo 2018].

2.3. Exemplos de bons usos da criptografia assimétrica

Esta seção usa programas que utilizam a API criptográfica padrão Java e a biblioteca criptográfica BouncyCastle [BouncyCastle 2018] para mostrar os seguintes conceitos: encriptação assimétrica aleatorizada, assinatura digital probabilística e validação de certificados digitais. Além disso, a seção mostra o uso dos algoritmos RSA-OAEP, RSA-PSS, ECDSA, ECIES e curvas elípticas seguras. Alguns dos programas foram adaptados de [Braga et al. 2017a].

2.3.1. Encriptação e decriptação assimétrica

Esta subseção ilustra a encriptação assimétrica e aleatorizada com dois esquemas criptográficos diferentes: RSA-OAEP e ECIES. Nos programas que se sequem, o leitor deve imaginar a interação entre os dois personagens tradicionais de criptossistemas, Ana e Beto, ou Alice e Bob.

2.3.1.1. RSA Aleatorizado: RSA-OAEP

O código na Listagem 2.1 contém o método principal (`main`) de um programa Java para encriptação com RSA aleatorizado RSA-OAEP. A linha 3 seleciona o provedor criptográfico *BouncyCastle*. A linha 4 cria um texto claro na variável `pt`. Nas linhas 6 e 7, cria-se o par de chaves RSA com 2048 bits. A linha 8 define o RSA como a implementação OAEP com *hash* SHA256 e *padding* MGF1 (o único disponível). As linhas de 9 a 12 criam duas máquinas criptográficas. Uma para encriptação (configurada com a chave pública) e outra de decriptação (usando a chave privada). Finalmente, as linhas de 14 a 17 realizam três encriptações sucessivas do mesmo texto claro para mostrar que elas são diferentes de fato. A linha 18 faz a decriptação. As linhas de 19 a 21 mostram todas as exceções tratadas para o RSA-OAEP em Java. Vale a pena mencionar as exceções de chave inválida, de tamanho de bloco inválido e de *padding* ruim ou desconhecido.

Listagem 2.1. RSA Aleatorizado com RSA-OAEP.

```
1 public static void main(String args[]) {
2   try {
3     Security.addProvider(new BouncyCastleProvider()); // provedor BC
4     byte[] pt = ("Randomized RSA").getBytes();
5
6     KeyPairGenerator g = KeyPairGenerator.getInstance("RSA", "BC");
7     g.initialize(2048); KeyPair kp = g.generateKeyPair();
8     String RSA_OAEP = "RSA/None/OAEPWithSHA256AndMGF1Padding";
9     Cipher e = Cipher.getInstance(RSA_OAEP, "BC");
10    e.init(Cipher.ENCRYPT_MODE, kp.getPublic());
11    Cipher d = Cipher.getInstance(RSA_OAEP, "BC");
12    d.init(Cipher.DECRYPT_MODE, kp.getPrivate());
13
14    U.println("Plaintext: " + U.b2x(pt));
15    U.println("Ciphertext: " + U.b2x(e.doFinal(pt)));
16    U.println("Ciphertext: " + U.b2x(e.doFinal(pt)));
17    U.println("Ciphertext: " + U.b2x(e.doFinal(pt)));
18    U.println("Plaintext: " + U.b2x(d.doFinal(e.doFinal(pt))));
19  } catch (NoSuchAlgorithmException | NoSuchPaddingException |
20           InvalidKeyException | IllegalBlockSizeException |
21           BadPaddingException | NoSuchProviderException e)
22  { System.out.println(e);}}
```

2.3.1.2. Outra maneira de usar o RSA-OAEP

O trecho de código na Listagem 2.2 mostra uma segunda maneira de utilizar o RSA aleatorizado OAEP. A linha 1 seleciona o provedor de serviços criptográficos *BouncyCastle*. As linhas 2 e 3 criam o par de chaves RSA com 2048 bits utilizado por Beto para receber mensagens encriptadas por Ana com a chave pública de Beto.

As linhas de 5 a 9 mostram as configurações comuns para Ana e Beto, em que os parâmetros do OAEP são definidos manualmente. Primeiro, a função de preenchimento MGF1 é configurada para usar a função de *hash* SHA512. Em seguida, os parâmetros do OAEP são definidos como SHA512, MGF1 (única opção disponível) e a fonte de aleatorização *default*. Finalmente, a linha 9 instancia um RSA-OAEP sem configurações ("RSA/None/OAEPPadding").

As linhas de 12 a 16 mostram como Ana encripta com a chave pública de Beto. Primeiro, nas linhas 12 e 13, a máquina de encriptação é configurada com os parâmetros do OAEP (definidos anteriormente) e a chave pública de Beto. Em seguida, na linha 14, é calculado o tamanho adequado, em bytes, do texto claro de entrada para o encriptador. Na fórmula, 2048 e 512 são os tamanhos, em bits, da chave e do *hash*, respectivamente. A linha 15 extrai um trecho do texto claro como tamanho adequado e a linha 16 realiza a encriptação.

Finalmente, nas linhas de 18 a 20, Beto usa sua chave privada (linha 18) para configurar a máquina criptográfica em modo de decriptação e com os parâmetros OAEP (linha 19), obtendo o texto claro a partir do criptograma (linha 20).

O RSA-OAEP limita o tamanho do texto claro que pode ser encriptado em uma única chamada da função. Este limite está relacionado ao tamanho do corpo finito (e da chave) usado na aritmética modular do algoritmo RSA e pode ser determinado, em bytes, pela fórmula *(ks-2*hs)/8–2*, onde *ks* é o tamanho da chave RSA em bits e *hs* é o tamanho do *hash* em bits usado pelo *padding* OAEP.

Listagem 2.2. Outro RSA Aleatorizado.

```
1 Security.addProvider(new BouncyCastleProvider());
2 KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA","BC");
3 kpg.initialize(2048); KeyPair kp = kpg.generateKeyPair();
4
5 // configuracoes comuns para Ana e Bato
6 MGF1ParameterSpec mgf1ps = MGF1ParameterSpec.SHA512;
7 OAEPParameterSpec OAEPps = new OAEPParameterSpec("SHA512","MGF1",
8         mgf1ps,PSource.PSpecified.DEFAULT);
9 Cipher c = Cipher.getInstance("RSA/None/OAEPPadding", "BC");
10
11 // Encriptacao pela Ana com a chabe publica de Beto
12 Key pubk = kp.getPublic();
13 c.init(Cipher.ENCRYPT_MODE, pubk, OAEPps);
14 int maxLenB = (2048-2*512)/8-2;//tamanho max do texto claro (bytes)
15 byte[] pt = U.cancaoDoExilio.substring(0, maxLenB).getBytes();
16 byte[] ct = c.doFinal(pt);
17
18 Key privk = kp.getPrivate();//decriptacao com chave privada do Beto
19 c.init(Cipher.DECRYPT_MODE, privk, OAEPps); //modo de decriptacao
20 byte[] ptBeto = c.doFinal(ct); // Decriptando
```

2.3.1.3. Encriptação assimétrica com curvas elípticas: ECIES

O ECIES é um esquema de encriptação com curvas elípticas [Hankerson et al. 2004]. A encriptação com ECIES é aleatorizada e autenticada, combinado no esquema criptográfico um algoritmo simétrico (como o AES) e um MAC para autenticação do criptograma. O ECIES pode ser uma alternativa ao RSA-OAEP com a vantagem de utilizar chaves menores que aqueles do RSA, mantendo o mesmo nível de segurança. Nenhum dos provedores criptográficos disponíveis na instalação da plataforma Java oferece uma implementação do ECIES, por isso foi utilizada a implementação do ECIES no provedor criptográfico *BouncyCastle* (BC).

O trecho de código da Listagem 2.3 mostra como utilizar o ECIES do provedor BC. A linha 3 adiciona dinamicamente o provedor BC e a linha 4 define o texto claro. Um gerador de par de chaves para o ECIES é instanciado na linha 5 e configurado para uma curva elíptica com chaves de 224 bits de tamanho (linha 6). Então, o par de chaves é criado na linha 6. Instancias do ECIES, para encriptação com a chave pública (linha 8) e decriptação com a chave privada (linha 10), são criadas a partir do identificador "ECIESwithAES–CBC", um ECIES com AES no modo CBC, sem *padding*.

As linhas de 13 a 16 computam duas vezes o criptograma sobre o mesmo texto claro, para mostrar que os criptogramas são diferentes. A linha 16 decripta o criptograma, recuperando o texto claro. Vale ainda observar que o criptograma gerado pelo ECIES é bem mais longo que o texto claro utilizado no exemplo. Este é uma característica da criptografia assimétrica aleatorizada, que carrega um *nonce* no criptograma, entre outras informações específicas de cada esquema criptográfico, como por exemplo a *tag* de autenticação. Este característica tem questões de armazenamento de podem inibir o uso do ECIES para proteger textos claros.

Listagem 2.3. Encriptação e decriptação com ECIES da BouncyCastle.

```
1 public static void main(String args[]) {
2  try {
3    Security.addProvider(new BouncyCastleProvider());// provedor BC
4    byte[] ptAna = ("Teste do ECIES").getBytes();
5    KeyPairGenerator g = KeyPairGenerator.getInstance("ECIES","BC");
6    g.initialize(224); KeyPair kp = g.generateKeyPair();
7
8    Cipher e=Cipher.getInstance("ECIESwithAES-CBC/NONE/NOPADDING","BC");
9    e.init(Cipher.ENCRYPT_MODE, kp.getPublic());
10   Cipher d=Cipher.getInstance("ECIESwithAES-CBC/NONE/NOPADDING","BC");
11   d.init(Cipher.DECRYPT_MODE, kp.getPrivate());
12
13   U.println("Plaintext : " + U.b2x(ptAna));
14   U.println("Ciphertext: " + U.b2x(e.doFinal(ptAna)));
15   U.println("Ciphertext: " + U.b2x(e.doFinal(ptAna)));
16   U.println("Plaintext : " + U.b2x(d.doFinal(e.doFinal(ptAna))));
17 } catch (NoSuchAlgorithmException | NoSuchPaddingException   |
18               InvalidKeyException | IllegalBlockSizeException|
19               BadPaddingException | NoSuchProviderException e)
20 { System.out.println(e); }}
```

2.3.2. Assinaturas digitais não-determinísticas

Esta subseção ilustra o uso de assinatura digital como mecanismo de autenticação de origem (autoria) de uma informação. Dois esquemas criptográficos de assinaturas digitais não-determinísticas são ilustrados: assinaturas probabilísticas com RSA-PSS e aleatorizadas com o ECDSA.

2.3.2.1. Assinaturas digitais probabilísticas com RSA-PSS

O trecho de código na Listagem 2.4 mostra como utilizar o RSA-PSS em Java. As linhas de 2 até 4 selecionam o provedor criptográfico "BC"e criam o par de chaves RSA com 3072 bits. As linhas de 7 a 10 criam uma instância da máquina de assinatura (o assinador) para o PSS, indicado pela string "SHA256withRSAandMGF1", e constroem um objeto de parâmetros do PSS com *hash* SHA256 e MGF1, de modo análogo ao RSA-OAEP.

As linhas de 12 a 15 realizam o processo de geração da assinatura digital. Primeiro, a linha 12 cria a mensagem (texto claro) que será assinada neste exemplo. Em seguida, a linha 13 inicializam o assinador com a chave privada e um gerador de números pseudoaleatórios seguro, enquanto a linha 14 alimenta o assinador com a mensagem a ser assinada e a linha 15 calcula a assinatura digital, com o método `sign()`.

Por outro lado, as linhas 17 e 20 realizam o processo de verificação da assinatura digital. Primeiro, na linha 17, a máquina de assinatura e inicializada para verificação e parametrizada com a chave pública correspondente àquela chave privada usada na assinatura. Em seguida, o verificador é alimentado com a mensagem em texto claro (Neste caso, para que o *hash* da mensagem seja recalculado). Finalmente, a linha 19 faz a verificação da assinatura digital com o método `verify()`.

Listagem 2.4. Assinatura digital aleatorizada com RSA-PSS.

```
1 public static void main(String[] args) throws Exception {
2   Security.addProvider(new BouncyCastleProvider()); // provedor BC
3   KeyPairGenerator kg = KeyPairGenerator.getInstance("RSA", "BC");
4   kg.initialize(3072, new SecureRandom());
5   KeyPair kp = kg.generateKeyPair();
6
7   Signature sig = Signature.getInstance("SHA256withRSAandMGF1", "BC");
8   PSSParameterSpec spec = new PSSParameterSpec("SHA256", "MGF1",
9         MGF1ParameterSpec.SHA256, 20, 1);
10  sig.setParameter(spec);
11
12  byte[] m = "Testing good RSA PSS".getBytes();
13  sig.initSign(kp.getPrivate(), new SecureRandom());
14  sig.update(m);
15  byte[] s = sig.sign();
16
17  sig.initVerify(kp.getPublic());
18  sig.update(m);
19  if (sig.verify(s)) { System.out.println("Verification succeeded.");}
20  else               { System.out.println("Verification failed.");    }
21 }
```

2.3.2.2. Assinaturas digitais aleatorizadas com ECDSA

O trecho de código na Listagem 2.5 é o primeiro exemplo deste texto que utiliza explicitamente as curvas elípticas. Por isto, as linhas de 2 a 5 têm novidades. A linha 2 contém a escolha da curva "secp256r1"ou NIST P-256. A linha 3 instancia um gerador de par de chaves para curvas elípticas, enquanto a linha 4 inicializa o gerador com a curva selecionada. A linha 5 gera o par de chaves. O gerador de par de chaves utiliza o provedor criptográfico "SunEC", que agrupa diversas implementações criptográficas relacionadas às curvas elípticas e está disponível na plataforma Java desde a versão 6 [Oracle b].

As linhas 7 e 8 instanciam um objeto `SecureRandom` com o melhor PRNG disponível para a implantação Java, geram uma semente de 24 bytes e configuram o objeto `SecureRandom` com ela. As linhas de 10 a 13 realizam os passos para geração de uma assinatura digital. A linha 10 instancia um objeto assinador para o esquema criptográfico ECDSA identificado por "SHA256withECDSA". A linha 11 configura o assinador com a chave privada e o PRNG. A linha 12 obtém o documento a ser assinado enquanto a linha 13 alimenta o assinador com o documento e calcula a assinatura digital.

As linhas de 15 a 21 repetem os passos de instanciação e configuração de um assinador e geração de uma assinatura digital para o mesmo documento assinado anteriormente, porém, desta vez armazenados em objetos diferentes. As duas assinaturas digitais são comparadas na linha 23, mostrando que elas são diferentes, apesar de terem sido geradas sobre o mesmo documento. Este exemplo não faz a verificação da assinatura digital. O programa que mostra a verificação de uma assinatura digital pode ser encontrado em [Braga and Dahab 2015b].

Listagem 2.5. ECDSA com nonce não repetido.

```
1 public static void main(String[] args) throws Exception {
2   ECGenParameterSpec ecps = new ECGenParameterSpec("secp256r1");
3   KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC","SunEC");
4   kpg.initialize(ecps);
5   KeyPair kpAna = kpg.generateKeyPair();
6
7   SecureRandom sr1 = SecureRandom.getInstanceStrong();
8   byte[] seed = sr1.generateSeed(24); sr1.setSeed(seed);
9
10  Signature signer1 = Signature.getInstance("SHA256withECDSA","SunEC");
11  signer1.initSign(kpAna.getPrivate(), sr1);
12  byte[] doc = U.cancaoDoExilio.getBytes();
13  signer1.update(doc); byte[] sign1 = signer1.sign();
14
15  SecureRandom sr2 =  SecureRandom.getInstanceStrong();
16  sr2.setSeed(seed);
17
18  Signature signer2 = Signature.getInstance("SHA256withECDSA","SunEC");
19  signer2.initSign(kpAna.getPrivate(), sr2);
20  doc = U.cancaoDoExilio.getBytes();
21  signer2.update(doc); byte[] sign2 = signer2.sign();
22
23  boolean ok = Arrays.areEqual(sign1, sign2);
24  if (ok){System.out.println("Nonce repeated! Signatures are equal!");}
25  else   {System.out.println("Signatures are different!");}
26 }
```

2.3.3. Curvas elípticas seguras

O provedor criptográfico "SunEC"oferece diversas opções de curvas elípticas. A Tabela 2.2 lista as curvas padronizadas e ainda consideradas seguras pelos padrões internacionais [NIST 2013, SEC 2010] e que estão disponíveis no provedor "SunEC", de acordo com [Mart and Hern 2013]. Na tabela, as curvas seguem a nomenclatura de SEC-2 [SEC 2010], com nomes no formato `sec[p|t]XXX[r|k]v`, onde `[p|t]` indica uma curva sobre corpo primo (`p`) ou binário (`t`). Ainda, o número `XXX` indica o tamanho em bits da chave e a letra `[r|k]` indica que a curva usa os parâmetros de Koblitz (`k`) ou randômicos (`r`), em uma determinada versão (`v`).

Na Tabela 2.2, a coluna da esquerda mostra o nível de segurança (em bits) e as outras duas colunas mostram as curvas elípticas sobre corpos primos F_p e binários F_{2^m}. As curvas secp224r1 (NIST P-224), secp256k1, secp256r1 (NIST P-256) e secp384r1 (NIST P384), marcadas com um asterisco, são consideradas inseguras por Daniel Bernstein [Bernstein et al. 2013]. Ele considera que estas curvas são complexas de implementar, facilitando a ocorrência de defeitos que levam a vulnerabilidades, entre outros problemas.

Tabela 2.2. Curvas elípticas seguras no provedor SunEC (de [Mart and Hern 2013]).

Segurança	Curvas sobre F_p	Curvas sobre F_{2^m}
80	–	sect163k1,sect163r1,sect163r2
96	secp192k1,secp192r1	–
112	secp224k1,*secp224r1**	sect233k1, sect233r1
115	–	sect239k1
128	*secp256k1*,secp256r1**	sect283k1, sect283r1
192	*secp384r1**	sect409k1, sect409r1
256	secp521r1	sect571k1, sect571r1

O trecho de código na Listagem 2.6 cria pares de chaves para cada uma das curvas na tabela 2.2 e lista os parâmetros das chaves. A execução do programa mostra que a chave pública é um ponto, em coordenadas (x,y), da curva elíptica, enquanto a chave privada é uma sequência de bits derivada de um número inteiro muito longo. Além disso, os nomes alternativos das curvas (quando existem) também são listados e a chave privada está codificada no formato PKCS#8.

Listagem 2.6. Selecionando as curvas seguras de SunEC.

```
1 public static void main(String argv[]) {
2  String[] curves={
3     "secp192k1","secp192r1","secp224k1","secp224r1","secp256k1",
4     "secp256r1","secp384r1","secp521r1","sect163k1","sect163r1",
5     "sect163r2","sect233k1","sect233r1","sect239k1","sect283k1",
6     "sect283r1","sect409k1","sect409r1","sect571k1","sect571r1"};
7  try { for (String curve : curves) {
8      ECGenParameterSpec ecps = new ECGenParameterSpec(curve);
9      KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC","SunEC");
10     kpg.initialize(ecps); KeyPair kp = kpg.generateKeyPair();
11     U.println("EC parameters "+ecps.getName());
12     U.println("Pub.  key: " + kp.getPublic());
13     U.println("Priv. key: " + U.b2x(kp.getPrivate().getEncoded()));
14     U.println("Algorithm: " + kp.getPrivate().getAlgorithm());
15     U.println("Format   : " + kp.getPrivate().getFormat());
16   }}catch(NoSuchAlgorithmException|InvalidAlgorithmParameterException|
17         NoSuchProviderException e){System.err.println("Error:"+e);}}
```

2.3.4. Validação completa de certificados digitais

O programa na Listagem 2.7 faz a validação de um certificado digital e sua cadeia de certificação em uma conexão segura SSL/TLS. As linhas de 4 a 6 estabelecem a conexão segura e realizam o (*handshake*) do protocolo. As linhas de 8 a 14 mostram informações da sessão segura, tais como versão do protocolo, *suite* criptográfica e os nomes do servidor e do sujeito no certificado.

Não há validação automática do nome do servidor, que deve ser feita manualmente. A comparação do nome do servidor (obtido da sessão) com o nome do sujeito no certificado é feita nas linhas 17 e 18. A validação do certificado é feita nas linhas de 20 a 24. A linha 20 obtém toda a cadeia de certificação associada ao certificado do servidor, incluindo ele próprio. A linha 22 verifica, no método `checkValididy()`, a data de validade do certificado do servidor, enquanto a linha 23 verifica a assinatura digital do certificado do servidor com o método `verify()`. Se a validação falhar, uma exceção específica é disparada (linha 24).

O restante do programa faz a verificação da cadeia de certificação assim como da revogação do certificado do servidor por meio de uma CRL. O processo possui dois passos preparatórios e um passo de verificação: (*i*) Obtenção do certificado raiz e dos outros certificados da cadeia de certificação, de modo que a cadeia possa ser completamente construída (linhas de 27 a 35). (*ii*) Designação dos certificados raízes como âncoras de confiança da cadeia (linhas 38 e 39). (*iii*) Validação da cadeia de certificação e do *status* de revogação do certificado do servidor (linhas 41 até 65). Por questões de espaço, apenas este passo é detalhado.

Nas linhas de 42 a 50, é criado um validador de cadeia de certificação (instância de `CertPathValidator`) com quatro opções configuradas: falha suavemente se não obtiver a CRL e nem a verificação por OCSP, sem mecanismo de recuperação, verifica apenas os certificados na ponta da cadeia, e prefere a verificação por CRL em vez de OCSP.

Nas linhas de 52 a 57, uma CRL associada ao certificado raiz (ancora de confiança) é recupera de modo confiável (detalhes omitidos) e adicionada aos parâmetros da verificação. Finalmente, as linhas de 60 a 65 realizam a verificação. O método `validate()` da linha 61 verifica a cadeia e a CRL passadas como parâmetros. Se exceções não ocorrem (verificação bem sucedida), as políticas de uso do certificado (linha 62) e a chave pública (linha 63) podem ser obtidas com confiança. No caso do SSL/TLS, o conteúdo da página indicada na URL é recuperado (linha 67). Senão, alguma exceção é disparada (linhas 63 e 65).

Listagem 2.7. Validação completa de Certificados.

```
1 public static void main(String[] args) throws Exception {
2  SSLSocket s = null; boolean ok = true;
3  try {
4    SSLSocketFactory f=(SSLSocketFactory)SSLSocketFactory.getDefault();
5    s = (SSLSocket) f.createSocket("www.google.com", 443);
6    s.startHandshake();//all validations happen after handshake
7
8    System.out.println("Session infos");
9    SSLSession ss = s.getSession();
10   System.out.println("Protocol: " + ss.getProtocol());
11   System.out.println("Ciphersuite: " + ss.getCipherSuite());
12   System.out.println("Host name: " + ss.getPeerHost());
13   Principal peer = ss.getPeerPrincipal();
14   System.out.println("SSL Peer Principal: " + peer);
15
```

```
16    // Cert, date, and Host validation
17    if (!peer.getName().contains("CN=" + ss.getPeerHost()))
18    {throw new CertificateException("Host and Principal mismatch");}
19
20    Certificate[] peerCerts = ss.getPeerCertificates();
21    if (peerCerts != null && peerCerts.length >= 2) {
22      ((X509Certificate) peerCerts[0]).checkValidity();
23      peerCerts[0].verify(peerCerts[1].getPublicKey());
24    } else {throw new CertificateException("Unable to verify cert.");}
25
26    // Cert Path validation
27    // Step 1. Obtain CA root certs and the cert path to validate
28    Certificate[] certs = ss.getPeerCertificates();
29    X509Certificate[] x509certs = new X509Certificate[certs.length-1];
30    for (int i = 0; i < certs.length-1; i++)
31      { x509certs[i] = (X509Certificate) certs[i];}
32    X509Certificate anchor = (X509Certificate) certs[certs.length-1];
33    List l = Arrays.asList(x509certs);
34    CertificateFactory cf=CertificateFactory.getInstance("X.509","SUN");
35    CertPath cp = cf.generateCertPath(l);
36
37    // Step 2. Create a PKIXParameters with the trust anchors
38    TrustAnchor ta = new TrustAnchor(anchor, null);
39    PKIXParameters params=new PKIXParameters(Collections.singleton(ta));
40
41    // Step 3. Use a CertPathValidator to validate the certificate path
42    CertPathValidator cpv=CertPathValidator.getInstance("PKIX","SUN");
43    PKIXRevocationChecker rc =
44            (PKIXRevocationChecker)cpv.getRevocationChecker();
45    rc.setOptions(EnumSet.of(PKIXRevocationChecker.Option.SOFT_FAIL));
46    rc.setOptions(EnumSet.of(PKIXRevocationChecker.Option.NO_FALLBACK));
47    rc.setOptions(EnumSet.of(
48                            PKIXRevocationChecker.Option.ONLY_END_ENTITY));
49    rc.setOptions(EnumSet.of(PKIXRevocationChecker.Option.PREFER_CRLS));
50    params.addCertPathChecker(rc);
51
52    // now it gets a valid CRL for Anchor
53    X509CRL crl = CertUtils.getCRL(anchor);// supposed valid CRL
54    List list = new ArrayList();list.add(crl);
55    CertStoreParameters csp=new CollectionCertStoreParameters(list);
56    CertStore store = CertStore.getInstance("Collection", csp);
57    params.addCertStore(store);
58
59    // validate certification path with specified params
60    PKIXCertPathValidatorResult cpvr
61            = (PKIXCertPathValidatorResult) cpv.validate(cp,params);
62    PolicyNode policyTree = cpvr.getPolicyTree();
63    PublicKey subjectPK = cpvr.getPublicKey();
64  } catch (CertificateException|InvalidAlgorithmParameterException|
65           NoSuchAlgorithmException|CertPathValidatorException e)
66  { System.out.println(e); ok = false; }
67  if (ok){System.out.println();CertUtils.handleSocket(s);}
68  else {System.out.println("Something wrong in cert validation.");}}
```

2.4. Maus usos criptográficos e vulnerabilidades associadas

Esta seção é organizada em torno dos maus usos de programação de criptografia assimétrica que levam a vulnerabilidades. Os maus usos são exemplificados por casos reais e ilustrados programaticamente por meio de programas em Java. Os maus usos comuns de criptografia assimétrica tratados no texto são os seguintes: encriptação determinística com RSA, problemas da versão 1 do padrão PKCS#1 (assinaturas determinísticas e *padding* inseguro), configurações fracas do RSA-OAEP, assinaturas digitais inseguras (com RSA, DSA e ECDSA), acordo de chaves DH e ECDH sem autenticação ou com chaves fracas e curvas elípticas inseguras.

2.4.1. Criptografia determinística com RSA

O trecho de código na listagem 2.8 mostra o uso indiscriminado do algoritmo RSA canônico, isto é, sem aleatorização, conforme descrito anteriormente no texto [Braga and Dahab 2015b] e na seção 2.1. O exemplo usa uma chave pequena de 1024 bits, insegura para os padrões atuais. Em Java, este mau uso pode ser identificado já na instanciação do algoritmo criptográfico, por exemplo, pelo método `getInstance(a)`, da classe `Cipher`, em que `a` é o nome do algoritmo. Há três opções inseguras para resolver o nome do algoritmo `a` para o RSA canônico: *(i)* apenas o nome do algoritmo, por exemplo, "RSA"; *(ii)* nome do algoritmo e modo ECB sem *padding*, por exemplo, "RSA/ECB/NoPadding"; e *(iii)* nome do algoritmo, sem modo e sem *padding*, por exemplo, "RSA/None/NoPadding". Vale observar que a primeira opção leva naturalmente ao erro, uma vez que, na falta de uma escolha explícita, o RSA canônico é a opção padrão implícita.

Listagem 2.8. Três maneiras de RSA determinístico.

```
1 public static void main(String args[]) {
2   try {
3     Security.addProvider(new BouncyCastleProvider()); // provedor BC
4     byte[] textoClaroAna = ("Cripto deterministica").getBytes();
5     KeyPairGenerator g = KeyPairGenerator.getInstance("RSA", "BC");
6     g.initialize(1024); KeyPair kp = g.generateKeyPair();
7     String[] rsa = {
8       "RSA","RSA/ECB/NoPadding","RSA/None/NoPadding", //deterministico
9       "RSA/None/PKCS1Padding","RSA/None/OAEPWithSHA1AndMGF1Padding"};
10    for (int a = 0; a < rsa.length; a++) {
11      Cipher enc = Cipher.getInstance(rsa[a], "BC");
12      enc.init(Cipher.ENCRYPT_MODE, kp.getPublic());
13      Cipher dec = Cipher.getInstance(rsa[a], "BC");
14      dec.init(Cipher.DECRYPT_MODE, kp.getPrivate());
15
16      U.println("Encriptado com: " + enc.getAlgorithm());
17      byte[][] criptograma = new byte[2][];
18      for (int i = 0; i < 2; i++) {
19        criptograma[i] = enc.doFinal(textoClaroAna);
20        byte[] textoClaroBeto = dec.doFinal(criptograma[i]);
21        U.println("Criptograma    : " + U.b2x(criptograma[i]));
22    }}
23  } catch (NoSuchAlgorithmException|NoSuchPaddingException|
24        InvalidKeyException|IllegalBlockSizeException|
25        BadPaddingException|NoSuchProviderException e)
26  { System.out.println(e); }}
```

2.4.2. Problemas do RSA no padrão PKCS#1

Esta subseção aborda em maior detalhe os problemas da versão 1 do padrão PKCS#1 a partir de três exemplos. Primeiro, a ausência de *padding* que leva à encritação com o RSA canônico. Em seguida, o uso de assinaturas digitais determinísticas com o PKCS#1. Em terceiro, encriptação com o *padding* PKCS#1v1.5, considerado inseguro, apesar de ser aleatorizado.

2.4.2.1. Ausência de *padding* na encriptação com RSA

O trecho de código na listagem 2.9 mostra mais uma vez o uso do RSA canônico para encriptação. A utilização do algoritmo RSA sem *padding* aleatorizado é considerada um mau uso porque pode levar a revelação indevida de informação pela identificação de padrões de texto claro no criptograma. Isto é, quando um mesmo texto claro é encriptado mais de uma vez com mesma chave. A identificação de tais padrões pode, por exemplo, favorecer a análise de tráfego por um atacante capaz de observar a ocorrência de padrões na comunicação.

Há outros ataques sobre o RSA canônico que são mencionados pela literatura especializada [Boneh 1999], dentre eles há dois ataques considerados simples. O ataque de *broadcast* acontece quando a mesma mensagem m é encriptada para e_i destinatários diferentes, onde e_i é o valor de alguma chave privada i. Cada destinatário tem seu próprio par de chaves. Nesta caso, é possível decriptar a mensagem sem precisar de qualquer chave privada d. A solução para este ataque, além de um *padding* seguro, é usar um sistema criptográfico híbrido com chaves assimétricas para transporte de chaves simétricas.

O ataque da e-nézima raiz de c acontece quando a mensagem m é muito pequena e o valor de e é muito baixo (por exemplo, no ataque da raiz cúbica com $e = 3$). Neste caso, o criptograma $c = m^e$ é menor que o módulo n e a decriptação pode ser obtida simplesmente pela computação da e-nézima raiz de c.

Listagem 2.9. RSA canônico sem *padding*.

```
1 public static void main(String args[]) {
2   try {
3     Security.addProvider(new BouncyCastleProvider()); // provedor BC
4     byte[] pt = ("Cripto deterministica").getBytes();
5     KeyPairGenerator g = KeyPairGenerator.getInstance("RSA","BC");
6     g.initialize(2048); KeyPair kp = g.generateKeyPair();
7
8     Cipher e = Cipher.getInstance("RSA/None/NoPadding","BC");
9     e.init(Cipher.ENCRYPT_MODE, kp.getPublic());
10    Cipher d = Cipher.getInstance("RSA/None/NoPadding","BC");
11    d.init(Cipher.DECRYPT_MODE, kp.getPrivate());
12    U.println("Texto claro: " + U.b2x(pt));
13    U.println("Encriptado com: " + e.getAlgorithm());
14    U.println("Criptograma: " + U.b2x(e.doFinal(pt)));
15    U.println("Criptograma: " + U.b2x(e.doFinal(pt)));
16    U.println("Criptograma: " + U.b2x(e.doFinal(pt)));
17    U.println("Texto claro: " + U.b2x(d.doFinal(e.doFinal(pt))));
18  } catch (NoSuchAlgorithmException|NoSuchPaddingException|
19           InvalidKeyException|IllegalBlockSizeException |
20           BadPaddingException|NoSuchProviderException e)
21  { System.out.println(e);}
```

2.4.2.2. Assinatura digital RSA determinística com PKCS#1v1

O trecho de código na listagem 2.10 mostra a assinatura digital determinística com RSA sobre um *hash* (no exemplo, SHA-512). O programa exemplo utiliza dois provedores criptográficos diferentes, o "SunRSASign"para assinatura e o "BC"para verificação. O contrário também funcionaria, porque os provedores são intercambiáveis e interoperáveis. Este foi o formato inicialmente usado no padrão PKCS#1 para assinaturas digitais e tinha a finalidade de evitar o uso do RSA canônico que é susceptível ao ataque de falsificação de assinaturas conhecido como *Blinding* [Boneh 1999].

Na listagem 2.10, as linhas de 2 a 6 faze o seguinte. A linha 2 adiciona dinamicamente o provedor criptográfico *BouncyCastle* (BC). A linha 3 instancia um gerador de par de chaves RSA do provedor BC. A linha 4 configura o gerador para chaves com 2048 bits de tamanho e um PRNG seguro, enquanto a linha 5 cria um par de chaves nas configurações indicadas.

As linhas de 7 a 17 realizam o processo de geração da assinatura digital. A linha 7 cria uma instancia do objeto assinador "SHA512withRSA"do provedor "SunRsaSign", enquanto a linha 8 declara o texto claro usado no exemplo. A linha 10 configura o assinador com a chave pública e uma instância de um PRNG seguro. As linhas de 10 a 16 configuram o objeto assinador 3 vezes com a mesma mensagem e PRNGs diferentes. Mesmo assim, as três assinaturas são idênticas. Vale observar que o formato "SHA512withRSA"não usa o PRNG, apesar da API permitir um objeto deste tipo. Esta é uma inconsistência da API.

As linhas de 19 a 22 realizam o processo de verificação da assinatura. A linha 19 cria um objeto assinatura "SHA1withRSA"do provedor "BC", enquanto a linha 21 configura este objeto para o modo de verificação com a chave pública e alimenta o verificador com a mensagem que foi assinada. Finalmente, a linha 22 faz a verificação da assinatura digital.

Listagem 2.10. Assinatura digital RSA determinística.

```
1 public static void main(String[] args) throws Exception {
2   Security.addProvider(new BouncyCastleProvider()); // provedor BC
3   KeyPairGenerator kg = KeyPairGenerator.getInstance("RSA", "BC");
4   kg.initialize(2048, new SecureRandom());
5   KeyPair kp = kg.generateKeyPair();
6
7   Signature sig = Signature.getInstance("SHA512withRSA","SunRsaSign");
8   byte[] m = "Testing RSA Signature PKCS1".getBytes();
9
10  sig.initSign(kp.getPrivate(), SecureRandom.getInstanceStrong());
11  sig.update(m); U.println("Signature: "+ U.b2x(sig.sign()));
12  sig.initSign(kp.getPrivate(), SecureRandom.getInstanceStrong());
13  sig.update(m); U.println("Signature: "+ U.b2x(sig.sign()));
14  sig.initSign(kp.getPrivate(), SecureRandom.getInstanceStrong());
15  sig.update(m); byte[] s = sig.sign();// generate a signature
16  sig.update(m); U.println("Signature: "+ U.b2x(s));
17
18  // verify a signature
19  Signature verifier = Signature.getInstance("SHA512withRSA","BC");
20  System.out.println("Algorithm: "+ sig.getAlgorithm());
21  verifier.initVerify(kp.getPublic()); verifier.update(m);
22  if (verifier.verify(s)) { U.println("Verification succeeded.");}
23  else                    { U.println("Verification failed.");    }}
```

2.4.2.3. Encriptação com RSA PKCS#1 v1.5

O trecho de código na Listagem 2.11 descreve a encriptação RSA com o preenchimento (ou *padding*) conforme o padrão PKCS#1 v1.5. Este padrão de preenchimento é, em certos casos, considerado inseguro [Bleichenbacher 1998], devido a sua susceptibilidade ao ataque de *padding oracle*, mas que ainda é bastante utilizado no SSL/TLS até a versão v1.2. Na listagem 2.11, as linhas 3 a 6 definem o provedor criptográfico e o texto claro usados no exemplo e criam o par de chaves com tamanho de 2048 bits.

O *padding* é feito conforme a ilustração a seguir, onde "02"são 16 bits indicando que o criptograma possui um *padding* PKCS#1, *nonce* é um valor pseudoaleatório de uso único e "00"é um separador de 16 bits entre o texto claro original e o *nonce*.

02	*nonce*	00	texto claro

Apesar da vulnerabilidade do PKCS#1v1.5 ao ataque de *padding oracle* ter sido publicada já há vinte anos, em 1998 [Bleichenbacher 1998], ainda hoje é possível achar implementações do PKCS#1v1.5 em sistemas criptográficos em produção e padrões em uso, como o SSL/TLS v1.2. Este fato levanta a questão de como é difícil seguir as boas práticas.

Ainda na Listagem 2.11, as linhas de 8 a 11 criam duas máquinas criptográficas para o RSA com *padding* PKCS#1v1.5, indicado pela *string* "RSA/None/PKCS1Padding": Uma para encriptação (configurada com a chave pública) e outra de decriptação (usando a chave privada). Finalmente, as linhas de 14 a 16 realizam três encriptações sucessivas do mesmo texto claro para mostrar que elas são diferentes de fato. A linha 17 faz a decriptação. As linhas de 18 a 21 mostram todas as exceções tratadas para o RSA PKCS#1v1.5 em Java.

Listagem 2.11. RSA com PKCS#1.

```
1 public static void main(String args[]) {
2   try {
3     Security.addProvider(new BouncyCastleProvider());// provedor BC
4     byte[] ptAna = ("Randomized RSA").getBytes();
5     KeyPairGenerator g = KeyPairGenerator.getInstance("RSA","BC");
6     g.initialize(2048); KeyPair kp = g.generateKeyPair();
7
8     Cipher e = Cipher.getInstance("RSA/None/PKCS1Padding","BC");
9     e.init(Cipher.ENCRYPT_MODE, kp.getPublic());
10    Cipher d = Cipher.getInstance("RSA/None/PKCS1Padding","BC");
11    d.init(Cipher.DECRYPT_MODE, kp.getPrivate());
12
13    U.println("Plaintext : " + new String(ptAna));
14    U.println("Ciphertext: " + U.b2x(e.doFinal(ptAna)));
15    U.println("Ciphertext: " + U.b2x(e.doFinal(ptAna)));
16    U.println("Ciphertext: " + U.b2x(e.doFinal(ptAna)));
17    U.println("Plaintext : " + U.b2x(d.doFinal(e.doFinal(ptAna))));
18  } catch (NoSuchAlgorithmException | NoSuchPaddingException |
19          InvalidKeyException | IllegalBlockSizeException |
20          BadPaddingException | NoSuchProviderException e)
21  { System.out.println(e); }}
```

2.4.3. Configurações fracas do RSA-OAEP

A Figura 2.8 mostra as combinações possíveis para funções de *hash* e tamanho de chaves (módulo) do RSA-OAEP, até o valor de 15360 de módulo (nível de segurança de 256 bits). Cada célula da figura tem a quantidade de bits processada pelo RSA-OAEP. As combinações tachadas têm níveis de segurança menores que 112 bits. As combinações sublinhadas têm nível de segurança de 112 bits. As combinações em negrito com módulos de 3072, 7680 e 15360 têm 128, 192 e 256 bits de segurança, respectivamente.

Tamanho da chave RSA OAEP	Tamanho do hash				Tamanho máximo do texto claro
Módulo	160	256	384	512	
256					
384	~~48~~				
512	~~176~~				
768	~~432~~	~~240~~			
1024	~~688~~	~~496~~	~~240~~		
2048	~~1712~~	*1520*	*1264*	*1008*	
3072	~~2736~~	**2544**	**2288**	**2032**	
4096	~~3760~~	**3568**	**3312**	**3056**	
7680	~~7344~~	*7152*	**6896**	**6640**	
15360	~~15024~~	*14832*	*14576*	**14320**	
	Tamanho máximo do texto claro				

Figura 2.8. Combinações de tamanho de chave (módulo) e *hash* para o RSA-OAEP.

O trecho de código na Listagem 2.12 mostra um exemplo de configuração insegura do RSA-OAEP em que as chaves tem tamanho de 1024 bits (linha 5) e o *hash* é de 160 bits, conforme a *string* "RSA/None/OAEPwithSHA1andMGF1Padding"(linha 7). Este código processa apenas 86 bytes de cada vez (linha 10).

Listagem 2.12. RSA-OAEP com chave de 1024 bits e SHA-1.

```
1 public static void main(String args[]) {
2  try {
3    Security.addProvider(new BouncyCastleProvider());
4    KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA","BC");
5    kpg.initialize(1024); KeyPair kp = kpg.generateKeyPair();
6
7    Cipher c=Cipher.getInstance("RSA/None/OAEPwithSHA1andMGF1Padding");
8
9    c.init(Cipher.ENCRYPT_MODE, kp.getPublic());
10   byte[] ptAna = U.cancaoDoExilio.substring(0,86).getBytes();
11   byte[] ct = c.doFinal(ptAna);
12
13   c.init(Cipher.DECRYPT_MODE, kp.getPrivate());
14   byte[] ptBeto = c.doFinal(ct); // Decriptando
15 } catch (NoSuchAlgorithmException | NoSuchPaddingException |
16          InvalidKeyException      | BadPaddingException     |
17          IllegalBlockSizeException | NoSuchProviderException e)
18 {System.out.println(e);}}
```

2.4.4. Assinaturas digitais inseguras

Esta subseção trata vulnerabilidades associadas às assinaturas digitais fracas. A primeira é a repetição do *nonce (number used once)* na geração de assinaturas ECDSA e a segunda é a utilização de configurações dos esquemas de assinatura ECDSA, DSA e RSA.

2.4.4.1. *Nonce* repetido na assinatura ECDSA

O ECDSA pode ser mal utilizado e produzir assinaturas digitais vulneráveis. No ECDSA, cada assinatura requer um número aleatório único e imprevisível, que deve ser utilizado apenas uma vez (*nonce*). Por isto, o ECDSA é vulnerável à geração de números pseudoaleatórios ruins. Aplicações de assinaturas digitais ECDSA são muito sensíveis aos maus usos do ECDSA, tais como sementes de PRNGs com entropia baixa e *nonces* repetidos (com valores fixos em programas). Assinaturas digitais geradas com o mesmo *nonce*, ou *nonces* parecidos (com alguns bits em comum) podem revelar a chave privada em aplicações reais como *eWallets* de *bitcoins* [Bos et al. 2014, Braga et al. 2017b].

Na Listagem 2.13, as linhas 6 e 7 instanciam um objeto `SecureRandom` do tipo "SHA1PRNG"e o configuram com uma semente de 24 bytes. As linhas de 9 a 12 realizam os passos para geração da assinatura digital. A linha 9 instancia um assinador para o "SHA256withECDSA". A linha 10 configura o assinador com a chave privada e o PRNG. A linha 12 calcula a assinatura digital sobre o documento. As linhas de 14 a 20 repetem os passos de instanciação e configuração de um assinador e geração da assinatura digital para o mesmo documento assinado anteriormente, mas, desta vez, reutilizando a mesma semente para o "SHA1PRNG". Por isso, as duas assinaturas digitais comparadas são iguais (linha 22).

Listagem 2.13. Nonce repetido com ECDSA.

```
1 public static void main(String[] args) throws Exception {
2   KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC","SunEC");
3   kpg.initialize(256, SecureRandom.getInstanceStrong());
4   KeyPair kpAna = kpg.generateKeyPair();
5
6   SecureRandom sr1 = SecureRandom.getInstance("SHA1PRNG","SUN");
7   byte[] seed = sr1.generateSeed(24); sr1.setSeed(seed);
8
9   Signature signer1 = Signature.getInstance("SHA256withECDSA","SunEC");
10  signer1.initSign(kpAna.getPrivate(), sr1);
11  byte[] doc = U.cancaoDoExilio.getBytes();
12  signer1.update(doc); byte[] sign1 = signer1.sign();
13
14  SecureRandom sr2 = SecureRandom.getInstance("SHA1PRNG","SUN");
15  sr2.setSeed(seed);
16
17  Signature signer2 = Signature.getInstance("SHA256withECDSA","SunEC");
18  signer2.initSign(kpAna.getPrivate(), sr2);
19  doc = U.cancaoDoExilio.getBytes();
20  signer2.update(doc); byte[] sign2 = signer2.sign();
21
22  boolean ok = Arrays.areEqual(sign1, sign2);
23  if (ok) {U.println("Nonce repeated! Signatures are equal!");}}
```

2.4.4.2. Configurações fracas de assinatura ECDSA

A Listagem 2.14 ilustra como uma curva elíptica insegura pode ser utilizada de modo implícito pelas configurações da assinatura digital ECDSA. No programa exemplo, a linha 3 mostra a criação de um par de chaves de 112 bits para curvas elípticas. Na linha 4, um objeto assinador é criado para "SHA1withECDSA". As curvas de 112 bits combinadas com SHA-1 (160 bits) resultam em segurança de 80 bits apenas. As linhas de 7 a 10 realizam os passos para geração da assinatura digital: configuram o assinador com a chave privada, parametrizam o assinador com o documento a ser assinado e calculam a assinatura digital do documento. Já as linhas de 11 a 13 realizam os passos para a verificação da assinatura digital com a chave pública.

Listagem 2.14. ECDSA com chave de 112 bits mas segurança de 80 bits.

```
1 public static void main(String[] args) throws Exception {
2    KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC","SunEC");
3    kpg.initialize(112, SecureRandom.getInstanceStrong());
4    Signature signer = Signature.getInstance("SHA1withECDSA","SunEC");
5    KeyPair kp = kpg.generateKeyPair();
6
7    signer.initSign(kp.getPrivate(), new SecureRandom());
8    byte[] doc = U.cancaoDoExilio.getBytes();
9    signer.update(doc); byte[] sign = signer.sign();
10
11   Signature verifier = Signature.getInstance("SHA1withECDSA","SunEC");
12   verifier.initVerify(kp.getPublic()); verifier.update(doc);
13   if(!verifier.verify(sign)){System.out.println("Signature not OK!");}
```

2.4.4.3. Configurações fracas de assinatura RSA

O trecho de código da Listagem 2.15 mostra na linha 4 três configurações fracas de assinaturas digitais RSA com *hashes*: MD2withRSA, MD5withRSA e SHA1withRSA. As função de *hash* MD2 é obsoleta, o MD5 já foi quebrado e o SHA-1 foi descontinuado. Por isto, assinaturas sobre estes *hashes* são inseguras. Chaves de 1024 bits adicionam outra vulnerabilidade ao código.

Listagem 2.15. Três configurações inseguras de assinaturas digitais RSA.

```
1 public static void main(String[] args) throws Exception {
2   KeyPairGenerator
      kpg=KeyPairGenerator.getInstance("RSA","SunRsaSign");
3   kpg.initialize(1024,SecureRandom.getInstanceStrong());
4   String[] weakrsa = {"MD2withRSA", "MD5withRSA", "SHA1withRSA"};
5   byte[] doc = U.cancaoDoExilio.getBytes();
6   for (String rsa : weakrsa) {
7     Signature signer = Signature.getInstance(rsa, "SunRsaSign");
8     KeyPair kp1 = kpg.generateKeyPair();
9     signer.initSign(kp1.getPrivate(), new SecureRandom());
10    signer.update(doc);byte[] sign = signer.sign();
11
12    Signature verifier = Signature.getInstance(rsa, "SunRsaSign");
13    verifier.initVerify(kp1.getPublic()); verifier.update(doc);
14    if (verifier.verify(sign)){U.println("Signature OK!");}}}
```

2.4.4.4. Configurações fracas de assinatura DSA

O trecho de código da Listagem 2.16 mostra duas configurações inseguras de assinaturas digitais DSA: "SHA1withDSA"e "NONEwithDSA". No primeiro caso, a assinatura tem o nível de segurança do elemento mais fraco, o SHA-1, com segurança menor que 80 bits, apesar das chaves de 2048 bits. No segundo caso, o "NONEwithDSA"permite que um *hash* seja calculado externamente ao assinador. Porém, a API só permite que o SHA-1 seja usado para cálculo do *hash* devido a uma restrição ao tamanho dos dados passados como parâmetros para o assinador, que devem ser de exatamente 20 bytes (128 bits), o tamanho da saída do SHA-1.

Ainda na Listagem 2.16, as linhas de 2 a 4 criam um par de chaves DSA de 2048 bits. As linhas 6 e 7 identificam as esquemas de assinatura digital usados no exemplo e o texto claro, o documento, a ser assinado. O laço das linhas 10 até 23 realizam os passos de geração e verificação de assinatura para os dois esquemas do exemplo: "SHA1withDSA"e "NONEwithDSA".

Primeiro, nas linhas de 10 a 15 criam o objeto assinador DSA, que é configurado com a chave privada e um PRNG. Para o caso da assinatura sobre *hash* externo, uma instância do `MessageDigest` SHA-1 é criada e usada para calcular o *hash* do documento. Na linha 16, o assinador recebe o documento a ser assinado ou o *hash* do documento e gera a assinatura digital.

Em seguida, nas linhas de 18 a 20, um verificador de assinaturas para o DSA é criado, configurado com a chave pública e alimentado com o documento (ou o *hash* do documento, no caso do *hash* externo), que foi assinado. Enquanto a linha 21 contém a verificação da assinatura propriamente, com o método `verify()` do verificador.

Listagem 2.16. Duas configurações inseguras de assinaturas digitais DSA.

```
1 public static void main(String[] args) throws Exception {
2   KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA", "SUN");
3   kpg.initialize(2048, SecureRandom.getInstanceStrong());
4   KeyPair kp = kpg.generateKeyPair();
5
6   String[] weakdsa = {"SHA1withDSA","NONEwithDSA"};
7   byte[] doc = U.cancaoDoExilio.getBytes();
8
9   for (String dsa : weakdsa) {
10    Signature signer = Signature.getInstance(dsa, "SUN");
11    signer.initSign(kp.getPrivate(), new SecureRandom());
12    if (dsa.equals("NONEwithDSA")){
13      MessageDigest md = MessageDigest.getInstance("SHA1");
14      doc = md.digest(doc);//Data must be exactly 20 bytes
15    }
16    signer.update(doc); byte[] sign = signer.sign();
17
18    Signature verifier = Signature.getInstance(dsa, "SUN");
19    verifier.initVerify(kp.getPublic()); verifier.update(doc);
20
21    if (verifier.verify(sign)){U.println("Signature OK!");}
22    else                       {U.println("Signature not OK!");}
23    }}
```

2.4.5. Acordo de chaves sem autenticação

Esta seção contempla os maus usos criptográficos associados aos protocolos de acordo de chaves baseados em Diffie-Hellman (DH) tradicional e sobre curvas elípticas (ECDH). A seção mostra quatro programas exemplo bastante parecidos entre si, mas com variações sutis que indicam o mau uso: DH sem autenticação com parâmetros estáticos ou dinâmicos e ECDH sem autenticação com parâmetros estáticos ou dinâmicos.

Conforme mencionado anteriormente na seção 2.2, tanto o DH quanto o ECDH não possuem autenticação intrínseca das trocas de mensagens entre os participantes do protocolo. A capacidade de autenticação das mensagens é obtida externamente pela combinação do protocolo com um esquema de assinaturas digitais. Os provedores criptográficos comumente usados em Java não possuem implementações do acordo de chaves autenticado, seja para DH ou ECDH. No SSL/TLS, o DH/ECDH sem autenticação é chamado de DH/ECDH anônimo, passando uma falsa sensação de segurança por anonimato para os usuários deste protocolo. Atualmente, o DH/ECDH anônimo do SSL/TLS é considerado inseguro.

2.4.5.1. DH estático não autenticado

Chama-se de DH estático à instância do DH clássico em que os parâmetros do protocolo compartilhados entre os participantes (o gerador g e o primo p) são definidos uma vez (possivelmente *a priori*) e reutilizados em várias execuções do protocolo. O trecho de código da Listagem 2.17 mostra uma implementação didática do DH não autenticado e com parâmetros estáticos, em que as duas partes comunicantes (Alice e Bob) são implementadas no mesmo código. A Listagem 2.17 contém o primeiro exemplo a explorar o acordo de chaves em Java e por isto será explicado em maior detalhe que os outros programas semelhantes.

As linhas de 3 a 8 são responsáveis por criar os parâmetros estáticos. Um gerador de parâmetros DH do provedor SunEC é instanciado na linha 4 e inicializado (linha 5) para chaves com 1024 bits de tamanho, enquanto as linhas 6 contém a geração dos parâmetros propriamente. Já as linhas 7 e 8 contém a formação dos parâmetros de acordo com a especificação da API.

As linha de 10 a 15 realizam os passos de criação de uma par de chaves DH para Alice e inicialização do protocolo DH. A linha 11 instancia um gerador e par de chaves para o DH. Na linha 12, o gerador de chaves é configurado com os parâmetros calculados anteriormente e um par de chaves é gerado de acordo. A linha 13 instancia um acordo de chaves DH, que é configurado com a chave privada da Alice (linha 14). A chave pública da Alice é codificada na linha 15, para que seja enviada para Bob.

As linhas de 17 a 24 realizam os passos do protocolo do ponto de vista do Bob. A linha 18 cria uma fábrica de chaves DH que recebe a chave pública da Alice (que foi codificada no passo anterior e transportada até Bob) e a converte (linha 19) em um instancia válida de chave pública da Alice que é mantida por Bob (linha 20). As linhas de 22 a 24 criam um par de chaves DH para Bob (linha 24) com base nos parâmetros compartilhados com a Alice, obtidos da chave pública da Alice (linha 22).

A instância do acordo de chaves, do ponto de vista de Bob, é criada na linha 26 e configurada com a chave privada de Bob na linha 27. Já a linha 28 codifica a chave pública de Bob para que seja enviada para Alice.

As linhas de 30 a 35 voltam para o ponto de vista da Alice. A linha 31 cria uma fábrica de chaves DH que recebe a chave pública da Bob (que foi codificada no passo anterior e transportada até Alice) e a converte (linha 32) em um instancia válida de chave pública da Alice que é mantida por Bob (linha 33). Já a linha 34 realizada o segundo passo do DH sobre a chave pública de Bob e calcula o segredo da Alice.

A linha 37 volta o protocolo para o ponto de vista de Bob, realizando o segundo passo do DH sobre a chave pública de Alice e calculando o segredo de Bob. Neste momento, Alice e Bob concluíram o protocolo e chegaram ao mesmo segredo. Na prática uma mensagem autenticada por um HMAC serve de evidência entre as partes que ambas compartilham um segredo.

Listagem 2.17. DH não autenticado e com parâmetros estáticos.

```
1 public static void main(String argv[]) {
2  try {
3    AlgorithmParameterGenerator apg
4        = AlgorithmParameterGenerator.getInstance("DH","SunJCE");
5    apg.init(1024); // here is the size
6    AlgorithmParameters p = apg.generateParameters();
7    DHParameterSpec dhps
8        = (DHParameterSpec) p.getParameterSpec(DHParameterSpec.class);
9
10   //Alice starts DH by creating a key pair
11   KeyPairGenerator aKPG = KeyPairGenerator.getInstance("DH","SunJCE");
12   aKPG.initialize(dhps); KeyPair aKP = aKPG.generateKeyPair();
13   KeyAgreement aKA = KeyAgreement.getInstance("DH","SunJCE");
14   aKA.init(aKP.getPrivate());
15   byte[] aPubKe = aKP.getPublic().getEncoded();
16
17   // Let's turn over to Bob.
18   KeyFactory bKF = KeyFactory.getInstance("DH","SunJCE");
19   X509EncodedKeySpec x509ks = new X509EncodedKeySpec(aPubKe);
20   PublicKey aPubK = bKF.generatePublic(x509ks);
21
22   DHParameterSpec dhps2 = ((DHPublicKey) aPubK).getParams();
23   KeyPairGenerator bKPG = KeyPairGenerator.getInstance("DH","SunJCE");
24   bKPG.initialize(dhps2); KeyPair bKP = bKPG.generateKeyPair();
25
26   KeyAgreement bKA = KeyAgreement.getInstance("DH","SunJCE");
27   bKA.init(bKP.getPrivate());
28   byte[] bPubKe = bKP.getPublic().getEncoded();
29
30   // Alice uses Bob's public key
31   KeyFactory aKF = KeyFactory.getInstance("DH","SunJCE");
32   x509ks = new X509EncodedKeySpec(bPubKe);
33   PublicKey bPubK = aKF.generatePublic(x509ks);
34   aKA.doPhase(bPubK, true); byte[] aSecret = aKA.generateSecret();
35
36   // Bob uses Alice's public key
37   bKA.doPhase(aPubK, true); byte[] bSecret = bKA.generateSecret();
38
39   //Alice and Bob completed DH key agreement protocol.
40   if (!Arrays.equals(aSecret,bSecret)) // not this way in real life
41        {throw new Exception("Secrets differ");}
42 }catch(Exception e){System.err.println("Error:"+e); System.exit(1);}}
```

2.4.5.2. DH efêmero não autenticado

Chama-se de DH efêmero à instância do DH clássico em que os parâmetros compartilhados (o gerador *g* e o primo *p*) são redefinidos para cada execução do protocolo e sempre descartados ao final da execução, sem reuso dos parâmetros. O trecho de código da Listagem 2.18 mostra uma implementação didática do DH não autenticado e com parâmetros descartáveis.

Em linhas gerais, a Listagem 2.18 é bastante parecida com aquela mostrada anteriormente, mas com uma diference importante: os parâmetros compartilhados (*g* e *p*) não são gerado explicitamente, mas sim criados implicitamente na geração do par de chaves de Alice. As linha de 3 a 7 realizam os passos de criação de uma par de chaves DH para Alice e a inicialização do protocolo DH. A linha 3 instancia um gerador e par de chaves para o DH. Na linha 4, o gerador de chaves é configurado para chaves com 1024 bits de tamanho (fracas) e um par de chaves é gerado de acordo (linha 5). A linha 6 instancia um acordo de chaves DH, que é configurado com a chave privada da Alice (linha 7). Os passos análogos para Bob estão nas linhas de 15 a 19.

Listagem 2.18. DH não autenticado e com parâmetros descartáveis ou efêmeros.

```
1 public static void main(String argv[]) {
2  try {
3    KeyPairGenerator aKPG=KeyPairGenerator.getInstance("DH","SunJCE");
4    aKPG.initialize(1024);
5    KeyPair aKP = aKPG.generateKeyPair();
6    KeyAgreement aKA = KeyAgreement.getInstance("DH", "SunJCE");
7    aKA.init(aKP.getPrivate());
8
9    byte[] aPubK = aKP.getPublic().getEncoded();
10
11   KeyFactory bKF = KeyFactory.getInstance("DH", "SunJCE");
12   X509EncodedKeySpec x509ks = new X509EncodedKeySpec(aPubK);
13   PublicKey apk = bKF.generatePublic(x509ks);
14
15   KeyPairGenerator bKPG=KeyPairGenerator.getInstance("DH","SunJCE");
16   bKPG.initialize(1024);
17   KeyPair bKP = bKPG.generateKeyPair();
18   KeyAgreement bKA = KeyAgreement.getInstance("DH", "SunJCE");
19   bKA.init(bKP.getPrivate());
20
21   byte[] bPubK = bKP.getPublic().getEncoded();
22
23   KeyFactory aKF = KeyFactory.getInstance("DH", "SunJCE");
24   x509ks = new X509EncodedKeySpec(bPubK);
25   PublicKey bobPubKey = aKF.generatePublic(x509ks);
26   aKA.doPhase(bobPubKey, true);
27   byte[] aSecret = aKA.generateSecret();
28
29   bKA.doPhase(apk, true);
30   byte[] bSecret = bKA.generateSecret();
31
32   if(!Arrays.equals(aSecret,bSecret)) // not this way in real life
33        throw new Exception("Secrets differ");
34 }catch(Exception e){System.err.println("Error: "+e);System.exit(1);}}
```

2.4.5.3. ECDH efêmero não autenticado

O trecho de código da Listagem 2.19 é análogo ao anterior e mostra uma implementação didática do ECDH não autenticado e com parâmetros descartáveis. Em Java, a única diferença relevante entre o ECDH e o DH tradicional está na geração do par de chaves. As linha de 3 a 7 realizam os passos de criação de uma par de chaves ECDH para Alice. A linha 3 instancia um gerador e um par de chaves para o ECDH. Na linha 4, o gerador de chaves é configurado para chaves com 224 bits de tamanho (chaves com apenas 112 bits de segurança) e um par de chaves é gerado de acordo (linha 5).

Os passos análogos, do ponto de vista de Bob, acontecem nas linhas de 15 e 16. Vale mencionar uma vantagem da criptografia de curvas elípticas, as chaves ECDH são bem menores que as chaves correspondentes (no mesmo nível de segurança) do DH clássico. Em particular, a chave de 224 bits usada neste exemplo do ECDH correspondem a uma chave de 2048 bits no DH.

Listagem 2.19. ECDH não autenticado e com parâmetros efêmeros.

```
1 public static void main(String argv[]) {
2  try {
3    KeyPairGenerator aKPG = KeyPairGenerator.getInstance("EC", "SunEC");
4    aKPG.initialize(224); // only 112 bits of security
5    KeyPair aKP = aKPG.generateKeyPair();
6
7    KeyAgreement aKA = KeyAgreement.getInstance("ECDH","SunEC");
8    aKA.init(aKP.getPrivate());
9    byte[] aPubKe = aKP.getPublic().getEncoded();// this goes to Bob
10
11   KeyFactory bKF = KeyFactory.getInstance("EC", "SunEC");
12   X509EncodedKeySpec x509ks = new X509EncodedKeySpec(aPubKe);
13   PublicKey apk = bKF.generatePublic(x509ks);
14
15   KeyPairGenerator bKPG = KeyPairGenerator.getInstance("EC","SunEC");
16   bKPG.initialize(224); KeyPair bKP = bKPG.generateKeyPair();
17
18   KeyAgreement bKA = KeyAgreement.getInstance("ECDH","SunEC");
19   bKA.init(bKP.getPrivate());
20
21   byte[] bPubKe = bKP.getPublic().getEncoded();//this goes to Alice
22
23   KeyFactory aKF = KeyFactory.getInstance("EC","SunEC");
24   x509ks = new X509EncodedKeySpec(bPubKe);
25   PublicKey bPubK = aKF.generatePublic(x509ks);
26   aKA.doPhase(bPubK, true);
27   byte[] aSecret = aKA.generateSecret();
28
29   bKA.doPhase(apk, true);
30   byte[] bSecret = bKA.generateSecret();
31
32   if(!Arrays.equals(aSecret,bSecret)) // not this way in real life
33       throw new Exception("Secrets differ");
34 }catch(Exception e){System.err.println("Error: "+e);System.exit(1);}}
```

2.4.6. Acordo de chaves com chaves assimétricas fracas

A utilização de chaves assimétricas fracas é um mau uso de criptografia comum em sistemas criptográficos de chave pública e, em relação aos protocolos de acordo de chaves, pode ocorrer tanto no DH quanto no ECDH, comprometendo a segurança do segredo compartilhado, que foi gerado por uma rodada do protocolo com chaves assimétricas inseguras.

2.4.6.1. Chaves assimétricas fracas com DH

O trecho de código da Listagem 2.20 é análogo aos exemplos anteriores e ilustra o uso de chaves assimétricas fracas no DH. Na linha 4, um gerador de par de chaves é configurado para gerar chaves DH de 512 bits e um par de chaves é gerado para Alice com este tamanho, na linha 5. O par de chaves de Bob é gerado nas linhas de 16 a 18, também com tamanho de 512 bits. Atualmente, qualquer tamanho de chaves DH menor que 2048 já é considerado inseguro.

Listagem 2.20. DH não autenticado e chaves fracas.

```
1 public static void main(String argv[]) {
2  try {
3    KeyPairGenerator aKPG = KeyPairGenerator.getInstance("DH","SunJCE");
4    aKPG.initialize(512);
5    KeyPair aKP = aKPG.generateKeyPair();
6
7    KeyAgreement aKA = KeyAgreement.getInstance("DH", "SunJCE");
8    aKA.init(aKP.getPrivate());
9
10   byte[] aPubKe = aKP.getPublic().getEncoded();
11
12   KeyFactory bKF = KeyFactory.getInstance("DH", "SunJCE");
13   X509EncodedKeySpec x509ks = new X509EncodedKeySpec(aPubKe);
14   PublicKey aPubK = bKF.generatePublic(x509ks);
15
16   KeyPairGenerator bKPG = KeyPairGenerator.getInstance("DH","SunJCE");
17   bKPG.initialize(512);
18   KeyPair bKP = bKPG.generateKeyPair();
19
20   KeyAgreement bKA = KeyAgreement.getInstance("DH","SunJCE");
21   bKA.init(bKP.getPrivate());
22
23   byte[] bPubKe = bKP.getPublic().getEncoded();
24
25   KeyFactory aKF = KeyFactory.getInstance("DH","SunJCE");
26   x509ks = new X509EncodedKeySpec(bPubKe);
27   PublicKey bPubK = aKF.generatePublic(x509ks);
28   aKA.doPhase(bPubK, true);
29   byte[] aSecret = aKA.generateSecret();
30
31   bKA.doPhase(aPubK, true);
32   byte[] bSecret = bKA.generateSecret();
33
34   if(!Arrays.equals(aSecret,bSecret)) // not this way in real life
35       throw new Exception("Secrets differ");
36 }catch(Exception e){System.err.println("Error: "+e);System.exit(1);}}
```

2.4.6.2. Chaves assimétricas fracas com ECDH

O trecho de código da Listagem 2.21 é análogo aos exemplos anteriores e ilustra o uso de chaves assimétricas fracas no ECDH. Na linha 4, um gerador de par de chaves é configurado para gerar chaves ECDH de 160 bits de tamanho (80 bits de segurança) e um par de chaves é gerado para Alice, na linha 5. O par de chaves de Bob é gerado nas linhas de 16 a 18, também com tamanho de 160 bits.

Atualmente, qualquer tamanho de chaves ECDH menor que 224 bits (112 bits de segurança) já é considerado inseguro. Além disso, o leitor não deve se enganar com a diferença de tamanho entre as chaves DH e ECDH, uma chave DH de 512 bits é muito mais fraca que uma chave ECDH de 160 bits e ambas são inseguras. A próxima seção aborda curvas elípticas inseguras e os tamanhos de chaves correspondentes.

Listagem 2.21. CEDH não autenticado e com chaves fracas.

```
1 public static void main(String argv[]) {
2  try {
3    KeyPairGenerator aKPG = KeyPairGenerator.getInstance("EC", "SunEC");
4    aKPG.initialize(160); // this has only 80 bits of security
5    KeyPair aKP = aKPG.generateKeyPair();
6
7    KeyAgreement aKA = KeyAgreement.getInstance("ECDH", "SunEC");
8    aKA.init(aKP.getPrivate());
9
10   byte[] aPubKe = aKP.getPublic().getEncoded();
11
12   KeyFactory bKF = KeyFactory.getInstance("EC", "SunEC");
13   X509EncodedKeySpec x509ks = new X509EncodedKeySpec(aPubKe);
14   PublicKey aPubK = bKF.generatePublic(x509ks);
15
16   KeyPairGenerator bKPG = KeyPairGenerator.getInstance("EC", "SunEC");
17   bKPG.initialize(160);
18   KeyPair bKP = bKPG.generateKeyPair();
19   KeyAgreement bKA = KeyAgreement.getInstance("ECDH", "SunEC");
20   bKA.init(bKP.getPrivate());
21
22   byte[] bPubKe = bKP.getPublic().getEncoded();
23
24   KeyFactory aKF = KeyFactory.getInstance("EC", "SunEC");
25   x509ks = new X509EncodedKeySpec(bPubKe);
26   PublicKey bPubK = aKF.generatePublic(x509ks);
27   aKA.doPhase(bPubK, true);
28   byte[] aSecret = aKA.generateSecret();
29
30   bKA.doPhase(aPubK, true);
31   byte[] bSecret = bKA.generateSecret();
32
33   if(!Arrays.equals(aSecret, bSecret)) // not this way in real life
34       throw new Exception("Secrets differ");
35 }catch(Exception e){System.err.println("Error: "+e);System.exit(1);}}
```

2.4.7. Curvas elípticas inseguras

Em Java, o programador pode usar a criptografia de curvas elípticas de forma implícita, indicando apenas o tamanho das chaves assimétricas, como foi mostrado nos exemplos até aqui. Outra opção disponível ao programador é a escolha explícita de uma curva elíptica, conforme mostrado no trecho de código da Listagem 2.22. Porém, cuidados devem ser tomados para evitar a seleção de curvas elípticas fracas (inseguras).

A Tabela 2.3 mostra curvas elípticas que fizeram parte da primeira versão do padrão SEC 2 [SEC 2000], mas que foram excluídas da segunda versão deste mesmo padrão [SEC 2010], por serem consideradas inseguras, com níveis de segurança já bastante baixos para as necessidades atuais. Todas estas curvas ainda podem ser usadas em Java, porque ainda estão disponíveis no provedor SunEC, conforme lustrado na Listagem 2.22.

Tabela 2.3. Curvas elípticas inseguras no SunEC (de [Mart and Hern 2013]).

Segurança	Curvas sobre F_p	Curvas sobre F_{2^m}
56	secp112r1,secp112r2	sect113r1, sect113r2
64	secp128r1,secp128r2	sect131r1, sect131r2
80	secp160k1,secp160r1,secp160r2	–
96	–	sect193r1, sect193r2

O trecho de código da Listagem 2.22 mostra como uma curva elíptica pode ser escolhida *a priori* e servir de insumo para a geração do par de chaves ECDH. As linhas de 2 a 4 listam todas as curvas fracas. O programa funciona em um laço (da linha 6 a linha 17) que cria os pares de chaves para cada uma das curvas inseguras. Nas linhas de 7 a 10, um gerador de par de chaves é configurado com uma curva passada como parâmetro na inicialização. Então, o par de chaves é criado como de costume, na linha 10. As linhas de 11 a 15 exibem diversas informações sobre o par de chaves.

Listagem 2.22. Curvas inseguras do SunEC.

```
1 public static void main(String argv[]) {
2  String[] curves={"secp112r1","secp112r2","secp128r1",
3       "secp128r2","secp160k1","secp160r1","secp160r2","sect113r1",
4       "sect113r2","sect131r1","sect131r2","sect193r1","sect193r2"};
5  try {
6    for (String curve : curves) {
7      ECGenParameterSpec ecps = new ECGenParameterSpec(curve);
8      KeyPairGenerator kpg=KeyPairGenerator.getInstance("EC","SunEC");
9      kpg.initialize(ecps);
10     KeyPair kp = kpg.generateKeyPair();
11     U.println("EC parameters " + ecps.getName());
12     U.println("Pub  key: " + kp.getPublic());
13     U.println("Priv key: " + U.b2x(kp.getPrivate().getEncoded()));
14     U.println("Algorithm: " + kp.getPrivate().getAlgorithm());
15     U.println("Format: " + kp.getPrivate().getFormat());
16     System.out.println();
17   }
18 }catch(NoSuchAlgorithmException|InvalidAlgorithmParameterException|
19         NoSuchProviderException e){System.err.println("Error:"+e);}}
```

2.5. Verificação insegura de certificados digitais

Este seção é organizada em torno das práticas inseguras de programação relacionadas à verificação de certificados digitais e ao uso programático do protocolo SSL/TLS, no lado cliente (isto é, um software cliente SSL/TLS). Os maus usos já foram relacionados aos ataques conhecidos contra o TLS na seção 2.2. Aqui, os maus usos são ilustrados programaticamente por meio de programas em Java. Os problemas de validação de certificados tratados nesta seção são aqueles relacionados à falta de validação dos elementos dos certificados, tais como: data de expiração, cadeia de certificação, nome do emissor e nome do sujeito, além da chave pública. Também são tratadas as armadilhas de programação que inibem a completa validação de certificados digitais e as configurações criptográficas inseguras do cliente SSL/TLS.

2.5.1. Certificado recebido e usado sem qualquer validação

Esta seção inicia uma sequência de exemplos de utilização de SSL/TLS em Java, em que as verificações dos elementos de certificado digital são acrescentadas aos poucos. Neste exemplo em particular, nenhuma validação é feita sobre o certificado recebido e a chave pública contida no certificado é utilizada de boa fé (uma abordagem bastante insegura).

O trecho de código da Listagem 2.23 mostra como a validação manual de um certificado pode ser realizada em Java. Este código ilustra o uso de certificação diretamente pela aplicação, por exemplo, para estabelecer um canal SSL/TLS com outros sistemas que não servidores web com HTTPS. As linhas de 1 a 15 contém o método `validate()`, cuja única função é validar certificados digitais contra as diversas informações passadas como parâmetros para o método, tais como a CA, o emissor, o sujeito/servidor e o período de validade. O método tem um defeito grave, a única verificação (linha 5) está comentada e por isto não é realizada. Caso a linha 5 seja descomentada, então o certificado de cliente terá sua assinatura digital verificada contra a chave pública da autoridade certificadora. Um ponto de atenção é que não há qualquer garantia quanto a autenticidade, ou mesmo integridade, da chave pública da CA.

O método `main()` mostrado nas linhas de 16 a 39 da Listagem 2.23 ilustra o uso do método `validate()` e cria uma cadeia de certificação que será utilizada em todos os exemplos deste seção. Esta cadeia de certificação é composta do certificado raiz (da CA de mais alto nível), certificado intermediário (CA de nível médio) e certificado de usuário final.

Listagem 2.23. Sem qualquer validação dos certificados recebidos.

```
1 public static boolean validate(X509Certificate cert,X509Certificate ca,
2   X500Principal issuer,X500Principal subj,Date date){
3   boolean ok = false;
4   try {
5     //cert.verify(ca.getPublicKey());
6     ok = true;
7   } catch (CertificateException ex) {
8     ok = false; System.out.println(ex);
9   } catch (NoSuchAlgorithmException | InvalidKeyException |
10            NoSuchProviderException | SignatureException ex){
11    ok = false; System.out.println(ex);
12  }
13  return ok;
14 }
15
```

```
16 public static void main(String[] args) {
17   Security.addProvider(new BouncyCastleProvider());
18   try {
19     KeyPair rkp = CertUtils.genRSAKeyPair();//create keys and root cert
20     X509Certificate root = CertUtils.buildSelfSignedCert(rkp);
21
22     // generate intermediate (middle) certificate
23     KeyPair mkp = CertUtils.genRSAKeyPair();
24     X509Certificate middle = CertUtils.buildMiddleCert(mkp.getPublic(),
25       "CN=Middle CA Cert", rkp.getPrivate(), root);
26
27     // generate end entity certificate
28     KeyPair ekp = CertUtils.genRSAKeyPair();
29     X509Certificate user = CertUtils.buildEndCert(ekp.getPublic(),
30       "CN=User Cert",mkp.getPrivate(), middle);
31
32     X500Principal issuer = new X500Principal("CN=Root Certificate");
33     X500Principal subj1  = new X500Principal("CN=Middle CA Cert");
34     X500Principal subj2  = new X500Principal("CN=User Cert");
35
36     if(validate(user, middle, subj1, subj2, null))
37     System.out.println("User Cert successfully validated");
38   } catch (Exception ex) { System.out.println(ex); }}
```

O trecho de código da Listagem 2.24 mostra como estabelecer um canal HTTPS com um servidor web de URL conhecida e utilizando a porta 443. Este exemplo ilustra o caso de um *browser* proprietário. No exemplo, o cliente HTTPS não faz qualquer verificação do certificado digital recebido do servidor. Após o *handshake* (linha 6), algumas informações sobre a conexão são obtidas (linhas 10 e 11) e exibidas (linhas de 11 a 17). Finalmente, se nada de errado ocorreu durante o estabelecimento da conexão, o conteúdo da página é recebido e tratado (linha 23). Este exemplo também serve de base para as verificações nos exemplos seguintes.

Listagem 2.24. Sem validação de certificados SSL/TLS.

```
1 public static void main(String[] args) throws Exception {
2   SSLSocket s = null; Boolean ok = true;
3   try {
4     SSLSocketFactory f=(SSLSocketFactory)SSLSocketFactory.getDefault();
5     s = (SSLSocket) f.createSocket("www.google.com", 443);
6     s.startHandshake();// all validations happen after handshake
7     SSLSession session = s.getSession();
8     Principal peerPrincipal = session.getPeerPrincipal();
9
10    System.out.println("Protocol: " + session.getProtocol());
11    System.out.println("Ciphersuite: " + session.getCipherSuite());
12    System.out.println("Host name: " + session.getPeerHost());
13    System.out.println("\n"+peerPrincipal);
14    System.out.println("\nPeer certificates");
15    Certificate[] peerCertificates = session.getPeerCertificates();
16    for (Certificate c : peerCertificates) { System.out.println(c);}
17  } catch (Exception e) { System.out.println(e); ok = false; }
18
19  if (ok) {CertUtils.handleSocket(s);}
20  else    {System.out.println("Something wrong in cert validation.");}}
```

2.5.2. Acrescentando a verificação do período de validade

O trecho de código da Listagem 2.25 mostra como verificar o período de validade de um certificado digital por meio de duas variações do método `checkValidity()` (linhas 5 e 6), com a data do sistema ou com uma data como parâmetro. Além disso, há duas exceções específicas (linha 8) para os casos de certificado expirado e ainda não válido. Se o certificado estiver dentro do período de validade, a assinatura é verificada (linha 13).

Listagem 2.25. Validação do período de validade.

```
1 public static boolean validate(X509Certificate cert,X509Certificate ca,
2   X500Principal issuer, X500Principal subj, Date date) {
3   boolean ok = false;
4   try {
5     if (date != null) { cert.checkValidity(date);}
6     else              { cert.checkValidity();    }
7     ok = true;
8   }catch(CertificateExpiredException|CertificateNotYetValidException e)
9   { ok = false; System.out.println(e);}
10
11  if (ok) {
12    try { ok = false; cert.verify(ca.getPublicKey()); ok =  true;}
13    catch (CertificateException e)                    { ok = false;}
14    catch (NoSuchAlgorithmException|InvalidKeyException |
15           NoSuchProviderException |SignatureException e){ok = false;}
16  }
17 return ok;}
```

O trecho de código da Listagem 2.26 mostra como um cliente pode verificar o período de validade de um certificado do servidor em uma conexão HTTPS. A linha 11 obtém a lista de certificados enviada pelo servidor. A linha 12 testa se a lista não é nula e tem pelo menos 2 certificados. As linhas 13 e 14 ativam os métodos `checkValidity()` e `verify()` sobre o primeiro certificado da lista e usando a chave pública do segundo certificado.

Listagem 2.26. Validação do período de validade do certificado SSL/TLS.

```
1 public static void main(String[] args) throws Exception {
2   SSLSocket s = null; boolean ok = true;
3   try {
4     SSLSocketFactory f=(SSLSocketFactory)SSLSocketFactory.getDefault();
5     s = (SSLSocket) f.createSocket("www.google.com", 443);
6     s.startHandshake();//all validations happen after handshake
7
8     SSLSession session = s.getSession();
9     Principal peerPrincipal = session.getPeerPrincipal();
10
11    Certificate[] peerCertificates = session.getPeerCertificates();
12    if (peerCertificates != null && peerCertificates.length >= 2) {
13      ((X509Certificate) peerCertificates[0]).checkValidity();
14      peerCertificates[0].verify(peerCertificates[1].getPublicKey());
15    }
16  }catch(CertificateExpiredException|CertificateNotYetValidException|
17      NoSuchAlgorithmException|InvalidKeyException|
18      NoSuchProviderException |SignatureException e) {ok = false;}}
```

2.5.3. Acrescentando a validação da cadeia de certificação

O trecho de código da Listagem 2.27 mostra como o método `validate()`, cuja implementação já foi mostrada anteriormente nesta seção, pode ser utilizado para validar manualmente não apenas os certificados na ponta da cadeia de certificação, como também, os certificados de CAs intermediárias. Nas linhas de 4 a 18, uma cadeia de certificação é criada. Na linha 20, o método `validate()` é usado para verificar o certificado da CA intermediária. Na linha 22, o método `validate()` é usado para verificar o certificado de usuário final.

Listagem 2.27. Validação manual dos certificados intermediários.

```
1 public static void main(String[] args) {
2   Security.addProvider(new BouncyCastleProvider());
3   try {
4     KeyPair rkp = CertUtils.genRSAKeyPair();
5     X509Certificate root = CertUtils.buildSelfSignedCert(rkp);
6
7     KeyPair mkp = CertUtils.genRSAKeyPair();
8     X509Certificate middle = CertUtils.buildMiddleCert(mkp.getPublic(),
9             "CN=Middle CA Certificate", rkp.getPrivate(), root);
10
11    // generate end entity certificate
12    KeyPair ekp = CertUtils.genRSAKeyPair();
13    X509Certificate user = CertUtils.buildEndCert( ekp.getPublic(),
14            "CN=User Certificate", mkp.getPrivate(), middle);
15
16    X500Principal issuer=new X500Principal("CN=Root Certificate");
17    X500Principal subj1=new X500Principal("CN=Middle CA Certificate");
18    X500Principal subj2=new X500Principal("CN=User Certificate");
19
20    if (validate(middle,root,issuer,subj1,null))
21      System.out.println("Middle certificate successfully validated");
22    if (validate(user,middle,subj1,subj2,null))
23      System.out.println("User Certificate successfully validated");
24  } catch (Exception ex) {System.out.println(ex);}
25 }
```

O trecho de código da Listagem 2.28 mostra a validação da cadeia de certificação para uma conexão SSL/TLS. As linhas de 4 a 8 estabelecem o canal seguro e obtém as informações da sessão. O restante do programa faz a verificação da cadeia de certificação. O processo possui dois passos preparatórios e um passo de verificação: (*i*) Obtenção do certificado raiz e dos outros certificados da cadeia de certificação, de modo que a cadeia possa ser completamente construída (linhas de 10 a 20); (*ii*) Designação dos certificados raízes como âncoras de confiança da cadeia (linhas 22 e 25); (*iii*) Validação da cadeia de certificação do certificado do servidor (linhas 27 até 36).

Na linha 28, é criado um validador de cadeia de certificação (instância de `CertPathValidator`). Finalmente, as linhas de 30 a 35 realizam a verificação. O método `validate()` da linha 32 verifica a cadeia passada como parâmetro. Se exceções não ocorrerem (verificação bem sucedida), as políticas de uso do certificado (linha 33) e a chave pública (linha 34) podem ser obtidas com confiança. No caso do SSL/TLS, o conteúdo da URL é recuperado (linha 41). Senão, alguma exceção é disparada (linhas 38 e 39).

Vale observar que o exemplo da Listagem 2.28 não verifica se o certificado do servidor, ou algum certificado da cadeia de certificação, foi revogado.

Listagem 2.28. Validação da cadeia de certificação no SSL/TLS.

```
1 public static void main(String[] args) throws Exception {
2  SSLSocket s = null; boolean ok = true;
3  try {
4    SSLSocketFactory f=(SSLSocketFactory)SSLSocketFactory.getDefault();
5    s = (SSLSocket) f.createSocket("www.google.com", 443);
6    s.startHandshake();//all validations happen after handshake
7    SSLSession session = s.getSession();
8    Principal peerPrincipal = session.getPeerPrincipal();
9
10   // Step 1. Obtain root certs and cert path to validate
11   Certificate[] certs = session.getPeerCertificates();
12   X509Certificate[] x509certs = new X509Certificate[certs.length-1];
13   for (int i = 0; i < certs.length-1; i++) {
14      x509certs[i] = (X509Certificate) certs[i];
15   }
16   X509Certificate anchor = (X509Certificate) certs[certs.length-1];
17
18   List l = Arrays.asList(x509certs);
19   CertificateFactory cf=CertificateFactory.getInstance("X.509","SUN");
20   CertPath cp = cf.generateCertPath(l);
21
22   // Step 2. Create a PKIXParameters with the trust anchors
23   TrustAnchor ta = new TrustAnchor(anchor, null);
24   PKIXParameters params=new PKIXParameters(Collections.singleton(ta));
25   params.setRevocationEnabled(false);//Revocation status ignored!
26
27   // Step 3. Use a CertPathValidator to validate the certificate path
28   CertPathValidator cpv = CertPathValidator.getInstance("PKIX","SUN");
29
30   // validate certification path with specified params
31   PKIXCertPathValidatorResult cpvr
32        = (PKIXCertPathValidatorResult) cpv.validate(cp, params);
33   PolicyNode policyTree = cpvr.getPolicyTree();
34   PublicKey subjectPK = cpvr.getPublicKey();
35   System.out.println("Certificate Chain successfully validated");
36   System.out.println(subjectPK);
37
38 }catch(CertificateException|InvalidAlgorithmParameterException|
39  NoSuchAlgorithmException|CertPathValidatorException e){ok = false;}
40
41 if(ok){CertUtils.handleSocket(s);}
42 else  {System.out.println("Something wrong in cert validation.");}}
```

2.5.4. Acrescentando a validação dos nomes do sujeito e do emissor

No trecho de código da Listagem 2.29, não apenas o período de validade (linhas 5 e 6) e a assinatura do certificado do cliente (linha 12) são verificados, como também os nomes do sujeito (contido no certificado) e do emissor são validado contra os nomes associados à aplicação. Este exemplo supõe que as verificações da cadeia de certificação e da revogação dos certificados já foram realizadas do lado de fora do método.

Na linha 21, o nome do emissor do certificado é comparado ao nome esperado pela aplicação. Enquanto na linha 24, o nome do sujeito do certificado é comparado ao nome esperado pela aplicação. Vale observar que estes nomes são recuperados de um certificado no formato X.509 e, por isto, seguem este padrão de formatação.

Listagem 2.29. Validação do nome do Sujeito ou Host.

```
1 public static boolean validate(X509Certificate cert,X509Certificate ca,
2   X500Principal issuer, X500Principal subj, Date date) {
3   boolean ok = false;
4   try {
5     if (date != null) { cert.checkValidity(date);}
6     else              { cert.checkValidity();    }
7     ok = true;
8   }catch(CertificateExpiredException|CertificateNotYetValidException e)
9   { ok = false; System.out.println(e); }
10
11  if (ok) { //DANGER: PubKey may not have been validated !!!!
12    try { ok = false; cert.verify(ca.getPublicKey());ok = true; }
13    catch (CertificateException ex) {
14      ok = false; System.out.println(ex);
15    } catch (NoSuchAlgorithmException | InvalidKeyException |
16             NoSuchProviderException | SignatureException ex)
17    { ok = false; System.out.println(ex); }
18  }
19  if (ok) {
20    ok = false;
21    if (cert.getIssuerX500Principal().equals(issuer)){ ok =  true;}
22    else { System.out.println("Issuer name mismatch"); ok = false;}
23
24    if (ok && cert.getSubjectX500Principal().equals(subj)){ok = true;}
25    else { System.out.println("Subject name mismatch"); ok = false;}
26  }
27  return ok;}
```

2.5.5. Validação da lista de certificados revogados

O último exemplo desta seção mostra as verificações da cadeia de certificação e da revogação dos certificados. O trecho de código da Listagem 2.30 faz a verificação da cadeia de certificação assim como do *status* de revogação do certificado do servidor por meio de uma CRL. O processo possui dois passos preparatórios e um passo de verificação: (*i*) Obtenção do certificado raiz e dos outros certificados da cadeia de certificação, de modo que a cadeia possa ser completamente construída (linhas de 4 a 8); (*ii*) Designação dos certificados raízes como âncoras de confiança da cadeia (linhas 10 e 12); (*iii*) Validação da cadeia de certificação e do *status* de revogação do certificado do servidor (linhas 14 até 43). Somente o terceiro passo é detalhado.

Nas linhas de 15 a 22, é criado um validador de cadeia de certificação (instância de `CertPathValidator`) com quatro opções configuradas: (*a*) falha suavemente se não obtiver a CRL e nem a verificação por OCSP, (*b*) sem mecanismo de recuperação, (*c*) verifica apenas os certificados na ponta da cadeia, e (*d*) prefere a verificação por CRL em vez de OCSP.

Nas linhas de 25 a 30, uma CRL associada ao certificado raiz (ancora de confiança) é recuperada de modo confiável (com o método `revokeCerts()`) e adicionada aos parâmetros da verificação. Finalmente, as linhas de 32 a 38 realizam a verificação. O método `validate()` da linha 34 verifica a cadeia e a CRL passadas como parâmetros. As exceções, incluindo a exceção de validação de cadeia de certificação, são capturadas nas linhas 35 e 36.

Listagem 2.30. Validação da lista de certificados revogados.

```
1 public static void main(String[] args) {
2  Security.addProvider(new BouncyCastleProvider());
3  try {
4    // Step 1. Obtain root certs and cert path to validate
5    X509Certificate[] certs = getCertificateList();
6    List l = Arrays.asList(certs);
7    CertificateFactory cf=CertificateFactory.getInstance("X.509","SUN");
8    CertPath cp = cf.generateCertPath(l);
9
10   // Step 2. Create a PKIXParameters with the trust anchors
11   TrustAnchor ta = new TrustAnchor(rootCA,null);
12   PKIXParameters params=new PKIXParameters(Collections.singleton(ta));
13
14   // Step 3. Use a CertPathValidator to validate the certificate path
15   CertPathValidator cpv = CertPathValidator.getInstance("PKIX","SUN");
16   PKIXRevocationChecker rc =
17         (PKIXRevocationChecker)cpv.getRevocationChecker();
18   rc.setOptions(EnumSet.of(Option.SOFT_FAIL));
19   rc.setOptions(EnumSet.of(Option.NO_FALLBACK));
20   rc.setOptions(EnumSet.of(Option.ONLY_END_ENTITY));
21   rc.setOptions(EnumSet.of(Option.PREFER_CRLS));
22   params.addCertPathChecker(rc);
23
24   // now it revokes the very same list of certificates
25   X509CRL crl=CertUtils.revokeCerts(rootCA,rootKP.getPrivate(),certs);
26   List list = new ArrayList();list.add(crl);
27   CertStoreParameters csp = new CollectionCertStoreParameters(list);
28   CertStore store = CertStore.getInstance("Collection", csp);
29   params.addCertStore(store);
30   //params.setRevocationEnabled(false);//remove comment to desable CRL
31
32   // validate certification path with specified params
33   PKIXCertPathValidatorResult cpvr =
34         (PKIXCertPathValidatorResult) cpv.validate(cp, params);
35 }catch(InvalidAlgorithmParameterException|CertPathValidatorException|
36       CertificateException                |NoSuchAlgorithmException e)
37                   { System.out.println(e);
38 }catch(Exception e){ System.out.println(e);}}
```

2.6. Considerações finais

Três anos após a publicação do primeiro texto voltado à criptografia simétrica, este capítulo dá outro passo na direção da construção de um arcabouço que proporcione ao programador não especialista em criptografia a confiança necessária para a codificação segura de software criptográfico. Alguns dos assuntos deixados para trás naquela ocasião foram recuperados e (esperamos) abordados de modo satisfatório para o leitor programador. Em linhas gerais, este capítulo abordou a criptografia assimétrica em Java com a API padrão da plataforma, apontando diversas maneiras de usar mal a API, causando vulnerabilidades, assim como também mostrando maneiras de usar bem a criptografia de chave pública. Em particular, foram cobertos os maus usos criptográficos associados às configurações inseguras de algoritmos assimétricos para acordo de chaves, encriptação e assinaturas digitais, à utilização programática de curvas elípticas inseguras e à validação incompleta de certificados digitais.

O desenvolvimento de software criptográfico é cheio de decisões de projeto e armadilhas de programação que confundem o programador comum, não acostumado à complexidade da tecnologia criptográfica. Neste texto, tentamos evitar a postura combativa contra os programadores, favorecendo o ponto de vista do desenvolvedor de software no tratamento dos maus usos criptográficos. Deste modo, esperamos contribuir para o fortalecimento da **segurança de software criptográfico** [Braga 2017] como um novo campo de estudo preocupado com o desenvolvimento sistemático de software criptográfico seguro. Esta nova área de pesquisa adota uma bordagem preventiva, buscando não apenas se antecipar aos maus usos criptográficos, mas também compreender as necessidades e dificuldades dos desenvolvedores de software.

A observação sistemática do mundo real, por meio de experimentos e análises empíricas, trouxe, com o tempo, a perspectiva necessária para perceber que há uma grande lacuna entre o que os criptologistas veem como maus usos de criptografia e aquilo que os desenvolvedores percebem como uso inseguro da tecnologia criptográfica. Este texto contribui para preencher essa lacuna, oferecendo para a comunidade de desenvolvedores de software insumos práticos que viabilizam tanto a aplicação no curto prazo, quanto os estudos futuros.

Como era de se esperar, o tema é fértil e o assunto é inesgotável. Por isto, há tópicos que gostaríamos de ter tratado, mas que por falta de espaço, foram deixados para outra oportunidade. Em particular, não foram explorados os maus usos criptográficos em outras linguagens de programação, como por exemplo, JavaScript. Além disso, a criptografia pós-quântica não fez parte deste texto porque ainda não há evidência empírica de como esta tecnologia pode ser mal utilizada na prática em atividades rotineiras de programação de software. Porém, a padronização e popularização da criptografia pós-quântica podem trazer oportunidades para uma possível continuação deste texto, no futuro.

Agradecimentos

Este trabalho foi realizado no Laboratório de Segurança e Criptografia (LASCA) do Instituto de Computação, Universidade Estadual de Campinas. Os autores agradecem o apoio financeiro do projeto ATMOSPHERE (**A**daptive, **T**rustworthy, **M**anageable, **O**rchestrated, **S**ecure, **P**rivacy-assuring, **H**ybrid, **E**cosystem for **RE**silient Cloud Computing), RNP e MCTIC, no âmbito to acordo de cooperação Número 51119. *ATMOSPHERE is funded by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement No 777154*.

Referências

[CRI 2012] (2012). The CRIME attack: netifera.com. *URL: http://netifera.com*.

[BRE 2013] (2013). The BREAH attack: SSL GONE IN 30 SECONDS. *URL: http://breachattack.com*.

[Log 2016] (2016). The logjam attack and weak diffie-hellman. *URL: https://weakdh.org*.

[Acar et al. 2017] Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M. L., and Stransky, C. (2017). Comparing the usability of cryptographic apis. In *Proceedings of the 38th IEEE Symposium on Security and Privacy*.

[Adrian et al. 2015] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., and Others (2015). Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM.

[Akhawe et al. 2013] Akhawe, D., Amann, B., Vallentin, M., and Sommer, R. (2013). Here's My Cert, So Trust Me, Maybe?: Understanding TLS Errors on the Web. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 59–70. International World Wide Web Conferences Steering Committee.

[Alashwali 2013] Alashwali, E. S. (2013). Cryptographic vulnerabilities in real-life web servers. In *Third International Conference on Communications and Information Technology (ICCIT)*, pages 6–11. Ieee.

[AlFardan et al. 2013] AlFardan, N. J., Bernstein, D. J., Paterson, K. G., Poettering, B., and Schuldt, J. C. N. (2013). On the security of rc4 in tls. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 305–320, Berkeley, CA, USA. USENIX Association.

[Anderson 1993] Anderson, R. (1993). Why cryptosystems fail. *Proceedings of the 1st ACM Conference on Computer . . .* , pages 215–227.

[Aviram et al. 2016] Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J. A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., and Shavitt, Y. (2016). DROWN: Breaking TLS using SSLv2. *Proceedings of the 25th USENIX Security Symposium*, (August):1–18.

[Bernstein 2006] Bernstein, D. J. (2006). Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer.

[Bernstein et al. 2013] Bernstein, D. J., Lange, T., et al. (2013). Safecurves: choosing safe curves for elliptic-curve cryptography. *URL: http://safecurves.cr.yp.to*.

[Bleichenbacher 1998] Bleichenbacher, D. (1998). Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs# 1. In *Annual International Cryptology Conference*, pages 1–12. Springer.

[Bleichenbacher et al. 2017] Bleichenbacher, D., Duong, T., Kasper, E., and Nguyen, Q. (2017). Project Wycheproof - Scaling crypto testing. In *Real World Crypto Symposium*, New York, USA.

[BlueKrypt] BlueKrypt. Cryptographic Key Length Recommendation. *URL: http://www.keylength.com*.

[Boneh 1999] Boneh, D. (1999). Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, pages 1–16.

[Bos et al. 2014] Bos, J. W., Halderman, J. A., Heninger, N., Moore, J., Naehrig, M., and Wustrow, E. (2014). Elliptic curve cryptography in practice. In *Financial Cryptography and Data Security*, pages 157–175. Springer.

[BouncyCastle 2018] BouncyCastle (2018). The Legion of the Bouncy Castle. *URL: http://www.bouncycastle.org/*.

[Braga and Dahab 2015a] Braga, A. and Dahab, R. (2015a). A Survey on Tools and Techniques for the Programming and Verification of Secure Cryptographic Software. In *XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2015*, pages 30–43, Florianópolis, SC, Brazil.

[Braga and Dahab 2015b] Braga, A. and Dahab, R. (2015b). Introdução à Criptografia para Programadores: Evitando Maus Usos da Criptografia em Sistemas de Software. In *Caderno de minicursos do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2015*, pages 1–50. Sociedade Brasileira de Computação.

[Braga and Dahab 2016] Braga, A. and Dahab, R. (2016). Mining Cryptography Misuse in Online Forums. In *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 143–150.

[Braga and Dahab 2017] Braga, A. and Dahab, R. (2017). A Longitudinal and Retrospective Study on How Developers Misuse Cryptography in Online Communities. In *XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'17)*, Brasília, DF, Brazil.

[Braga et al. 2017a] Braga, A., Dahab, R., Antunes, N., Laranjeiro, N., and Vieira, M. (2017a). Practical Evaluation of Static Code Analysis Tools for Cryptography: Benchmarking Method and Case Study. In *The 28th IEEE International Symposium on Software Reliability Engineering (ISSRE)*.

[Braga et al. 2017b] Braga, A., Marino, F., and Santos, R. (2017b). Segurança de Aplicações Blockchain Além das Criptomoedas. In SBC, editor, *Livro de minicursos do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'17)*, chapter 3.

[Braga 2017] Braga, A. M. (2017). Towards the safe development of cryptographic software (Rumo ao desenvolvimento seguro de software criptográfico).

[Calvo 2018] Calvo, R. (2018). The true cost of certificate authority trials: Can you trust them? *URL: https://www.isc2.org/News-and-Events/Infosecurity-Professional-Insights*.

[Chandra et al. 2002] Chandra, P., Messier, M., and Viega, J. (2002). Network security with OpenSSL. *O'Reily, June*.

[Diffie and Hellman 1976] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654.

[Egele et al. 2013] Egele, M., Brumley, D., Fratantonio, Y., and Kruegel, C. (2013). An empirical study of cryptographic misuse in android applications. *ACM SIGSAC conference on Computer & communications security - CCS '13*, pages 73–84.

[Eldewahi et al. 2015] Eldewahi, A. E. W., Sharfi, T. M. H., Mansor, A. A., Mohamed, N. A. F., and Alwahbani, S. M. H. (2015). Ssl/tls attacks: Analysis and evaluation. In *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*, pages 203–208.

[Fahl et al. 2012] Fahl, S., Harbach, M., and Muders, T. (2012). Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *ACM conference on Computer and communications security*, pages 50–61.

[Fardan and Paterson 2013] Fardan, N. A. and Paterson, K. (2013). Lucky thirteen: Breaking the TLS and DTLS record protocols. *Security and Privacy (SP), 2013 IEEE Symposium on (2013)*.

[Georgiev et al. 2012] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., and Shmatikov, V. (2012). The most dangerous code in the world. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, page 38. ACM Press.

[Gluck et al. 2013] Gluck, Y., Harris, N., and Angel, A. (2013). BREACH: Reviving the CRIME Attack. In *Black Hat Conference*.

[Google] Google. Google Android Developers. *URL: https://groups.google.com/forum/#!forum/android-developers*.

[Gutmann a] Gutmann, P. Everything you Never Wanted to Know about PKI but were Forced to Find Out. *URL: https://www.cs.auckland.ac.nz/ pgut001/pubs/pkitutorial.pdf*.

[Gutmann b] Gutmann, P. Godzilla crypto tutorial - Part 2, Key Management and Certificates.

[Gutmann 2002] Gutmann, P. (2002). Lessons Learned in Implementing and Deploying Crypto Software. *Usenix Security Symposium*.

[Hankerson et al. 2004] Hankerson, D., Vanstone, S., and Menezes, A. (2004). *Guide to elliptic curve cryptography*.

[IACR 2012] IACR (2012). Real world crypto symposium. *URL: https://rwc.iacr.org/index.html*.

[Jonsson and Burt Kaliski 2003] Jonsson, J. and Burt Kaliski (2003). RSA Laboratories Public-Key Cryptography Standards (PKCS)#1: RSA Cryptography Specifications Version 2.1. *URL: https://tools.ietf.org/html/rfc3447*.

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

[Marlinspike 2009] Marlinspike, M. (2009). New tricks for defeating SSL in practice. *URL: https://blackhat.com/presentations/bh-europe-09/Marlinspike/blackhat-europe-2009-marlinspike-sslstrip-slides.pdf*.

[Mart and Hern 2013] Mart, V. G. and Hern, L. (2013). Implementing ECC with Java Standard Edition 7. *International Journal of Computer Science and Artificial Intelligence*, 3(4):134–142.

[Menezes et al. 1996] Menezes, A. J., Van Oorschot, P. C., and Vanstone, S. A. (1996). *Handbook of applied cryptography*. CRC press.

[Miller 1985] Miller, V. S. (1985). Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer.

[Möller et al. 2014] Möller, B., Duong, T., and Kotowicz, K. (2014). The POODLE attack. *URL: https://www.openssl.org/ bodo/ssl-poodle.pdf*.

[NIST 2012] NIST (2012). Recommendation for Key Management – Part 1: General (Revision 3). *URL: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf*.

[NIST 2013] NIST (2013). Digital Signature Standard (DSS).

[OpenSSL.org] OpenSSL.org. OpenSSL Cryptography and SSL/TLS toolkit. *URL: OpenSSL.org*.

[Oracle a] Oracle. Java Cryptography Architecture (JCA) Reference Guide. *URL: docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html*.

[Oracle b] Oracle. Java Cryptography Architecture Oracle Providers Documentation for Java Platform Standard Edition 8. *URL: docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html*.

[OWASP 2015] OWASP (2015). OWASP Testing Project v4. *URL: https://www.owasp.org/index.php/OWASP_Testing_Project*.

[Rescorla 2018] Rescorla, E. (2018). The transport layer security (tls) protocol version 1.3. *URL: https://tools.ietf.org/html/rfc8446*.

[Ristic 2015] Ristic, I. (2015). *OpenSSL cookbook*. Feisty Duck, 2nd. ed. (version 2.1-draft published in june 2018) edition.

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

[Schneier 1998] Schneier, B. (1998). Cryptographic design vulnerabilities. *Computer*, (September):29–33.

[SEC 2000] SEC, S. (2000). Sec 2: Recommended elliptic curve domain parameters, version 1. *Standards for Efficient Cryptography Group, Certicom Corp (http://www.secg.org/)*.

[SEC 2010] SEC, S. (2010). Sec 2: Recommended elliptic curve domain parameters, version 2. *Standards for Efficient Cryptography Group, Certicom Corp ()http://www.secg.org/*.

[Shuai et al. 2014] Shuai, S., Guowei, D., Tao, G., Tianchang, Y., and Chenjie, S. (2014). Modelling Analysis and Auto-detection of Cryptographic Misuse in Android Applications. In *IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 75–80.

[Stallings 2003] Stallings, W. (2003). *Cryptography and network security, principles and practices*.

[Sullivan 2018] Sullivan, N. (2018). A detailed look at rfc 8446 (a.k.a. tls 1.3). *URL: https://blog.cloudflare.com/rfc-8446-aka-tls-1-3*.

[Wikipedia 2018] Wikipedia (2018). Netscape. *URL: https://en.wikipedia.org/wiki/Netscape*.

Capítulo 3

Análise de Binários e Sistemas Assistida por *Hardware*

Marcus Botacin[1], Paulo Lício de Geus[2], and André Grégio[1]

[1]Universidade Federal do Paraná (UFPR)
[2]Universidade Estadual de Campinas (UNICAMP)

Abstract

Binary analysis is a key step for security procedures, such as systems inspection and validation. Modern architectures are powered by many resources and features which end up in more efficient—and also more complex—applications and codes. These resources and features include virtual machine support, BIOS and chipset code execution, isolated enclaves and even external hardware. In this chapter, we introduce these features and the challenges they pose for current systems monitoring. We also present new opportunities for developing monitoring and analysis solutions based on the presented features.

Resumo

A análise de binários é uma etapa fundamental dos processos de segurança, tais como inspeção e validação de sistemas. Arquiteturas modernas possuem diversos mecanismos e recursos que resultam na construção de aplicações e códigos mais eficientes, porém também mais complexos. Estes recursos incluem suporte a máquinas virtuais, execução de código pela `BIOS` *e pelo chipset, enclaves isolados e até mesmo componentes de hardware externos. Neste capítulo, introduzimos estes mecanismos, os desafios que estes trazem para a monitoração de sistemas e também novas possibilidades para a construção de mecanismos e técnicas de monitoração e análise.*

3.1. Introdução

No mundo atual, os sistemas computacionais tem papel de destaque, visto que as capacidades destes suportam muitos dos aspectos da vida cotidiana, tal qual pode ser verificado pela adoção de sistemas de *Internet Banking* [G1 2017] e de iniciativas de governo eletrônico [do Brasil 2018]. No entanto, a medida em que cresce a dependência da sociedade com relação a estes sistemas, crescem também as ameças a estes e seus usuários. Os mesmos desenvolvimentos que trouxeram benefícios a sociedade, trouxeram novos riscos, como as fraudes bancárias digitais [Grégio et al. 2013] e o *cybercrime* [Kaspersky 2015]. Para lidar com estes novos desafios, organizações e usuários adotam soluções de análise [Sikorski and Honig 2012], que permitem a especialistas obter uma melhor compreensão das ameaças para o desenvolvimento de mecanismos de remediação, proteção e prevenção.

Contudo, a batalha entre atacantes e analistas constitui uma verdadeira corrida armamentista (*arms-race*), visto que os atacantes continuamente desenvolvem técnicas de anti-análise [Botacin et al. 2017] para evitar a descoberta e inspeção de seus artefatos. Deste modo, soluções de análise tradicionais [Botacin et al. 2018c] não são mais suficientes para uma cobertura completa das ameaças atuais, visto que o uso de técnicas de anti-análise tem crescido tanto no cenário nacional [Botacin et al. 2015] quanto global [Branco et al. 2012, Barbosa and Branco 2014]. Deste modo, o desenvolvimento de soluções de análise mais eficientes é fundamental para garantir a manutenção da segurança de aplicações atuais e futuras. Contudo, este desenvolvimento é desafiador, sobretudo para soluções completamente baseadas em *software*, dada a inerente possibilidade de subversão de um componente de *software* voltado a proteção por um componente de *software* malicioso.

Avanços recentes no desenvolvimento de *hardware* tem apresentado novos recursos atrelados aos processadores modernos, como máquinas virtuais de *hardware* e modos de execução isolados do restante do sistema. O surgimento destes recursos tem impulsionado o desenvolvimento de soluções de análise com suporte de *hardware* que podem proporcionar aos analistas recursos e capacidades que os levem um passo a frente dos atacantes. Desta forma, esse capitulo se dedica a apresentar estes recursos, discutir seus pontos fortes e limitações, bem como apontar oportunidades de pesquisa existentes.

Neste capítulo, consideramos procedimentos de análise em um sentido amplo, cobrindo desde mecanismos para engenharia reversa e *debugging*, em um sentido de análise mais estrito, até a monitoração de sistemas em tempo real, a detecção de ataques e procedimentos forenses, aplicações que apresentam requisitos de análise diversos.

Este capítulo foi planejado para ser apresentado de forma autocontida, isto é, não requerindo que o leitor consulte referências externas para sua compreensão. Contudo, assumimos que a audiência possui conhecimentos básicos na área de segurança e, em especial, na área de análise de aplicações, de modo que possamos partir do ponto das diferenças trazidas pelos desenvolvimentos recentes em relação aos já conhecidos pelos leitores.

De um modo geral, nossa contribuição é atualizar analistas, profissionais da área e estudantes em relação ao estado da arte em soluções de análise e apontar possibilidades

futuras que estes possam explorar em suas pesquisas e/ou atividades profissionais.

Este capítulo está organizado da seguinte forma: na Seção 3.2, apresentamos os principais recursos de inspeção e características das soluções de análise modernas. A implementação destes recursos é o objetivo do uso das tecnologias e técnicas apresentadas nas demais seções; na Seção 3.4, apresentamos diferentes mecanismos arquiteturais e tecnologias a serem usados para implementar os diferentes recursos de inspeção previamente apresentados; na Seção 3.3, apresentamos as técnicas de monitoração empregadas de modo a prover os recursos de inspeção mencionados através do uso das tecnologias anteriormente apresentadas; na Seção 3.5, discutimos aplicações de análise com base nas tecnologias e técnicas de implementação apresentadas; na Seção 3.6, traçamos um panorama das soluções apresentadas e discutimos problemas em aberto; finalmente, apresentamos nossas conclusões na Seção 3.7.

3.2. Recursos de Inspeção

Este capítulo versa sobre a análise de binários e sistemas. Contudo, o termo análise é suficientemente amplo para abranger distintos recursos de inspeção. Nesta seção, apresentamos os principais recursos presentes nas soluções modernas e seus respectivos cenários de aplicação.

3.2.1. Transparência

Soluções de análise que visem inspecionar aplicações específicas, tais como *debuggers* e analisadores de *malware*, devem ser transparentes ao objeto monitorado, isto é, não devem permitir que o objeto sob análise identifique sua execução em um ambiente monitorado. Isto é essencial para garantir a correta execução dos objetos, visto que muitas aplicações modernas são equipadas com técnicas de anti-análise, evitando, assim, procedimentos de engenharia reversa. As técnicas de anti-análise são empregadas tanto por aplicações legítimas, como *anti-cheats* de plataformas digitais [Li et al. 2004], quanto por aplicações maliciosas, como exemplares de *malware*, que visam evadir os processos de análise [Chen et al. 2008].

A identificação dos ambientes de análise se dá através de quatro principais mecanismos [Botacin et al. 2017]:

1. ***Fingerprinting***: Técnicas de *fingerprinting* consistem em localizar identificadores únicos no sistema alvo, como o IP público de um sistema de análise, permitindo o lançamento de ataques direcionados ou a evasão de procedimentos de análise. Este tipo de técnica é extensamente abordada na literatura [Liu et al. 2014] e não será coberta em profundidade neste capítulo.

2. **Verificação de integridade**: A verificação de integridade consiste em detectar a injeção de código no processo em execução—como nos procedimentos de injeção de bibliotecas (*DLL Injection*, por exemplo)—ou na alteração de estruturas do sistema—como no *hooking* de APIs. Desta forma, um exemplar de *malware* pode, por exemplo, detectar que uma biblioteca de monitoração foi carregada e, assim, terminar sua execução.

3. **Identificação de efeitos colaterais de execução em ambientes instrumentados**: A detecção de efeitos colaterais de execução consiste em identificar comportamentos anômalos do sistema devido a instrumentação deste para fins de monitoração. Muitas soluções de análise se baseiam em máquinas virtuais de *software* ou em emuladores, que não traduzem fielmente o comportamento dos sistemas reais. Por exemplo, instruções atômicas em máquinas reais podem ser executadas em múltiplos passos e, emuladores [Willems et al. 2012a], o que permite a um atacante agir maliciosamente apenas em máquinas reais e, portanto, atacar apenas os usuários finais. Tais instruções são conhecidas como "*red pills*" [Martignoni et al. 2009, Shi et al. 2014] e atualmente a identificação destas pode ser realizada de maneira automática [Paleari et al. 2009].

4. **Identificação de variações nas medidas de tempo**: A evasão por detecção de medidas divergentes de tempo se baseia no fato de que os códigos de instrumentação adicionados ao sistema—sejam no *kernel*, em bibliotecas ou no próprio emulador—consomem ciclos de CPU adicionais em relação a execução do código monitorado, resultando em variações no tempo de execução total do objeto monitorado. Um código que verifique seu próprio tempo de execução é capaz de inferir sua execução em um ambiente monitorado e, assim, evadir o processo de análise.

Formalmente, o conceito de transparência de um ambiente e da execução neste é definido por cinco requisitos [Dinaburg et al. 2008a]:

1. **Monitoração com privilégios superiores**: O mecanismo de análise deve ser mais privilegiado do que o objeto sendo monitorado. Como exemplo, a monitoração de recursos em nível de usuário deve ser implementada por mecanismos de *kernel*. Recursos de *kernel*, por sua vez, devem ser monitorados por uma entidade externa, como um *hypervisor*, e assim sucessivamente.

2. **Execução livre de efeitos colaterais não privilegiados**: Qualquer instrução que introduza um efeito colateral no sistema deve ser tratada por uma exceção capaz de esconder este efeito.

3. **Instruções com semânticas idênticas ao caso base**: Cada instrução executada no ambiente de monitoração deve ter o mesmo efeito e conduzir a execução da mesma instrução seguinte do que sua execução no ambiente não-monitorado.

4. **Tratamento transparente de exceções**: Dado uma exceção na *i-ésima* instrução, o fluxo de execução após o tratador deve retornar para a *(i+1)-ésima* instrução.

5. **Medidas de tempo idênticas**: A medida de tempo decorrido para a execução no ambiente monitorado e no ambiente não-monitorado devem ser idênticas.

Arquiteturas modernas proveem recursos de inspeção que permitem monitorar a execução de aplicações sem a necessidade de se injetar código, sendo, portanto, essenciais para o desenvolvimento de soluções transparentes. Além disto, recursos presentes nas arquiteturas modernas, como máquinas virtuais de *hardware*, permitem que o código

monitorado seja executado no processador real, não apresentando, assim, efeitos colaterais de execução. O uso destes recursos para fins de análise transparente é descrito em detalhes nas seções a seguir.

3.2.2. Amplitude de monitoração

Cada modelo de ameaça apresenta diferentes requisitos quanto a amplitude da monitoração, isto é, das diferentes fontes de dados requeridas para a monitoração. Por exemplo, enquanto a detecção de comportamento anômalo pode ser implementada a nível de aplicação, procedimentos forenses podem requerir o *dump* da memória do *kernel*. A implementação de um mecanismo de análise em cada um dos níveis requer a compreensão das possibilidades e limitações arquiteturais das plataformas modernas.

Processadores modernos isolam os diferentes níveis de execução em *rings*. As aplicações de nível de usuário são executadas em `ring` 3, ao passo em que o *kernel* é executado no `ring` 0. Os *rings* 1 e 2 são dedicados a *drivers* e bibliotecas, mas não são utilizados, na prática, pelos sistemas modernos. Os *rings* são ilustrados pela Figura 3.1.

Os privilégios de execução de um nível permitem que este dê garantias sobre os demais. Por exemplo, o *kernel* é capaz de monitorar o nível de usuário sem que este possa subverter o *kernel*, por estar em um nível de execução inferior e, portanto, isolado por *hardware*. O isolamento entre os níveis se dá, por exemplo, através do impedimento do mapeamento de uma região de memória pertencente a uma aplicação operando em um *ring* superior ao que a aplicação solicitante opera.

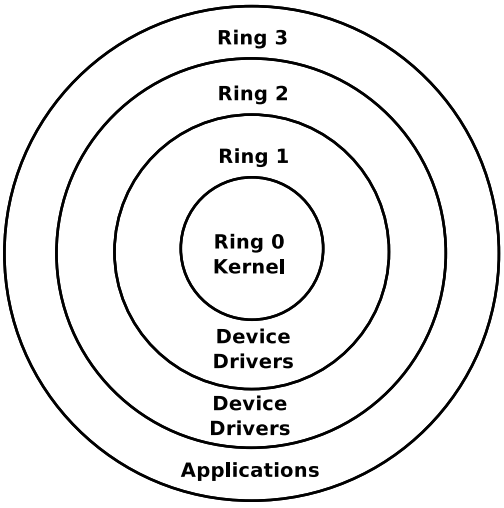


Figura 3.1. Privilégios de Execução. O *ring* 0 é o mais privilegiado e pode monitorar os demais *rings*. As aplicações executam em *ring* 3 e não podem interferir com os demais *rings*.

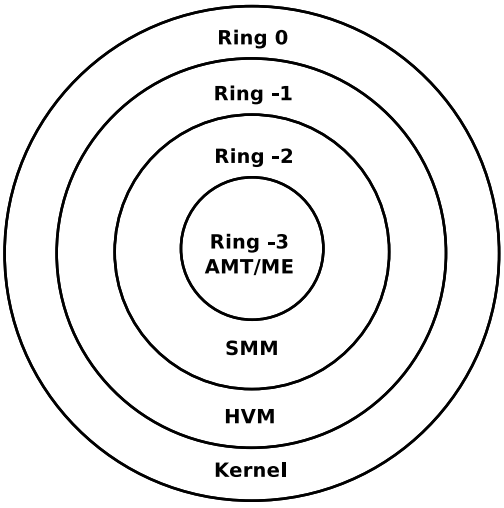


Figura 3.2. Novos *rings* privilegiados. O *ring* −3 é o mais privilegiado e pode monitorar os outros *rings*. Nesta nova configuração, o *kernel* executa dentro de uma máquina virtual de *hardware* (HVM), em *ring* −1.

Sistemas de análise modernos apresentam modelos de ameaças cada vez mais amplos e, com esta necessidade, novos *rings* vem sendo propostos. O uso de máquinas

virtuais de *hardware*—*Hardware Virtual Machines* (HVM)—possibilita que um *hypervisor* monitore tanto o *kernel* quanto o modo usuário do sistema *guest*. Portanto, sistemas baseados em HVM vem sendo chamados de *ring* -1. Similarmente, sistemas baseados na instrumentação do modo SMM—*System Management Mode*—, presente na BIOS, são capazes de monitorar tanto o *kernel* e modo usuário quanto o próprio *hypervisor* em execução no sistema. Desta forma, o modo SMM é denominado *ring* -2. Finalmente, o modo ME (*Management Engine*) dos *chipsets* é capaz de monitorar inclusive o funcionamento da BIOS, sendo, portanto, denominado *ring* -3. Estes novos *rings* são ilustrados pela Figura 3.2.

A monitoração entre os diferentes níveis, contudo, impõe um desafio significativo: a interpretação dos dados, já que que cada nível apresenta um `nível de abstração` diferente. Por exemplo, enquanto a monitoração em nível de usuário é capaz de identificar um dado evento como sendo uma chamada de função, a monitoração do mesmo evento em nível de HVM é capaz apenas de identificar a execução de um dado endereço. As Figuras 3.3 e 3.4 ilustram, respectivamente, as tecnologias de análise apresentadas neste capítulo e seus níveis de abstração.

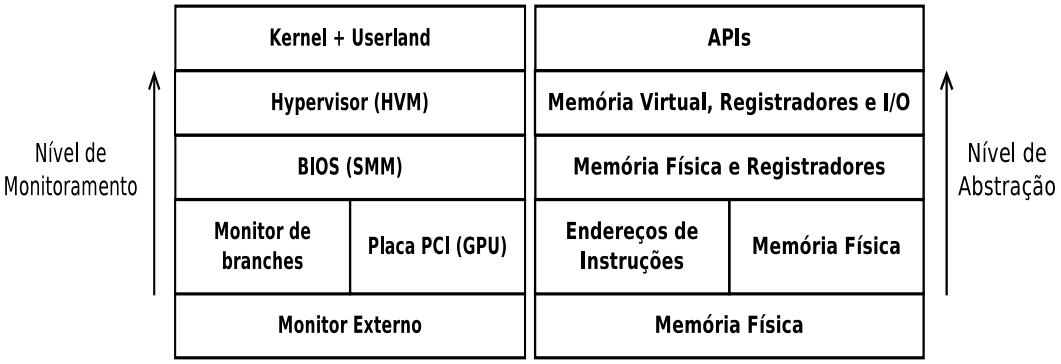


Figura 3.3. Níveis de monitoramento. Cada tecnologia apresentada neste capítulo monitora o sistema em um nível diferente. Os níveis mais altos se aproximam de soluções de *software*, enquanto níveis inferiores operam mais próximo do *hardware*.

Figura 3.4. Níveis de abstração. Cada tecnologia apresentada neste capítulo opera em um nível de abstração diferente. Quanto mais alto o nível, mais próximo de uma informação compreensível para o ser humano. A passagem da informação de um nível de abstração para o outro exige procedimentos de introspecção para a adequação da sua representação.

A diferença de representações entre os níveis, chamada de *gap* semântico, requer que procedimentos de "enriquecimento" dos dados sejam desenvolvidos para aumentar o nível de abstração das informações coletadas. Por exemplo, um sistema HVM precisa previamente mapear endereços em chamadas de função de modo a identificar as chamadas quando monitorando o sistema em tempo real através dos endereços executados. Este procedimento de "enriquecimento" das informações coletadas em um nível de abstração inferior para interpretação em um nível de abstração superior é chamado introspecção.

Ferramentas de análise podem se basear em soluções conhecidas de introspecção, como a LibVMI [LibVMI 2015], ou desenvolver suas próprias soluções [Botacin et al. 2018a].

De uma forma geral, mecanismos de introspecção podem ser classificados em quatro categorias [More and Tapaswi 2014]:

1. **Memória**: A introspecção de memória consiste em reconstruir estruturas a partir das visões de memória obtidas em cada nível. Este tipo de técnica permite, por exemplo, interpretar séries de apontadores como elementos de uma lista de processos. Este tipo de técnica é frequentemente usado por soluções forenses para a reconstrução de contexto a partir de *dumps* de memória.

2. **I/O**: De forma similar a introspecção de processos, procedimentos de reconstrução podem ser aplicados a eventos de I/O para reconstruir a estrutura dos dados sendo transferidos. Este tipo de técnica é frequentemente utilizado por soluções que imponham políticas de segurança durante a transmissão de dados.

3. **Chamadas de sistema**: A introspecção a nível de chamadas de sistema (*syscalls*) é focada em reconstrução de contextos, tais como os processos que invocam uma dada *system call* e seus respectivos argumentos. Dado que processos são uma abstração de alto nível, a identificação destes ocorre através da associação destes com valores em *hardware*, tais como o valor do registrador `CR3`, usado como apontador da tabela de páginas de memória e, portanto, único para cada processo. Este tipo de técnica é frequentemente utilizado por soluções de monitoramento HVM e SMM.

4. **Processos**: A introspecção a nível de processo objetiva obter informações específicas de cada processo, como os recursos acessados por cada processo analisado. Este tipo de introspecção é frequentemente utilizado por soluções antivírus para a detecção de *malware* em nível de espaço de usuário (*userland*).

Na prática, a literatura acadêmica apresenta diversas soluções de análise baseadas em introspecção, tais como Anubis [Bayer et al. 2006, ISECLAB 2010] e Danubis [Neugschwandtner et al. 2010], implementados sobre QEMU [Bellard 2005], e Bitblaze [Song et al. 2008] e Virtice [Quynh and Suzaki 2010], sobre TEMU. Contudo, mecanismos de introspecção se tornaram fundamentais a partir da popularização do uso dos novos recursos de *hardware* para a implementação dos mecanismos de análise. O uso de introspecção pelas diversas soluções são detalhados nas seções a seguir.

3.2.3. Posicionamento e capacidade de ação

Sistemas computacionais modernos são organizados em diversas camadas e o monitoramento em cada uma delas apresenta vantagens e desvantagens significativas. Mais do que abstrações, essas camadas, de fato, cumprem diferentes funções e, portanto, apresentam diferentes desafios para suas instrumentações e impõem diferentes custos. O monitoramento em alto nível, tal como a nível de sistema operacional, apresenta baixo custo de implementação, já que a solução de análise pode utilizar bibliotecas de sistema. Por outro lado, este tipo de monitoramento é mais suscetível a subversão por mecanismos mais privilegiados. O monitoramento em baixo nível, por sua vez, apresenta maior proteção contra

subversão, porém, seus custos de implementação são maiores, dado a limitada possibilidade de reuso de código e os conhecimentos de implementação exigidos. Desta forma, encontrar um balanço entre o nível de proteção requerido por um dado modelo de ameaça e o custo de sua implementação é um desafio significativo.

Além disso, a camada em que o sistema de análise é posicionado influencia a capacidade de ação deste. Sistemas posicionados no mesmo nível do sistema monitorado são capazes de bloquear a execução caso uma ameaça seja detectada. Sistemas posicionados externamente ao objeto monitorado podem ser incapazes de bloquear a execução de um objeto interno, embora possuam capacidade de análise do mesmo. Este cenário é observado quando se delega a tarefa de monitoração a uma entidade de *hardware* externa ao mecanismo de análise, tal como um co-processador. Uma discussão aprofundada deste tipo de solução é apresentada nas seções a seguir.

3.2.4. Carregamento em tempo real

Diferentes tarefas de análise apresentam diferentes requisitos para as ferramentas a serem utilizadas. Enquanto tarefas como análise de *malware* são realizadas em ambientes controlados e previamente configurados, tarefas como a forense são frequentemente realizadas em campo, apresentando significativas restrições para a instalação e configuração de ferramentas no ambiente. Desta forma, uma característica desejável para soluções de análise, em especial as voltadas à forense computacional, é o carregamento da solução em tempo real. Também chamado de *live-loading*, este tipo de carregamento permite ao perito forense inserir a solução de análise no sistema alvo e utilizar o próprio sistema para a inspeção, com garantias de que o carregamento da solução não interfira nos artefatos a serem coletados. Este tipo de carregamento é possível, por exemplo, através do uso de máquinas virtuais de *hardware*, que permitem transformar o sistema *host* em um *guest* da mesma máquina. A implementação deste tipo de recurso é apresentada nas seções a seguir.

3.2.5. Base de código confiável

Soluções de análise requerem um determinado conjunto de códigos para seu funcionamento. Este conjunto inclui todos os códigos que suportam sua operação, desde os códigos responsáveis por implementar a tarefa de monitoração em si aos *frameworks* sob os quais a solução é construída. Este conjunto é denominado base de código confiável (*Trusted Code Base*—TCB), uma vez que deve ser considerado como legítimo e íntegro *a priori*. Quanto mais se confia em recursos de terceiros, maior o TCB. Por exemplo, um *driver* de monitoração operando em *kernel* deve considerar todo o *kernel* como seu TCB, visto que subversões deste podem acarretar em subversões do mecanismo de análise em si.

De um modo geral, soluções de análise visam reduzir o TCB, ampliando suas garantias de segurança e também seu modelo de ameaças. Contudo, estas devem buscar um balanço entre a diminuição do TCB e do aumento dos custos de implementação, visto que, ao não se utilizar de códigos terceiros, a solução deve implementar seus próprios códigos para todas as tarefas. Uma solução em BIOS, por exemplo, enquanto apresenta um TCB minimal, requer a implementação até mesmo de protocolos de comunicação,

visto que este suporte não é provido pela BIOS do sistema.

A Figura 3.5 ilustra o tamanho da base de código confiável (em número de linhas de código) para algumas das soluções apresentadas neste capítulo. Uma solução operando a nível de *kernel* (Linux) requer a inclusão de mais de 10 milhões de linhas de código em seu TCB para contar com os recursos e facilidades oferecidos por este. Uma solução operando a nível de máquina virtual (Xen) requer a inclusão de mais de 500 mil linhas para fazer uso de uma solução de virtualização completa, com todos os *drivers* de dispositivos. Como alternativa para reduzir o TCB, soluções de propósito específico podem ser desenvolvidas (MAVMM [Nguyen et al. 2009]). Neste caso, recursos opcionais e *drivers* não utilizados são removidos da base de código. Soluções inteiramente baseadas em hardware externo (Ki-mon [Lee et al. 2013]) requerem menos de 1000 linhas de código, visto que estas não se baseiam em nenhum suporte *a priori*.

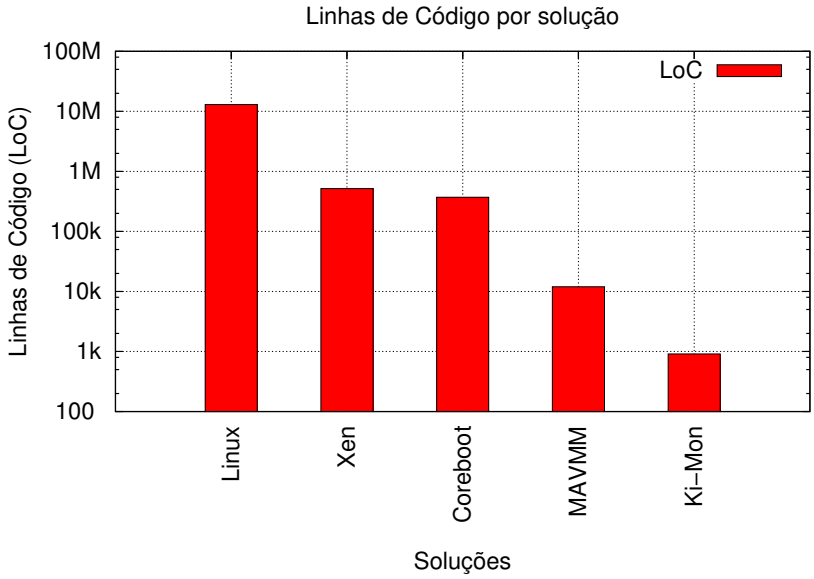


Figura 3.5. Base de código confiável (em número de linhas de código) por aplicação. Quanto maior o nível de abstração, maior a base de código responsável por implementar a solução de monitoramento, portanto, maior a base de código que deve ser confiada (TCB). Como exceção à regra, ferramentas podem ser desenvolvidas com o propósito de minimizar o TCB, como no caso da MAVMM.

3.2.6. Manutenção de integridade e operação

Um sistema voltado a prover garantias de segurança deve manter sua integridade. Uma solução de segurança comprometida pode ser incapaz de detectar ataques, prover informações imprecisas ao analista ou mesmo invalidar perícias forenses. A manutenção de integridade, contudo, é desafiadora em múltiplos aspectos: i) quanto a manutenção do desempenho, uma vez que a solução deve gastar ciclos de CPU para computar *hashes* que atestem sua própria integridade; ii) quanto a operação em meios não confiáveis, uma vez que as soluções atuam diretamente com componentes comprometidos. Uma solução que envie os dados coletados em um sistema para um agente externo deve criptografar sua comunicação uma vez que a interface de rede pode ser comprometida pelo atacante, o

que afetaria os pacotes trafegados. A operação em meios não confiáveis é particularmente desafiadora quando a solução é carregada em um ambiente previamente comprometidos, como em perícias forenses. Nestes casos, a solução deve validar as informações coletadas através da correlação das informações obtidas através de múltiplos fontes de dados, em um processo conhecido como detector de mentiras (*lie detector*). Por exemplo, uma solução baseada em máquinas virtuais pode coletar informações interna e externamente (por introspecção) ao sistema monitorado e considerá-las válidas se estas coincidirem. Este processo atesta que a informação provida é legítima, uma vez que o *hypervisor* faz parte do TCB e, portanto, não foi comprometido.

3.2.7. Sincronia

Uma decisão de projeto importante para se implementar uma solução de monitoração é o tipo de mecanismo de captura e coleta de dados a ser utilizado: síncronos ou assíncronos. Mecanismos síncronos coletam dados em intervalos determinados, sendo, geralmente, implementados através de *polling*. Desta forma, estes são adequados para tarefas que podem ser delegadas a etapas de processamento posteriores (pós-processamento), como o "enriquecimento" dos dados coletados em um traço de execução. Mecanismos assíncronos, por sua vez, coletam os dados quanto determinados eventos ocorrem (*event-driven*), sendo, portanto, adequados para soluções em tempo real, como *debuggers* ou detectores de ataques. Soluções síncronas não são adequadas para mecanismos de tempo real pois um ataque poderia ocorrer no intervalo entre 2 inspeções [Moon et al. 2012], como exemplificado na Figura 3.6.

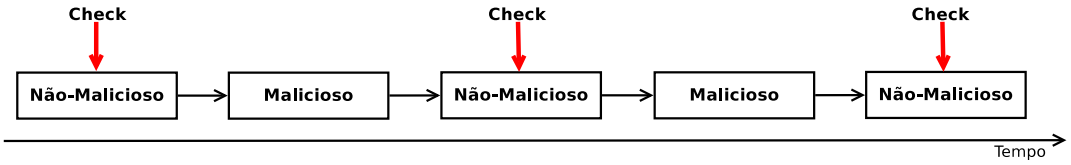


Figura 3.6. Evasão de soluções do tipo *snapshot*. Em soluções que implementem verificações síncronas, ações maliciosas podem não ser detectadas caso estas ocorram no intervalo entre duas inspeções.

Soluções síncronas, no entanto, ainda são populares pois a implementação de mecanismos assíncronos é desafiadora, uma vez que eventos precisos devem ser escolhidos como gatilhos de inspeção. Arquiteturas modernas oferecem suporte em *hardware* para a implementação deste tipo de gatilho através do lançamento de interrupções programáveis. Estas são descritas em detalhes nas seções a seguir.

Desempenho. Enquanto o foco principal das ferramentas de análise são os aspectos de segurança, requisitos de desempenho não podem ser desprezados para muitas aplicações, notavelmente as de operação em tempo-real. Soluções que busquem aplicar políticas de segurança ou detectar ataques em tempo real não devem consumir um tempo significativo processando suas rotinas para não degradar o desempenho da aplicação sendo monitorada. Arquiteturas modernas possuem recursos como contadores de *performance* em hardware que podem ser utilizados para capturar dados com baixo impacto no desempenho, podendo, portanto, serem utilizados para a implementação de mecanismos de análise em tempo real [Opsahl 2013]. Maiores detalhes da aplicação destes são apresentados na

seções a seguir.

3.2.8. Dispensa de atualizações frequentes

Uma característica desejável para os sistemas de análise é que, uma vez desenvolvidos, estes não exijam reestruturação e/ou recompilações frequentes. Este tipo de limitação é presente em abordagens de detecção baseadas em assinatura.

Frequentemente, as soluções de análise e monitoração de artefatos maliciosos [Vasudevan and Yerraballi 2006b, Vasudevan and Yerraballi 2006a] propõem mitigar os efeitos colaterais de execução e de técnicas de anti-análise através de instrumentação dinâmica , utilizando-se, por exemplo, das soluções DynamoRIO [DynamoRIO 2001] e PIN [Intel 2015]. Apesar de eficazes contra as técnicas conhecidas, o processo de mitigação não é escalável [Kang et al. 2009], visto que a descoberta de novas técnicas de anti-análise requerem que novas mitigações sejam implementadas na solução. Arquiteturas modernas tornam as soluções de análises escaláveis, dado que estas não precisam implementar contramedidas para efeitos colaterais de execução, pois estas ocorrem em ambientes de processamento nativos.

3.2.9. Compatibilidade e Integração

Apesar das limitações impostas pelo TCB em relação ao sistema de análise, componentes externos ao sistema são frequentemente construídos sobre outras soluções, reduzindo, portanto, os custos de implementação e propiciando suporte a sistemas e padrões legados. De um modo geral, as soluções de análise atuais [Zhang et al. 2015, Wang et al. 2011] são integradas ao *frontend* GDB, proporcionando que este manipule os elementos arquiteturais da plataforma em *backend*.

3.3. Tecnologias e Implementações

Nesta seção, apresentamos tecnologias e recursos arquiteturais presentes em plataformas modernas que podem ser utilizados para fins de análise.

3.3.1. Máquinas Virtuais de *Hardware*

Hardware Virtual Machines (HVM) são extensões dos modos de operação presentes nos processadores modernos. Tipicamente, processadores `x86-64` apresentam três modos de operação:

1. **Modo de endereçamento real**: Modo de operação original da arquitetura `x86`. Este modo é ativado quando o computador é inicializado (*boot*) e não conta com proteções de memória.

2. **Modo protegido**: Extensão do modo original da arquitetura `x86` e modo nativo de operação dos processadores modernos, fornecendo suporte a memória virtual e paginação.

3. **Modo SMM**: Modo de gerência de recursos do processador, como controle de energia e de temperatura. O código deste modo é executado pela BIOS e pode ser instrumentado, como mostrado nas seções a seguir.

As instruções de virtualização implementadas pelas tecnologias Intel VT-x [Intel 2013] e AMD-v/SVM [AMD 2013] adicionam ao processador dois novos modos de operação: *VM-root* e *VM-non-root*. Estes modos correspondem a operação do *hypervisor* e do *guest*, respectivamente, como mostrado na Figura 3.7, e a transição entre estes modos é dada por *VM-EXITs* (*non-root* para *root*) e *VM-ENTRY* (*non-root*) para *root*), como mostrado na Figura 3.8. As ações que resultam em saídas do *guest* para o *hypervisor* são configuráveis pelo *hypervisor* e tratadas por este através de rotinas instrumentáveis. Desta forma, o *hypervisor* pode monitorar ações específicas da máquina virtual, tais como escritas em registradores e instruções executadas.

Ao contrário de máquinas virtuais de *software*, o *hypervisor* conta com assistência do processador para a identificação e tratamento dos eventos, resultando em um significativo ganho de desempenho. Por exemplo, o próprio *hardware* é responsável por realizar a cópia dos registradores em cada troca de contexto. Além disso, as instruções de ambos os modos (*root* e *non-root*) são executados pelo processador nativo, sem tradução ou emulação, o que torna a execução, ainda que monitorada pelo *hypervisor*, livre de efeitos colaterais.

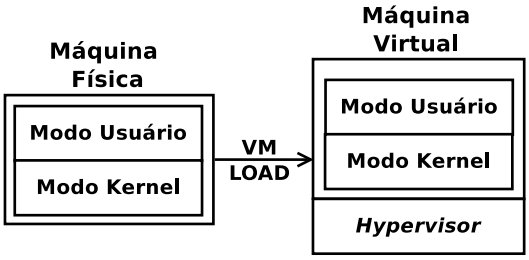


Figura 3.7. Migração do sistema para máquinas virtuais. Quando operando em uma máquina virtual, tanto o modo *kernel* quando o modo usuário ficam sobre supervisão de um *hypervisor*.

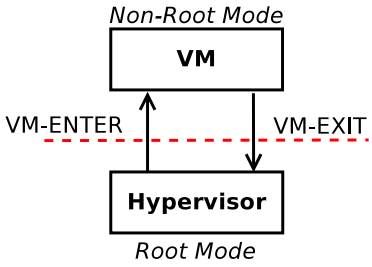


Figura 3.8. Novos modos de operação. A instrução `vmload` inicia a operação da máquina virtual. Quando uma ação monitorada ocorre no sistema *guest*, uma saída para o *hypervisor* é causada.

HVMs apresentam significativa diferença em relação a VMs tradicionais quanto ao gerenciamento de memória. Enquanto em VMs tradicionais a memória física do sistema *guest* é diretamente traduzida para memória física do *host*, HVMs apresentam um mecanismo de tradução dupla, que resulta em uma mapeamento da memória física do *guest* em memória virtual do *host*, como mostrado na Figura 3.9. O mecanismo de tradução implementado pelo processador pode também ser instrumentado, possibilitando o desenvolvimento de mecanismos de análise de memória. Complementarmente, HVMs virtualizam em *hardware* as funções de I/O através da IOMMU (*Input Output Memory Management Unity*), o que permite monitorar, também, os demais acessos do sistema, tais como acessos diretos à memória (DMA).

O carregamento de HVMs se dá através de *drivers* de *kernel* que configuram os registradores em *hardware* para a operação no modo de máquina virtual. Esta característica permite que sistemas em execução sejam transformados em VMs em tempo de execução (*live loading*), bastando ao *hardware* copiar todos os registradores da máquina

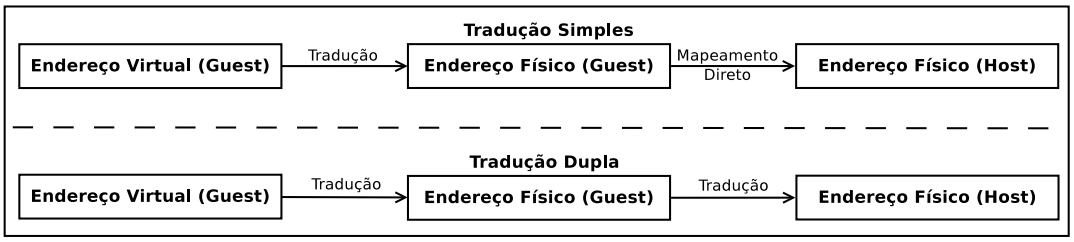


Figura 3.9. Mecanismos de tradução de memória de um HVM. Enquanto máquinas virtuais por *software* apresentam apenas uma etapa de tradução, no *guest*, HVMs apresentam duas etapas, incluindo uma etapa adicional no *hypervisor*. Este mecanismo adicional de tradução pode ser instrumentado de modo a permitir o monitoramento de memória.

física para o contexto da máquina virtual. A Figura 3.10 ilustra o bloco de controle (*Virtual Machine Control Structure*—VMCS) das máquinas virtuais tal qual definido pela Intel [Intel 2013]. Para fins de monitoração, este bloco de controle deve ser preenchido com as ações que causem saídas do *hypervisor*. O código 3.1 ilustra este preenchimento pela solução de virtualização `Xen` [Project 2015].

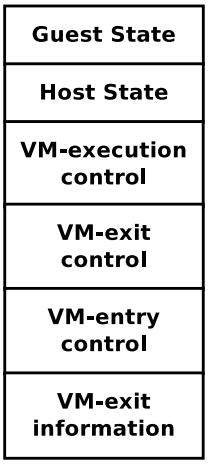


Figura 3.10. Estrutura de Controle da máquina virtual. Cada bloco da estrutura armazena informações relativas a diferentes aspectos da operação. Por exemplo, o *guest state* armazena informações dos registradores enquanto o *exit control* define os eventos que serão monitorados.

```
1  int vmx_init_vmcs_config(){
2  min = (CPU_BASED_HLT_EXITING |
3  CPU_BASED_VIRTUAL_INTR_PENDING |
4  CPU_BASED_CR8_LOAD_EXITING |
5  CPU_BASED_CR8_STORE_EXITING |
6  CPU_BASED_INVLPG_EXITING |
7  CPU_BASED_CR3_LOAD_EXITING |
8  CPU_BASED_CR3_STORE_EXITING |
9  CPU_BASED_MONITOR_EXITING |
10 CPU_BASED_MWAIT_EXITING |
11 CPU_BASED_MOV_DR_EXITING |
12 CPU_BASED_ACTIVATE_IO_BITMAP |
13 CPU_BASED_USE_TSC_OFFSETING |
14 CPU_BASED_RDTSC_EXITING);
```

Código 3.1. Código do Xen. *Virtual Machine Control Structure* (VMCS) deve ser configurado com os eventos que causaram saídas do *hypervisor* e, portanto, permitirão a monitoração.

3.3.2. Modo SMM

O modo de gerência da BIOS (*System Management Mode*) é um modo de operação do processador voltado a tarefas de gerência, como controle de energia e de temperatura. O código que opera em modo SMM é residente na BIOS, não sendo acessível a partir de outros modos de operação do processador. Entradas no modo SMM são possíveis apenas

através de interrupções especiais do modo SMM denominadas SMI. SMIs podem ser geradas através de escritas em pinos especiais do processador, por dispositivos PCI ou pela redireção de interrupções do *chipset* para a BIOS. A Figura 3.11 ilustra a possibilidade de se configurar as interrupções dos contadores de *performance* para serem entregues como SMIs. Neste caso, eles seriam tratadas por um tratador de interrupções tal qual mostrado no Código 3.2.

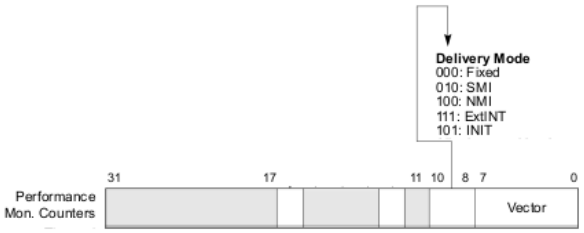


Figura 3.11. Geração de SMIs. Interrupções de diversos tipos, como as geradas pelos contadores de *performance*, podem ser entregues via SMIs e tratadas pelo modo SMM da BIOS.

```
1 void smi_handler(u32
      smm_revision) {
2 unsigned int node;
3 smm_state_save_area_t
      state_save;
4 node=nodeid();
```

Código 3.2. Código do Coreboot. Interrupções do tipo SMI tem um tratador especial dentro da BIOS. Este pode ser instrumentado para obter a causa do evento (node) e realizar tarefas de monitoração e imposição de políticas de segurança.

Como o modo SMM não é endereçado pelos demais modos de operação, este é protegido de subversão, tornando-se atraente para a implementação de soluções de análise de artefatos maliciosos. Além disso, tal qual HVMs, o modo SMM permite a execução de código no processador nativo, sendo livre de efeitos colaterais de execução. Como distinção do modo HVM, que apresenta tradução dupla de endereços em *hardware*, o modo SMM é capaz apenas de endereçar memória física. Além disso, as interrupções SMIs não entregam diretamente seu motivo causador, requerindo, portanto, procedimentos de introspecção mais complexos para a identificação dos eventos. A operação no modo SMM também é mais exigente quanto a implementação, já que a BIOS não fornece qualquer suporte prévio para extensões. Desta forma, recursos típicos de soluções de análise, como protocolos de comunicação e funções de *hash* devem ser inteiramente implementados pela solução de análise. As soluções mais populares para a reescrita de BIOS são Coreboot [CoreBoot 2015] e SeaBios [SeaBIOS 2015].

3.3.3. Modo ME

O modo ME (*Management Engine*) é um dos modos de gerenciamento de sistema presente nas arquiteturas Intel, sendo, também, responsável por controles de temperatura e energia. Ao contrário dos modos HVM e SMM, que são implementados pela CPU, o modo ME é implementado pelo *chipset*, podendo, portanto, controlar o processador como um todo, independente do modo de operação deste (*kernel*, HVM ou SMM). O modo ME opera autonomamente e conta com suas próprias unidades de processamento (ALU), memória (RAM/ROM), *timer* e DMA. Similar ao modo ME, plataformas AMD apresentam um modo de controle denominado *Secure Processor* [AMD 2016].

3.3.4. Contadores de *performance*

Hardware Performance Counters (HPCs) são recursos de *hardware* internos ao processador que registram a ocorrência de eventos no sistema, desde instruções executadas a acessos de memória. A operação dos contadores ocorre em paralelo a execução de instruções no processador e, por isso, esta não impacta significativamente o desempenho do sistema. Por isso, os contadores de *performance* são frequentemente utilizados em soluções de análise em tempo real. As soluções baseadas em contadores de *performance* podem ser classificados em dois modos de operação: i) por amostragem; e ii) por evento.

Contadores com operação por amostragem, tais como o Intel PEBS (*Precise Event Based Sampling*), contam a quantidade de ocorrências de um dado evento em um determinado período de tempo. A coleta sucessiva destes dados em uma série temporal permite construir um perfil da operação normal do sistema e depois compará-lo com dados obtidos em tempo real, detectando, assim, comportamentos anômalos.

Contadores com operação orientada a eventos, tais como os monitores de *branch* Intel *Last Branch Record* (LBR) e *Branch Trace Store* (BTS) e o monitor de sistema *Processor Tracer* (PT), registram a ocorrência de eventos individuais e informações sobre estes. Os monitores de *branch*, por exemplo, são capazes de registrar endereços de origem e destino das instruções de desvio de fluxo. Estes endereços podem ser utilizados, por exemplo, para verificar a integridade do fluxo de controle da execução.

Independente do modo de coleta (por amostragem ou por evento), os contadores operam de forma similar. Os eventos são armazenados em registradores especiais do processador (*Model Specific Registers*—MSRs) ou em páginas de memória do sistema operacional e uma interrupção é gerada quando um *threshold* de armazenamento pré-estabelecido (em número de eventos) é atingido, como mostrado na Figura 3.12. Destaca-se que a coleta de dados pelos contadores ocorre de maneira global no sistema, não havendo separação dos dados por processos, requerindo, portanto, procedimentos específicos e de introspecção para esta tarefa [Botacin et al. 2018a]. Apesar disto, os contadores de *performance* permitem filtrar os tipos de eventos monitorados e o nível de ocorrência, como mostrado na Figura 3.13. Este tipo de filtragem permite, por exemplo, monitorar apenas aplicações em modo usuário ou apenas o *kernel* do sistema.

3.3.5. Enclaves Isolados

A possibilidade trazida pelos modos HVM, SMM e ME/AMT de se monitorar o sistema como um todo exigiu o desenvolvimento de ambientes específicos para a execução de aplicações que requerem fortes garantias de segurança, como a gerência de chaves criptográficas, visto que estes modos poderiam ser utilizados por atacantes para subverter a operação deste tipo de aplicação e assim, por exemplo, vazar chaves criptográficas e segredos, comprometendo a segurança do sistema.

Considerando este cenário, enclaves isolados foram propostos, tais como o Intel *Software Guard Extensions* (SGX) [Aumasson and Merino 2016, Mandt et al. 2016] e o ARM TrustZone [ARM 2009]. Nestes ambientes, as páginas de memória são isoladas pelo controlador de memória em *hardware* (*Memory Management Unity*—MMU), criptografadas durante as trocas de contexto e destruídas após sua utilização, de modo que

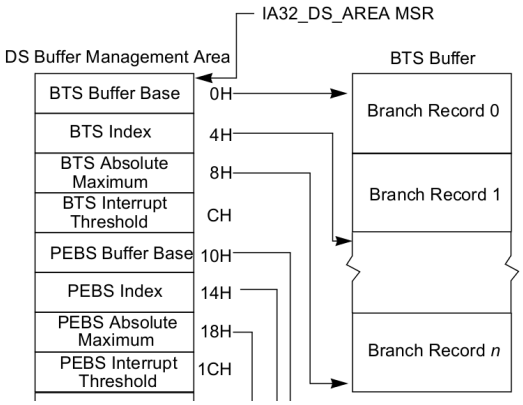


Figura 3.12. Configuração do monitor de *branch*. No monitor BTS, os dados são armazenados em páginas de memória. Quando o armazenamento atinge um *threshold* pré-estabelecido, uma interrupção é gerada. Fonte: Manual da Intel [Intel 2013].

Bit Field	Bit Offset	Access	Description
CPL_EQ_0	0	R/W	When set, do not capture branches occurring in ring 0
CPL_NEQ_0	1	R/W	When set, do not capture branches occurring in ring >0
JCC	2	R/W	When set, do not capture conditional branches
NEAR_REL_CALL	3	R/W	When set, do not capture near relative calls
NEAR_IND_CALL	4	R/W	When set, do not capture near indirect calls
NEAR_RET	5	R/W	When set, do not capture near returns
NEAR_IND_JMP	6	R/W	When set, do not capture near indirect jumps
NEAR_REL_JMP	7	R/W	When set, do not capture near relative jumps
FAR_BRANCH	8	R/W	When set, do not capture far branches
Reserved	63:9		Must be zero

Figura 3.13. Filtragem de eventos dos contadores de *performance*. Os contadores de *performance*, dentre os quais os monitores de *branch*, podem filtrar os eventos monitorados por tipo (*branches* diretos, indiretos, chamadas de função) ou por nível de ocorrência (*kernel* ou modo usuário). Fonte: Manual da Intel [Intel 2013].

outras aplicações não possam interferir com a aplicação em execução dentro do enclave, tal qual ilustrado pela Figura 3.14.

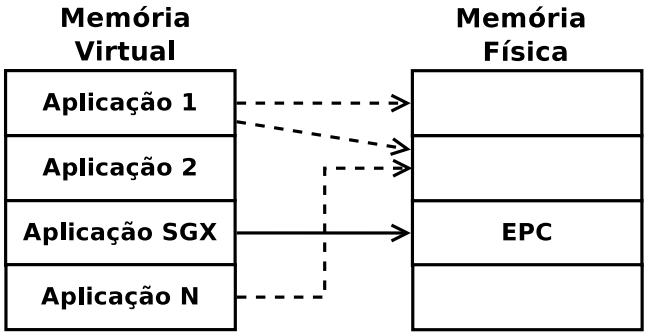


Figura 3.14. Proteção de memória dos enclaves SGX. O gerenciamento de memória dos enclaves (MMU e TLB) impede que outras aplicações mapeiem, direta ou indiretamente, a memória alocada para o enclave (*Enclave Page Cache*—EPC).

Para mover aplicações para dentro de um enclave SGX isolado, as aplicações precisam ser recompiladas de modo a suportar as instruções SGX, o que impede seu uso em aplicações legadas. Este capítulo visa discutir os impactos de segurança trazidos pela introdução dos enclaves isolados. Para informações sobre a programação neste ambiente, consulte [Will et al. 2017].

3.3.6. Recursos de *Hardware*

Além de modos de operação específicos, recursos individuais de *hardware* originalmente projetados para outros propósitos podem ser utilizados para auxiliar mecanismos de análise. Placas de expansão PCI, de um modo geral, e, em particular, GPUs (*Graphic Processing Unity*), possuem acesso direto a memória (*Direct Memory Access*—DMA), o que lhes permite mapear a memória do sistema como um todo, tanto para leitura quanto para escrita, como mostrado na Figura 3.15.

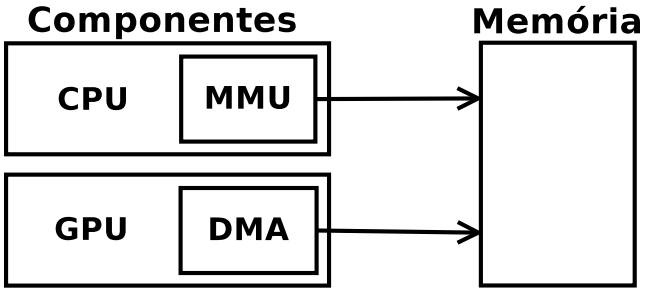


Figura 3.15. Acesso direto à memória. Este recurso permite que placas PCI, tais como GPU, acessem a memória diretamente. Como o acesso não é protegido pela MMU ou outro componente, os dispositivos PCI podem mapear a memória do sistema como um todo, incluindo as mesmas regiões acessadas pela CPU. A enumeração da memória como um todo permite, por exemplo, o *dump* para fins de forense.

O acesso direto a memória foi originalmente projetado para permitir a leitura e es-

crita dos *buffers* das placas de modo assíncrono, sem penalizar o desempenho do sistema. No entanto, por ser capaz de enumerar a memória de forma completa e sem restrições por parte da MMU ou outro componente, este recurso pode, também, ser utilizado, por exemplo, para fazer um *dump* completo da memória do sistema, um recurso desejável para muitos procedimentos forenses.

3.3.7. *Hardware* Externo

Em cenários onde o suporte arquitetural é limitado, o uso de *hardware* externo é uma alternativa para a implementação de mecanismos de segurança. Tipicamente, este tipo de abordagem é implementada por co-processadores responsáveis por realizar o monitoramento (*snooping*) do tráfego no barramento de memória compartilhado entre o processador monitor e o processador monitorado, como mostrado na Figura 3.16.

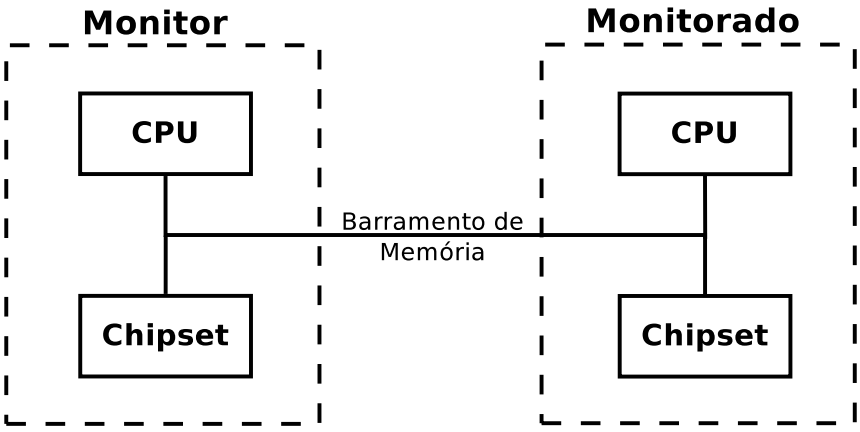


Figura 3.16. Monitoração com *hardware* externo. Dispositivos externos podem ser conectados a um barramento de memória compartilhado com a plataforma monitorada para realizar *snooping* dos dados.

Por estar posicionado externamente ao sistema monitorado, este tipo de solução não dispõe de informações de registradores, de modo que a reconstrução de contexto deve ser feita por procedimentos de introspecção a partir apenas dos dados em memória, uma tarefa significativamente mais complexa.

3.4. Técnicas de Monitoração

Embora as arquiteturas modernas apresentem diversos recursos que podem suportar procedimentos de análise, a efetiva aplicação destes pelas soluções requer o uso de técnicas de monitoração específicas. Nesta seção, apresentamos as técnicas mais frequentemente utilizadas.

3.4.1. Monitoração de eventos

Como as soluções de análise modernas são orientadas a eventos, a implementação da notificação da ocorrência dos eventos é uma etapa de desenvolvimento fundamental. Neste sentido, soluções HVM proveem um modo nativo de identificar a ocorrência de eventos, dado que cada `VM-EXIT` especifica a causa de sua ocorrência, como mudanças nos

valores do registrador `CR3`, interrupções geradas, escritas em regiões de memória monitoradas, entre outras. O modo SMM, por sua vez, não possui mecanismos nativos de notificação de eventos. No entanto, mecanismos de notificação podem ser implementado através do redirecionamento de portas do *chipset* e do controlador de interrupções (*Advanced Programmable Interrupt Controller*—APIC) para que eventos gerem interrupções do modo SMM (SMIs), disparando tratadores de eventos dentro da BIOS instrumentada. A notificação de eventos através de interrupções também ocorre para contadores de *performance*. Neste caso, contudo, interrupções ordinárias, e não SMIs, são geradas por padrão. Quando da ocorrência da interrupção, uma rotina de tratamento a nível de *kernel* (*Interrupt Service Routine*—ISR) é responsável por coletar os valores armazenados nos registradores ou nas páginas de memória do sistema. Os demais mecanismos de monitoramento, tais como DMA, são passivos e, portanto, não geram interrupções.

3.4.2. Callbacks

Além do disparo de rotinas de inspeção quando da ocorrência de eventos, pode-se também executar código de monitoração e verificação em meio as instruções do programa analisado, através de mecanismos de *callbacks*. O uso de *callbacks* por mecanismos de análise está principalmente associado a soluções baseadas no modo SMM. Soluções SMM inserem *callbacks* em instruções, por exemplo, para superar as barreiras de endereçamento de memória física. Além disso, *callbacks* são utilizadas pelo modo SMM para causar *VM-EXITs* de um *hypervisor* em modo *root*, possibilitando, assim, a inspeção deste.

3.4.3. Monitoração de memória

O monitoramento de memória é uma tarefa de inspeção essencial, dado que os valores em memória permitem reconstruir o contexto da execução. De um modo geral, a monitoração de memória é implementada através da geração proposital de *page faults*. Nesta técnica, uma ou mais páginas são marcadas como não presentes ou sem a requerida permissão (de execução, leitura ou escrita), causando, portanto, um *page fault* durante uma tentativa de acesso. O sistema de gerenciamento de memória é instrumentado de modo a registrar o evento (*logging*) e restaurar a permissão retirada, permitindo, assim, que a execução prossiga. A escolha de qual permissão será retirada determina o tipo de evento monitorado. Uma política frequentemente imposta é a chamada `Write Xor Execute` [Roemer et al. 2012], que garante que páginas de código (executáveis) não podem ser escritas e/ou que páginas escritas (dados) não sejam executáveis. Desta forma, as tentativas de se escrever em páginas executáveis ou de se executar páginas com permissão de escrita resultam na violação da política, com respectiva invocação do tratador de *page faults*.

A sucessiva escrita e execução de código é observada, por exemplo, em ataques de *buffer overflow*, no qual código externo é injetado e executado a partir da pilha. Sistemas modernos impõem a política de pilha não executável, como mostrado no Código 3.3.

```
1 cat /proc/self/maps
2 00400000-0040c000 r-xp 00000000 /bin/cat
3 7f0bef204000-7f0bef3c4000 r-xp  libc-2.23.so
4 7ffe3a213000-7ffe3a234000 rw-p  [stack]
```

Código 3.3. Pilha não executável. Plataformas modernas não permitem que a pilha seja executada para evitar ataques de injeção de código. Note que apenas atributos de leitura e escrita (rw-) são atribuídos a pilha (*stack*).

A escrita e execução de páginas de código também é observada em exemplares de *malware* do tipo auto-modificável (*Self Modifying Code*—SMC) [Botacin et al. 2018a, Cai et al. 2007]. Neste tipo de exemplar, o código original se auto-modifica de modo a se tornar malicioso. Esta estratégia é empregada para que o código original possa sobreviver a análises por soluções de segurança, como por antivírus. Um exemplo de código auto-modificável é apresentado na Figura 3.17.

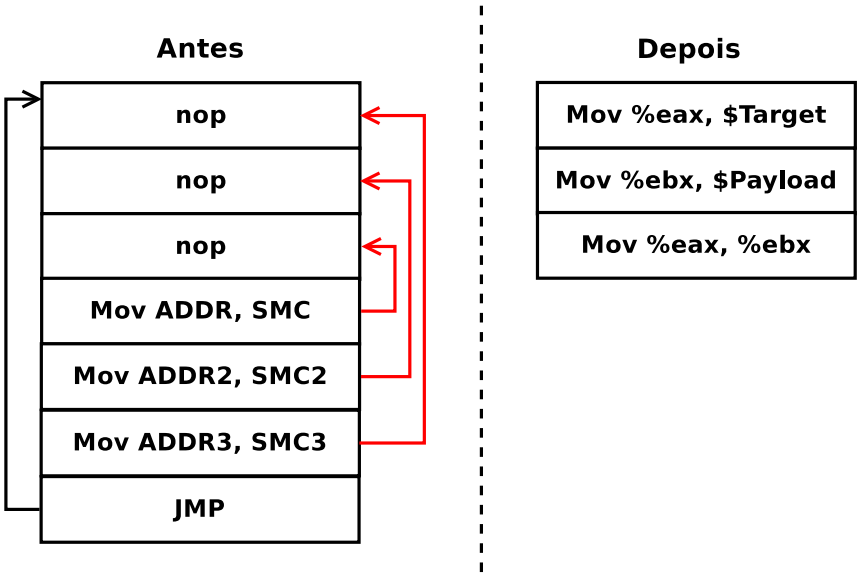


Figura 3.17. Código Auto-Modificável (SMC). Durante sua execução, o código altera suas próprias instruções (`Mov SMC`), tornando as instruções `NOP`, em instruções `MOV` de propósito malicioso. Após as modificações, o código salta (`JMP`) para poder executar as instruções modificadas.

Durante o tratamento do *page fault*, os dados presentes na página podem ser alterados pela solução de análise, o que é um requisito de soluções de *debugging*. Contudo, a forma usual de se implementar modificações nas páginas de memória é através de *shadow copies*. Esta técnica consiste em duplicar o conteúdo das páginas acessadas de modo a poder comparar a versão original e a versão modificada. Uma sequencia de *shadow copies* permite, por exemplo, realizar o *unpack* de códigos maliciosos. A cópia de valores de memória é uma tarefa constante durante processos de monitoração. Procedimentos forenses, por exemplo, realizam o *dump* da memória para análise *offline*. Verificadores de sistema, por sua vez, inspecionam a memória periodicamente para calcular sua integridade. Contudo, copiar a memória de forma completa é um procedimento custoso,

tanto em armazenamento quanto em tempo de processamento, o que requer abordagens alternativas. A técnica usual para este tipo de tarefa consiste em se realizar um primeiro *snapshot* completo do sistema e posteriormente acessar apenas as páginas modificadas. Os recursos arquiteturais como os controladores de memória permitem marcar as páginas modificadas e realizar a cópia apenas destas. Mais especificamente, pode-se postergar a cópia dos dados até que uma requisição para que estes sejam modificados seja feita, uma abordagem conhecida como *dump on write*, em referência a política *copy on write* dos sistemas Unix [Thober et al. 2008]. Este tipo de cópia de memória é ilustrado na Figura 3.18.

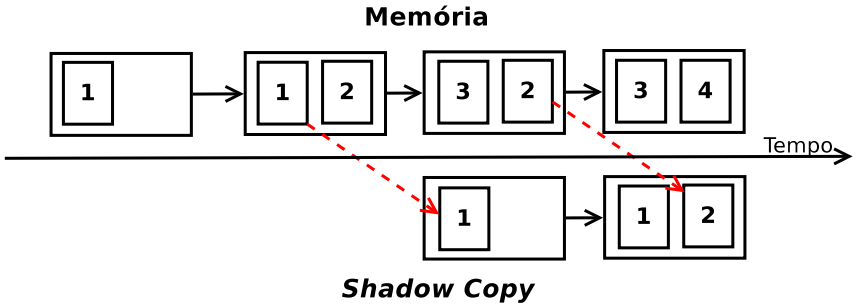


Figura 3.18. *Shadow Copies).* A duplicação de memória pode ser implementada de uma forma sob demanda, coletando os dados apenas quando estes são modificados, uma política denominada *dump-on-write*.

Enquanto as técnicas de manipulação de permissões e de *shadow copies* não são inéditas por si só, o uso destas foi impulsionado pelos novos recursos de MMU presentes nas arquiteturas modernas, notavelmente em HVMs.

3.4.4. Execução em pequenos passos

A execução em pequenos passos é uma tarefa de inspeção importante pois permite ao analista observar regiões de código específicas, com controle de granularidade. De um modo geral, o controle do avanço da execução em plataformas modernas pode ser implementado de duas maneiras: i) pelo controle das permissões das páginas de código; ou ii) por interrupções dos contadores de eventos. No primeiro caso, as páginas de código são marcadas como não-presentes e/ou não-executáveis, causando *page-faults* quando da tentativa de execução, requerindo que o tratador de eventos restaure a permissão da página para que a execução prossiga. Ao se repetir este procedimento para cada instrução, pode-se implementar o avanço de forma *step-by-step*, tal qual requerido por *debuggers*. No caso dos contadores de eventos, estes são pré-configurados em seus valores máximos, causando um *overflow* a cada instrução executada, dado que um novo evento de *hardware* é gerado. O tratador de interrupções é responsável por tratar o evento e novamente definir o contador para o valor máximo permitido, o que gerará uma nova interrupção quando da execução da instrução seguinte. Esta estratégia também permite o avanço *step-by-step*.

3.4.5. Tratamento de instruções individuais

Uma desvantagem das técnicas de interceptação baseadas em *faults* e *overflows* é que o tratador de eventos não tem acesso imediato a instrução que gerou o evento. O tratador

de *page faults*, por exemplo, tem acesso apenas ao endereço base da página. Para tratar estes casos, estes tratadores devem operar em conjunto com outros mecanismos, como os monitores de *branch*. Desta forma, quando o tratador de eventos é invocado, este deve olhar o topo da pilha de *branches* para identificar o último bloco de instruções executado.

3.4.6. Pontos de parada

Breakpoints são extensões naturais da execução em múltiplos passos, permitindo ao analista interromper a execução em pontos de interesse específicos. Processadores tradicionalmente apresentam suporte para dois tipos de *breakpoints*: *hardware breakpoints* e *software breakpoints*. *Hardware breakpoints* são registradores nos quais o analista pode inserir endereços de instruções para que o processador interrompa a execução quando o apontador de instruções (*Instruction Pointer*—IP) corresponder a um endereço de um dos registradores. Por requerir o armazenamento em registradores físicos, o número de *hardware breakpoints* é naturalmente limitado. Além disto, estes são compartilhados entre o sistema *host* e o sistema *guest*. *Software Breakpoints*, por sua vez, são ilimitados, mas introduzem efeitos colaterais de execução, pois requerem que os primeiros *bytes* da instrução monitorada sejam substituídos por uma *trap flag* (`int 3` ou `0xCC`), que faz com que o processador interrompa a execução quando esta é identificada.

A necessidade de se prover suporte a um grande número de pontos de parada e, ao mesmo tempo, não impor efeitos colaterais de execução, fez com que as soluções de análise modernas se baseassem em outras formas de se interromper a execução. De um modo geral, duas técnicas são frequentemente implementadas: i) execução por avanço-e-comparação; e ii) pontos de paradas invisíveis. No primeiro caso, as técnicas de avanço previamente apresentadas (*page fault trapping* e *performance counter overflows*) são empregadas de modo que o endereço do apontador de instruções seja comparado com uma lista de endereços de parada a cada etapa, consistindo, portanto, em uma implementação mista entre *hardware* e *software*. No segundo caso, *software breakpoints* tradicionais são utilizados, mas os efeitos colaterais são mascarados por tratadores de exceção. Por exemplo, a instrução `PUSHF` precisa ser interceptada para que a *trap flag* seja removida. Caso contrário, um programa monitorado poderia ler suas próprias *flags* e identificar que este está sob monitoração.

3.4.7. Filtragem de granularidade

Se, por um lado, a visão do sistema como um todo (*system-wide view*) permite ao analista trabalhar com modelos de ameaça abrangentes, como os que incluem o *kernel* ou o *hypervisor*, por outro, a quantidade de dados coletados pode dificultar o trabalho de inspeção. Por isso, filtros são empregados de modo a isolar eventos e regiões de interesse.

Uma escolha de granularidade frequente por parte dos analistas é quão profundo será o mergulho dentro das chamadas de funções. Uma visão de todo do sistema permite mergulhar recursivamente dentro das chamadas que implementam uma dada função de alto nível. Este tipo de operação é denominado *step-into* e é exemplificado pela Figura 3.19.

Podemos observar que, neste modo, as funções internas à função `printf` também são monitoradas, permitindo ao analista compreender a forma como a função é im-

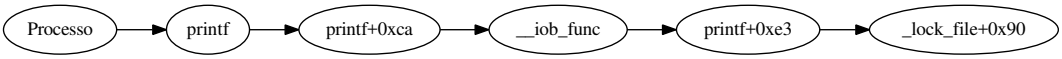


Figura 3.19. *Step-Into.* Quando a análise ocorre neste nível de granularidade, o analista mergulha em cada chamada de função, incluindo as funções auxiliares dentro da função principal sendo analisada. No exemplo, pode se observar os *locks* usados para garantir a sincronia da função `printf`.

plementada, o que é usual em processos de depuração e engenharia reversa. Identifica-se, por exemplo, o uso de *locks* para acesso exclusivo, dado que a função não é reentrante.

No entanto, quando o analista está interessado em informações mais gerais, como identificar o comportamento de um *malware* através das funções chamadas, este pode limitar o escopo da análise as chamadas de mais alto nível, desconsiderando as chamadas internas, o que é conhecido como modo *step-over*, como representado na Figura 3.20.

Figura 3.20. *Step-Over.* Quando a análise ocorre neste nível de granularidade, o analista está interessado nas funções de mais alto nível e, portanto, mais externas, simplificando o procedimento de análise. No exemplo, as chamadas internas das funções não são consideradas.

Neste exemplo, as chamadas internas das funções `printf` e `scanf` são omitidas, de modo que o analista pode identificar o comportamento geral da aplicação (ler um dado e imprimi-lo).

Dado o *gap* semântico, filtros de alto nível não são nativos do *hardware* e devem ser implementados pela solução de análise. De um modo geral, mecanismos de filtragem se baseiam em *breakpoints* condicionais controlados por *software*. Neste tipo de *breakpoint*, a parada da execução em um dado ponto acorre apenas quando uma condição de análise definida pelo analista é satisfeita, como, por exemplo, quando um dado registrador ou valor de memória assume o valor zero.

3.4.8. Mitigação de evasão por medidas de tempo

Apesar de não requerir injeções de código nem causar efeitos colaterais de execução, a análise de objetos em soluções de *hardware* ainda pode ser identificada por medidas de tempo. Notavelmente no caso de HVMs, o registrador *TimeStamp Counter* (TSC) é incrementado tanto para as instruções do *guest* quanto do *host*, o que permitiria a um *guest* malicioso identificar as instruções executadas pelo *hypervisor* instrumentado. Para lidar com este cenário, as soluções de análise modernas adulteram o valor do registrador TSC para omitir o tempo gasto pelo processamento no *hypervisor*. Por esta razão, medidas de tempo tomadas em máquinas virtuais, de um modo geral, não são confiáveis.

3.4.9. Mitigação de *fingerprinting*

Uma maneira de se detectar um ambiente de análise é localizar identificadores únicos conhecidos, como o nome de soluções de análise ou caminhos de sistema de soluções de segurança. De um modo geral, soluções de análise modernas empregam técnicas para evi-

tar este tipo de evasão. Em particular, soluções baseadas em HVM utilizam-se de técnicas de *rootkits* para esconder seus *drivers* de carregamento das APIs do sistema operacional. Soluções baseadas em SMM, por sua vez, geram SMIs baseadas em um *timer* periódico para executar rotinas de verificação, dentre as quais, se os registradores de *debug* foram modificados.

3.5. Aplicações

Nesta seção, apresentamos diversas aplicações de análise que podem se beneficiar dos recursos arquiteturais e das técnicas de implementação apresentadas.

3.5.1. Detecção de *bugs*

Defeitos de *software* (*bugs*) são a raíz de muitos problemas de segurança, o que torna a detecção destes tarefa fundamental. Embora diferentes soluções de monitoração possam ser aplicadas à esta tarefa, as soluções baseadas em monitores de *branch* são as mais adequadas, visto que a detecção de *bugs* exige apenas a coleta de dados relativos aos caminhos percorridos pela execução.

Programas podem ser entendidos como máquinas de estados, cujas transições são as instruções de *branch*. Um código compilado apresenta diversas instruções de *branch*, como apresentado no Código 3.4, de modo que os caminhos formados por estas representam diferentes estados. Um *bug* em um programa pode ser entendido como um estado inesperado, atingido através de uma sequência de *branches*. Portanto, identificar o *branch* origem do caminho até o estado inesperado permite identificar a causa do *bug*, como mostrado na Figura 3.21.

```
1 je      0x4b9cf6   0x4b9c40
2 jne     0x4b9c46   0x4b9bd1
3 jle     0x4b9bd3   0x4b9c88
4 jne     0x4b9c9c   0x4b9c88
5 je      0x4b9c9c   0x4b9c74
6 jne     0x4b9c7a   0x4b9cd8
7 je      0x4b9ce8   0x4b9ad0
8 je      0x4b9ae3   0x4b9b08
```

Código 3.4. *Disassembly*. Endereços (origem de destino) das instruções de controle de fluxo de uma aplicação que apresenta um *bug*.

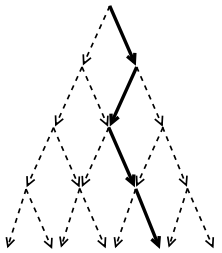


Figura 3.21. Árvore de decisão. O acompanhamento dos *branches* tomados permite identificar o caminho percorrido e, assim, identificar em qual ponto a execução diverge do esperado.

De um modo geral, a identificação de *bugs* pode ocorrer em três diferentes estágios do ciclo de vida de uma aplicação: i) durante o desenvolvimento; ii) durante o *deployment*; e iii) análises *post-mortem*.

Análises em tempo de desenvolvimento consistem em exercitar os diferentes caminhos de execução possíveis, em um procedimento denominado *fuzzing* [Felderer et al. 2016]. Para mapear os diferentes caminhos tomados, monitores de *branch* como o BTS podem ser empregados [Paleari 2015].

Análises de aplicações *deployed* consistem em monitorar o caminho de execução tomado pela aplicação quando exercitada com uma entrada que leve a uma falha. A monitoração dos caminhos visa identificar qual bloco de decisão leva a falha para sua posterior correção. Estratégias de *root-cause-analysis* podem, também, ser implementadas utilizando-se de monitores de *branch* como o LBR [Arulraj et al. 2014].

Análises *post-mortem* consistem em coletar dados de execução e submetê-los aos desenvolvedores na forma de *crash reports*. As informações de caminhos tomados podem ser enviadas para auxiliar na localização da falha de implementação. Esta abordagem pode ser implementada, por exemplo, através do uso do monitor `Processor Tracer` [Xu et al. 2017].

3.5.2. Análise de *Malware*

HVM e SMM são candidatos naturais para a implementação de soluções de análise de *malware*, pois eles permitem a execução do exemplares malicioso de forma transparente e livre de injeção de código. Além disso, a base de código confiável destes permite modelos de ameaça amplos, cobrindo, por exemplo, a análise de *rootkits* de *kernel*.

Diversas soluções baseadas em HVM foram propostas para a tarefa de análise de *malware* [Dinaburg et al. 2008b, Willems et al. 2012b, Nguyen et al. 2009]. De um modo geral, o recurso de *page fault trapping* é estendido para todo o sistema, de modo que qualquer parte deste tocada pelo exemplar de *malware* cause um `VM-EXIT`, a ser tratado pelo *hypervisor*. Este, por sua vez, é instrumentado de modo a implementar a solução de análise em si.

Uma desvantagem significativa do uso de HVM é o impacto no desempenho imposto pelas sucessivas saídas do *hypervisor*. Como alternativa, soluções de *replay* podem ser implementadas [Yan et al. 2012]. Neste tipo de abordagem, apenas a coleta dos dados de execução dos exemplares é realizada em um ambiente HVM, enquanto os procedimentos de análises são realizados em etapas posteriores, que podem ser implementados em *software*. Desta forma, obtém-se tanto as vantagens da execução em ambiente HVM—a mitigação das tentativas de evasão por parte do exemplar de *malware*—quanto a flexibilidade de uma solução em *software*. Este tipo de abordagem permite, por exemplo, repetir as instruções executadas pelo exemplar de *malware* em uma solução HVM em uma solução de *software*, de modo a identificar padrões maliciosos em sua execução.

O impacto no desempenho pode também ser mitigado pela adoção de tecnologias de coleta de dados mais leves, como os monitores de *branch* [Willems et al. 2012a, Botacin et al. 2018a]. Neste caso, contudo, restrições ao modelo de ameaças são impostas. Como os monitores requerem que o *kernel* trate as interrupções geradas por estes, este deve ser incluído na base de código confiável.

3.5.3. *Debugging*

Mais do que traçar o comportamento de programas, os recursos arquiteturais das plataformas modernas permitem também interromper a execução do objeto monitorado em pontos de interesse e alterar o contexto deste (incluindo valores em memória, em registradores e até mesmo o apontador de instruções), o que os torna adequados para a implementação de ferramentas de *debugging*.

Soluções baseadas em HVM [Fattori et al. 2010] são capazes de prover *stepping* de instruções, definição de pontos de paradas e acesso a registradores e memória através de `VM-EXITs`. Além disso, a capacidade de atuar no sistema como um todo possibilita inclusive o *debugging* do *kernel* do sistema em execução. Por esta mesma característica, soluções de *debugging* baseadas em SMM também foram propostas [Zhang et al. 2015]. Os recursos de inspeção providos por esta são similares aos das soluções HVM.

As interrupções geradas por contadores de *performance* também podem ser utilizadas para gerar pontos de parada para *debugging*. Contudo, seu tratamento a nível de *kernel* permite apenas que aplicações do modo usuários sejam inspecionadas. Além disso, o tipo de monitor utilizado influencia na granularidade da inspeção. O uso de monitores de *branch*, por exemplo, permite que a execução ocorra *branch-by-branch* ao invés de *step-by-step* [Botacin et al. 2018a].

3.5.4. Forense

A capacidade de executar código privilegiado também torna os recursos das arquiteturas modernas adequados para a extração de dados do sistema para fins de forense.

Soluções baseadas em HVM [Martignoni et al. 2010] são capazes de ser instanciadas em ambientes já em execução devido ao recurso *live loading*, sendo adequadas para o uso por peritos, como mostrado na Figura 3.22. Como o ambiente pode ter sido comprometido, a solução emprega um mecanismo de *lie detection* para verificar a confiança do sistema. Neste, processos em execução são obtidos através das APIs do SO e também via introspecção do *hypervisor*. O sistema é considerado comprometido se os dados em ambas as fontes divergirem.

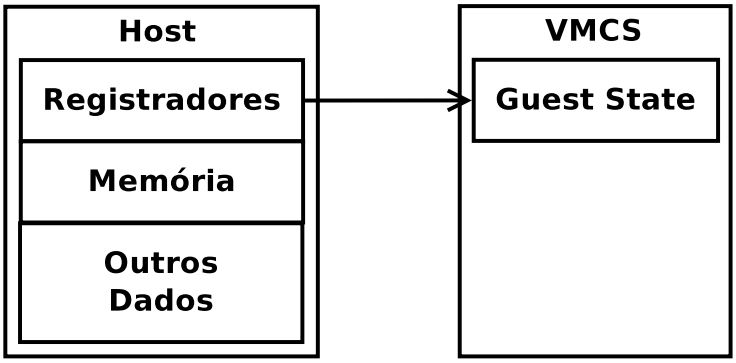


Figura 3.22. Carregamento em tempo real. Soluções HVM permitem copiar o estado corrente da máquina física para o contexto de uma máquina virtual, movendo o ambiente em tempo de execução. Este recurso permite, por exemplo, que analises forenses sejam realizadas utilizando-se da própria máquina a ser analisada.

Soluções forenses baseadas em SMM também foram propostas [Reina et al. 2012]. Como um mecanismo de *live loading* nativo não existe neste modo, a BIOS instrumentada deve ser instalada *a priori*. Como, de um modo geral, não se sabe a tarefa forense a ser requerida de antemão, usualmente se utiliza o modo SMM para realizar o *dump* total da memoria do sistema para que as análises sejam realizadas posteriormente.

Uma forma leve, em termos de desempenho e de implementação, de se realizar o *dump* completo da memória do sistema é utilizar-se do acesso DMA, como proposto por algumas soluções [Wang et al. 2011]. Neste caso, o perito poderia inserir uma placa PCI no sistema e utilizar do acesso DMA para varrer a memória como um todo. Para garantir a consistência, isto é, que os dados não sejam alterados durante o processo de coleta, o sistema deve ser pausado de alguma forma.

3.5.5. Imposição de políticas de segurança

Além de permitir aos analistas inspecionarem os binários em execução, os recursos arquiteturais podem ser empregados para impor políticas de execução sobre estes em tempo real. Isto garante a organizações, por exemplo, que suas políticas de segurança sejam impostas, a despeito de recursos da aplicação ou do sistema operacional, uma vez que a imposição é implementada em camadas inferiores.

O modo HVM, por exemplo, pode ser utilizado para impor politicas de armazenamento, como proposto em Bitvisor [Shinagawa et al. 2009]. Nesta solução, todas as escritas em disco são interceptadas e criptografadas, garantindo a segurança dos dados armazenados, dado que a chave de decriptação fica armazenada no *hypervisor*, a qual o *guest* não tem acesso.

3.5.6. Detecção de ataques

Os recursos arquiteturais das plataformas modernas também podem ser utilizados para detectar ataques em tempo real, se beneficiando dos recursos de análise suportados por estes.

Soluções baseadas em HVM podem causar `VM-EXITs` quando eventos suspeitos ocorrem. A solução Secvisor [Seshadri et al. 2007], por exemplo, permite que apenas código aprovado seja executado, invocando as rotinas de inspeção do *hypervisor* quando de violações desta política. Este tipo de política impede, por exemplo, ataques de redireção do *kernel* para o modo usuário. Neste tipo de ataque, o atacante se vale dos privilégios de execução do *kernel*, através, por exemplo, de um *bug* em um *driver*, para escrever em processos de modo usuário, estejam estes sob controle do atacante ou de terceiros. Se esta escrita contemplar *flags* de proteção, o processo afetado pode ter seus mecanismos de segurança desabilitados através da atribuição de privilégios administrativos.

As capacidades do modo SMM também podem ser utilizadas para a detecção de ataques. A solução SPECTRE [Zhang et al. 2013], por exemplo, implementa verificações em memória em tempo real. Pode-se, por exemplo, aplicar expressões regulares sob os dados coletados para se detectar a injeção de *shellcodes* através de uma sequência de NOPs (*No-Operation*), uma característica de ataques do tipo *buffer overflow*, dada a necessidade de alinhamento do código na pilha.

Além de políticas de detecção de ataques de um modo geral, alguns recursos arquiteturais são particularmente interessantes para a detecção de ataques específicos, como as políticas de integridade do fluxo de controle e a detecção por efeitos colaterais. Atualmente, ataques de injeção de código utilizam da técnica denominada programação orientada a retorno (*Return Oriented Programming*—ROP), de modo a contornar políticas de restrição da execução de código externo, como `Write Xor Execute`. Neste tipo

de ataque, ao invés de injetar um *shellcode* externamente, como em ataques de *buffer overflow* tradicionais, o *shellcode* é construído em memória através do encadeamento de sequências de instruções terminadas por uma instrução de retorno (`RET`) (*gadgets*), como mostrado na Figura 3.23.

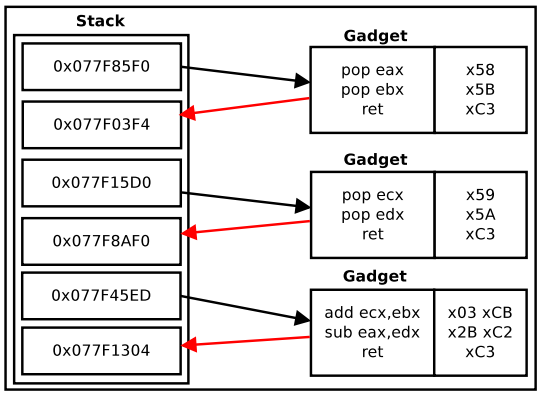


Figura 3.23. Programação Orientada a Retorno (ROP). O *payload* neste tipo de ataque é construído através do encadeamento de sequências de instruções terminadas por uma instrução de retorno (*gadgets*).

Este tipo de construção de código viola uma expectativa comum de que toda instrução de retorno seja precedida por uma instrução do tipo `CALL`, uma vez que chamadas de função seguem este padrão. Desta forma, ataques `ROP` podem ser detectados por políticas de integridade do fluxo de controle, como garantir que todo bloco de instruções terminados em `RET` seja precedido por uma instrução `CALL`, como exemplificado na Figura 3.24.

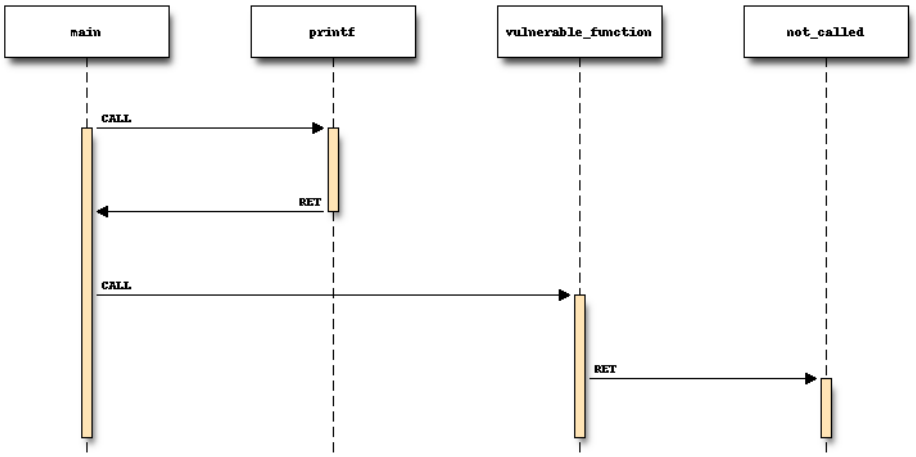


Figura 3.24. Integridade de Fluxo de Controle (CFI). Tipicamente, espera-se que instruções de retorno sejam precedidas por instruções do tipo `CALL` (como no fluxo entre *main* e *printf*). Em um ataque do tipo ROP, esta condição é violada, o que permite sua identificação quando da execução do *payload* (como no fluxo entre *vulnerable_function* e *not_called*).

Políticas de integridade do fluxo de controle (*Control Flow Integrity*—CFI) podem ser implementadas de forma eficiente através do uso de monitores de *branches*, dado que estes podem monitorar especificamente este tipo de instrução, tal qual proposto em diversas soluções [Xia et al. 2012, Pappas et al. 2013, Cheng et al. 2014]. Por monitorar apenas as instruções de desvio de fluxo, este tipo de abordagem reduz significativamente o impacto da solução de monitoração sobre o sistema em comparação com soluções baseadas em *stepping*, que necessitam verificar todas as instruções para posteriormente desconsiderar as que não estejam envolvidas na mudança do fluxo de execução.

Além de políticas explicitas, como CFI, políticas implícitas também podem ser empregadas para a detecção de ataques. A detecção de efeitos colaterais consiste em se definir um padrão normal de execução do sistema, tais como o número de instruções de *branch* executadas em um intervalo de tempo, e comparar os resultados obtidos em tempo real com este padrão normal. Desta forma, ataques `ROP` seriam detectados pela execução anômala de seguidas instruções de retorno. Contadores de *performance* são adequados para esta tarefa, tal qual proposto em [Kompalli and Sarat 2014, Bahador et al. 2014], dado que estes podem contar eventos no sistema como um todo sem a necessidade de se instrumentar cada aplicação.

3.5.7. Verificação de integridade

Além de tentar detectar ataques de forma deliberada, os recursos arquiteturais podem ser utilizados para garantir a integridade do sistema e assim prover garantias de segurança derivadas desta.

Tipicamente, soluções de verificação de integridade são implementadas com o suporte de *hardware* externo [Petroni et al. 2004, Moon et al. 2012, Lee et al. 2013]. Este tipo de solução é capaz de efetuar a leitura da memória a ser verificada via barramento compartilhado e computar um *hash* dos valores lidos, identificando se estes se alteram ao longo do tempo. A hipótese que suporta esta abordagem é que alguns componentes do sistema, como o código de *boot* ou do *kernel*, devem ser imutáveis.

Soluções HVM também podem ser utilizados para verificar a integridade de componentes do sistema, como as configurações de I/O [Zhang 2013], de modo a garantir que as portas não sejam remapeadas. O remapeamento de mecanismo de DMA poderia levar, por exemplo, a um ataque do tipo *man-in-the-middle* de memória, dando ao atacante controle sob os dados.

A depender do modelo de ameças considerado, verificações a nível de *hypervisor* podem não ser adequadas, visto que ataques aos *hypervisors* são atualmente conhecidos [Rutkowska and Wojtczuk 2008]. Neste tipo de ataque, o código em execução dentro da máquina virtual mapeia a memória física do *hypervisor* de modo a fazer com que este atribua maiores permissões ao código monitorado, como exemplificado pela Figura 3.25.

Desta forma, garantir a integridade do *hypervisor* em si é uma tarefa essencial em muitos cenários para prevenir e detectar este tipo de modificação. Para tanto, soluções em mais baixo nível são necessárias, tais como as baseadas no modo SMM [Azab et al. 2010, Wang et al. 2010]. Tal qual as abordagens em *hardware* externo, estas soluções também se baseiam em *hashes* para garantir que o código do *hypervisor* não seja alterado.

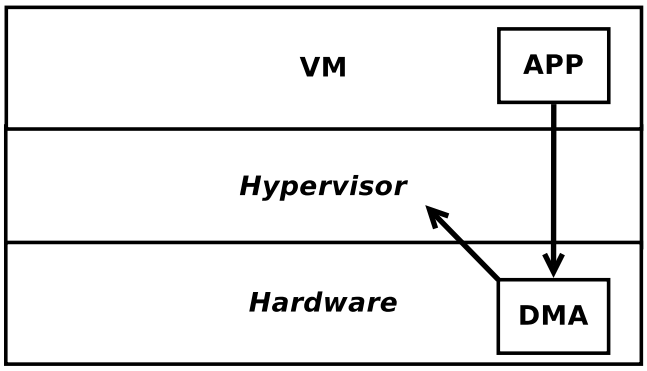


Figura 3.25. Ataques ao *Hypervisor*. A aplicação dentro da máquina virtual pode mapear a memória física do *hypevisor* e, através de acesso DMA, modificar seu conteúdo. A aplicação pode, por exemplo, alterar as permissões que o *hypervisor* atribui a sua execução, elevando, assim, seus privilégios.

3.6. Perspectivas Futuras: Ameaças e Oportunidades

Os recursos arquiteturais presentes nas arquiteturas modernas trouxeram novas possibilidades de implementação de mecanismos de análise e também novos desafios. Esta seção apresenta os problemas atualmente em aberto e perspectivas de futuros desenvolvimentos. Acreditamos que os pontos aqui apresentados podem ser abordados em futuros projetos de pesquisa.

Máquinas Virtuais de *Hardware* apresentam um enorme potencial para o desenvolvimento de soluções de segurança e por isso novas aplicações tem se baseado fortemente neste tipo de recurso. O sistema operacional Qubes [Rutkowska 2010], por exemplo, move todo o ambiente Linux para uma máquina virtual de modo a isolar os processos em execução através do *hypervisor*. Tentativas de violar o controle de acesso resultam em exceções do *hypervisor*. Estratégia similar é empregada no sistema Windows 10 quando da utilização do modo *Secure Kernel* [Ionescu 2015]. Com a consolidação desta tendência, técnicas de detecção de máquina virtual não serão suficientes para inferir um ambiente como sendo de análise. Desta forma, códigos evasivos terão de diretamente detectar que estão sob análise, utilizando-se, por exemplo, de informações do contexto da execução, o que é significativamente mais difícil. Entre os desafios para a consolidação das máquinas virtuais de *hardware* estão o suporte a *guests* aninhados (máquinas virtuais dentro de máquinas virtuais) e o desenvolvimento de processos de introspecção para este cenário.

A consolidação dos ambientes virtualizados pode, por outro lado, permitir novos recursos de segurança para as aplicações. Aplicações conscientes de sua execução em uma máquina virtual podem delegar ao *hypervisor* diversas tarefas. A delegação ocorre através de chamadas do *hypervisor* (*hypercalls*) a partir do *kernel*. *Hypercalls* podem ser entendidas como uma extensão das *syscalls* que ocorrem do modo usuário para o *kernel* (Figura 3.26). Soluções atuais, como o Xen, já fornecem algum suporte, ainda que limitado, a *hypercalls*, como mostrado no Código 3.5. Este tipo de chamada pode ser estendida para que um *kernel* notifique o *hypervisor* sobre, por exemplo, uma região de memória que deve ser protegida por este.

O **modo SMM** pode ser empregado em diversas aplicações de segurança, tendo

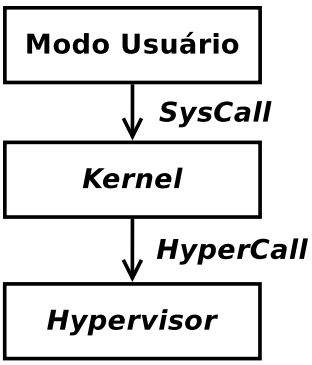


Figura 3.26. *Hypercall.* Chamadas do *hypervisor* a partir do *kernel* são correspondentes a chamadas deste pelo modo usuário.

```
1 enum hypercall_num {
2 #define __HYPERVISOR_set_callbacks
3 #define __HYPERVISOR_set_debugreg
4 #define __HYPERVISOR_get_debugreg
5 #define __HYPERVISOR_xen_version
6 #define __HYPERVISOR_vm_assist
7 #define __HYPERVISOR_callback_op
8 #define __HYPERVISOR_sysctl
9 #define __HYPERVISOR_domctl
```

Código 3.5. Código do Xen. *Hypercalls* são suportadas pelo *hypervisor*, de forma que um *kernel* ciente da sua operação em uma máquina virtual pode invocá-las para assistir sua operação.

notável sucesso na verificação de integridade de *hypervisors*, dado seu posicionamento privilegiado. Como desafio, tal qual para HVMs, procedimentos de introspecção devem ser desenvolvidos para que o modo SMM possa interpretar os conteúdos em execução dentro do *hypervisor*. Além disso, a instrumentação de BIOS não é uma tarefa simples e pode ser até mesmo impedida em algumas plataformas recentes.

A aplicação dos modos **ME/AMT** evoluiu de mecanismos de monitoração e controle do estado do processador, tais como consumo de energia e temperatura, para a monitoração da execução em si, tendo como maior vantagem possibilitar o controle total sobre o sistema. Atualmente, apenas provas de conceitos utilizando-se deste modo foram apresentadas, mas acreditamos que esta tecnologia possui potencial para ser aplicada em tarefas como *tracing* e *debugging*.

Contadores de *Performance* são uma alternativa leve para a monitoração de sistemas em tempo real. Enquanto soluções que identificam desvios de execução de um perfil do sistema definido a partir dos dados dos sensores são bem estabelecidas, avanços recentes das técnicas de aprendizados de máquina podem impulsionar novos desenvolvimentos. Abordagens leves como estas devem ser popularizadas a medida que as demandas por soluções de alta performance aumentem.

O **Acesso direto à memória** é uma abordagem poderosa para a monitoração do sistema de uma maneira ampla. Como desvantagem, o contexto limitado pela ausência de registradores dificulta o desenvolvimento de mecanismos de introspecção. Além disso, um problema em aberto relativo a DMA é a ausência de autenticação de dispositivos PCI [Intel 2018], que implementam os mecanismos DMA, o que permite a introdução de dispositivos maliciosos em um sistema. Um dispositivo PCI malicioso pode, por exemplo, realizar o *dump* de memória em busca de chaves criptográficas e outros segredos.

Dispositivos de *hardware* externos são exemplo de uma classe de aplicações pouco explorada por mecanismos de segurança. Este tipo de mecanismo apresenta significativas vantagens, como não impor nenhum *overhead* ao sistema e ser protegido de subversão pelo seu posicionamento externo. Este posicionamento, contudo, limita as

soluções ao papel de agentes passivos, sendo capazes apenas de reportar ataques mas não de bloqueá-los. A extensão destas soluções para mecanismos ativos é um problema em aberto.

Enclaves Isolados, tais como o modo SGX, surgiram para conter o vazamento de informações mesmo na presença de aplicações que executem em *rings* mais privilegiados, como HVM e SMM. Contudo, ao mesmo tempo em que este modo possibilita proteger aplicações legítimas, tais como gerenciadores de chaves criptográficas, de subversões e vazamentos, este modo também impede a inspeção de aplicações maliciosas.

Enquanto um binário SGX pode ser analisado estaticamente, fora do enclave, a sua execução dentro do enclave não pode ser monitorada devido ao isolamento das páginas de memória. Desta forma, atacantes podem criar exemplares que evadam a monitoração estática por parte de soluções de segurança, como antivírus, transferindo a exibição de seu comportamento malicioso para a etapa de execução dentro do enclave. Isto pode ser feito através da compressão do binário e/ou da geração de código em tempo de execução. Como o código gerado em tempo real fica armazenado nas páginas de memória, este é isolado pelo enclave de qualquer inspeção externa. Como agravante, este tipo de mecanismo pode ser usado para implementar exemplares do tipo *downloader* [Rossow et al. 2013], que obtém diferentes *payloads* através da Internet e os extrai apenas dentro do enclave. A adoção de conexão criptografada, como `HTTPS`, pode impedir completamente a inspeção do tráfego entre o enclave e o servidor remoto, tal qual proposto em exemplares conceituais [van Prooijen 2016]. Esse tipo de operação maliciosa é ilustrado na Figura 3.27.

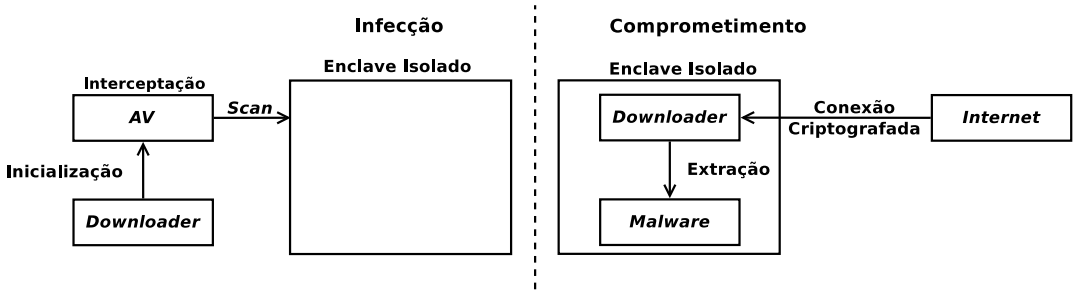


Figura 3.27. *Malware* em um enclave isolado. Atacantes podem evadir a detecção por antivírus movendo seus códigos para dentro de um enclave isolado. Ao se modularizar a amostra, o antivírus é capaz apenas de analisar o componente que carrega os demais, que não é malicioso por definição. Este componente, por sua vez, obtém os componentes maliciosos apenas dentro do enclave isolado, estando livre, portanto, de inspeções.

Atualmente, a única forma de se obter informações, ainda que limitadas, a partir de enclaves SGX é através de métodos de canal lateral [Schwarz et al. 2017]. Neste tipo de ataque, meta-informações de execução são utilizadas para inferir o conteúdo do enclave. Por exemplo, pode-se utilizar do monitor *Processor Tracer* para se comparar o perfil (frequência de *branches*, por exemplo) de uma aplicação conhecida quando em execução fora do enclave com o perfil da aplicação desconhecida em execução no enclave de modo a identificá-la [Lee et al. 2017]. Este tipo de técnica é efetiva visto que, apesar das páginas de memória serem isoladas, o modo SGX compartilha recursos de CPU, como

ALUs, unidades de *branch* e memória *cache* com as demais aplicações em execução no processador.

Ataques, de um modo geral, podem ser desenvolvidos utilizando-se não apenas de enclaves isolados, mas de todas as tecnologias anteriormente apresentadas, pois, da mesma maneira que as capacidades de monitoração destas podem ser empregadas para fins de análise, estas podem ser empregadas para propósitos maliciosos.

O carregamento de máquinas virtuais de *hardware* em sistemas já em execução (*live loading*) pode ser utilizado para mover sistemas inteiros para um estado infectado. possibilitando *rootkits* com ampla abrangência [Rutkowska 2006, Myers and Youndt 2007] e até mesmo com operação multi-plataforma. Além de HVMs, *rootkits* podem ser também implementados em SMM [Duflot et al. 2007, BSDaemon et al. 2008, Wecherowski 2009], se beneficiando da visão de todo o sistema que esse modo possibilita. O modo SMM pode ser utilizado, ainda, para a implementação de *keyloggers* [Embleton et al. 2008, Schiffman and Kaplan 2014], através do redirecionamento das interrupções de teclados para SMIs. *Keyloggers* podem ser também implementados a partir de dispositivos PCI, como GPUs [Ladakis et al. 2013], que se utilizem do acesso via DMA para monitorar escritas no *buffer* de teclado.

Enquanto tais ameaças ainda são, em sua maioria, provas de conceito, ataques utilizando-se destas possibilidades poderão ser visto na prática em um futuro próximo, tais como o primeiro *malware* de modo AMT foi recentemente identificado [Khandelwal 2017].

3.7. Considerações Finais

A análise de binários é uma tarefa fundamental dos processos de segurança, sendo aplicada em mecanismos de verificação e validação de códigos e sistemas. Arquiteturas modernas são complexas e, ao mesmo tempo em que possibilitam construções mais eficientes, trazem novos desafios para a monitoração dos sistemas construídos sobre elas. Notavelmente, a interferência em aplicações em execução é significantemente dificultada por mecanismos arquiteturais de proteção. Se, por um lado, isto amplia a proteção de aplicações legítimas, por outro, dificulta a análise de binários maliciosos.

Neste capítulo, apresentamos um panorama dos elementos estruturais presentes nas arquiteturas modernas, de modo a destacar os benefícios advindos destes e também os novos desafios impostos pelos mesmos. Destacamos, por exemplo, que o uso de enclaves isolados é capaz de aumentar a resistência de aplicações protegidas contra ataques diretos ao mesmo tempo em que permite a criação de exemplares de *malware* resistentes a inspeção por soluções antivírus tradicionais.

Na Seção 3.2, apresentamos os principais recursos de inspeção presentes em soluções de análise modernas, tal qual o carregamento de soluções forense em sistemas infectados, técnica que se tornou possível somente com o desenvolvimento das extensões de máquinas virtuais de *hardware* presentes nas arquiteturas modernas.

Na Seção 3.3, apresentamos diferentes tecnologias e como estas podem ser utilizadas para implementar os distintos recursos de inspeção anteriormente apresentadas. Discutimos soluções de diferentes granularidades e mostramos que, enquanto soluções

como máquinas virtuais de *hardware* possuem suporte nativo a muitas das tarefas de inspeção, outras soluções, como as baseadas no modo SMM, necessitam de maiores desenvolvimentos.

Na Seção 3.4, apresentamos as técnicas de monitoração empregadas de modo a prover os recursos de inspeção anteriormente apresentados. Discutimos, por exemplo, como seguidos *overflows* de contadores de *performance* podem ser utilizados para implementar execução *step-by-step*. Este tipo de técnica é embasada pela compreensão dos efeitos causados pela introdução deste tipo de recurso (contadores) na plataforma.

Na Seção 3.5, apresentamos diferentes aplicações para as tecnologias e técnicas de inspeção anteriormente discutidas. Nossa abordagem cobriu soluções de tempo real, como detecção de ataques e políticas de integridade de fluxo, e de inspeção, como as voltadas para procedimentos forenses e *debugging*.

Na Seção 3.6, discutimos o atual estágio de desenvolvimento do campo de soluções de análise com suporte de *hardware* e apontamos problemas em aberto. Dentre esses, destacam-se os desafios impostos pela adoção de enclaves isolados, tais como SGX, que, por um lado, protege aplicações legítimas de subversão, mas, ao mesmo tempo, torna exemplares de código maliciosos mais difíceis de serem analizados em tempo de execução.

Acreditamos que este trabalho possa contribuir com estudantes e profissionais da área em seus desenvolvimentos, seja pela apresentação dos recursos presentes nas arquiteturas modernas, pela categorização destes de acordo com a respectiva tarefa de análise ou mesmo pela identificação dos problemas em aberto, que podem ser abordados pela audiência em seus futuros projetos de pesquisa.

Por fim, nós encorajamos a leitura do *survey* [Botacin et al. 2018b], que prove informações complementares sobre mecanismos e técnicas de análise com suporte de *hardware*.

Agradecimentos. Os autores agradecem a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), em especial via Projeto FORTE - Forense Digital Tempestiva e Eficiente (Processo: 23038.007604/2014-69 - Edital 24/2014 - Programa Ciências Forenses).

Referências

[AMD 2013] AMD (2013). *AMD64 Architecture Programmer's Manual Volume 2*. AMD.

[AMD 2016] AMD (2016). Amd secure processor (built-in technology). `https://tinyurl.com/yaq2rhmv`.

[ARM 2009] ARM (2009). *ARM Sec. Technology - Building a Secure System using TrustZone Technology*. ARM.

[Arulraj et al. 2014] Arulraj, J., Jin, G., and Lu, S. (2014). Leveraging the short-term memory of hardware to diagnose production-run software failures. *SIGARCH Comput. Archit. News*, 42(1).

[Aumasson and Merino 2016] Aumasson, J. and Merino, L. (2016). Sgx secure enclaves in practice: Sec. and crypto review.

[Azab et al. 2010] Azab, A. M., Ning, P., Wang, Z., Jiang, X., Zhang, X., and Skalsky, N. C. (2010). Hypersentry: Enabling stealthy in-context measurement of hypervisor integrity. In *Proc. 17th ACM Conf. on Comp. and Comm. Sec.*, CCS '10. ACM.

[Bahador et al. 2014] Bahador, M., Abadi, M., and Tajoddin, A. (2014). Hpcmalhunter: Behavioral malware detection using hardware performance counters and singular value decomposition. In *2014 4th Intl. Conf. on Comp. and Knowledge Engineering (ICCKE)*.

[Barbosa and Branco 2014] Barbosa, G. N. and Branco, R. R. (2014). Prevalent characteristics in modern malware. `http://www.kernelhacking.com/rodrigo/docs/blackhat2014-presentation.pdf`.

[Bayer et al. 2006] Bayer, U., Kruegel, C., and Kirda, E. (2006). Ttanalyze: A tool for analyzing malware. In *15th European Inst. for Comp. Antivirus Research (EICAR 2006) Annual Conf.* EICAR.

[Bellard 2005] Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *Proc. USENIX Annual Technical Conf.*, ATC '05. USENIX Association.

[Botacin et al. 2017] Botacin, Falcão, Geus, and Grégio (2017). Analysis, anti-analysis, anti-anti-analysis: An overview of the evasive malware scenario. `https://sbseg2017.redes.unb.br/wp-content/uploads/2017/04/20171109_ANAIS_SBSEG_2017_FINAL_E-BOOK.pdf`.

[Botacin et al. 2015] Botacin, Geus, and Grégio (2015). Uma visão geral do malware ativo no espaço nacional da internet entre 2012 e 2015. `http://siaiap34.univali.br/sbseg2015/anais/WFC/artigoWFC02.pdf`.

[Botacin et al. 2018a] Botacin, M., Geus, P. L. D., and Grégio, A. (2018a). Enhancing branch monitoring for security purposes: From control flow integrity to malware analysis and debugging. *ACM Trans. Priv. Secur.*, 21(1):4:1–4:30.

[Botacin et al. 2018b] Botacin, M., Geus, P. L. D., and grégio, A. (2018b). Who watches the watchmen: A security-focused review on current state-of-the-art techniques, tools, and methods for systems and binary analysis on modern platforms. *ACM Comput. Surv.*, 51(4):69:1–69:34.

[Botacin et al. 2018c] Botacin, M. F., de Geus, P. L., and Grégio, A. R. A. (2018c). The other guys: automated analysis of marginalized malware. *Journal of Computer Virology and Hacking Techniques*, 14(1):87–98.

[Branco et al. 2012] Branco, R. R., Barbosa, G. N., and Neto, P. D. (2012). Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti- vm technologies. `http://www.kernelhacking.com/rodrigo/docs/blackhat2012-paper.pdf`.

[BSDaemon et al. 2008] BSDaemon, coideloco, and D0nad0n (2008). System management mode hack - using smm for "other purposes". https://tinyurl.com/jxeao4u.

[Cai et al. 2007] Cai, H., Shao, Z., and Vaynberg, A. (2007). Certified self-modifying code. *SIGPLAN Not.*, 42(6):66–77.

[Chen et al. 2008] Chen, X., Andersen, J., Mao, Z. M., Bailey, M., and Nazario, J. (2008). Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *2008 IEEE Intl. Conf. on Depend. Syst. and Net. With FTCS and DCC (DSN)*. IEEE.

[Cheng et al. 2014] Cheng, Y., Zhou, Z., Miao, Y., Ding, X., and Deng, H. R. (2014). Ropecker: A generic and practical approach for defending against rop attacks. In *Symp. on Net. and Dist. System Sec. (NDSS)*. Internet Society.

[CoreBoot 2015] CoreBoot (2015). Coreboot. http://www.coreboot.org/.

[Dinaburg et al. 2008a] Dinaburg, A., Royal, P., Sharif, M., and Lee, W. (2008a). Ether: Malware analysis via hardware virtualization extensions. In *Proc. of the 15th ACM Conf. on Comp. and Comm. Sec.*, CCS '08. ACM.

[Dinaburg et al. 2008b] Dinaburg, A., Royal, P., Sharif, M., and Lee, W. (2008b). Ether: Malware analysis via hardware virtualization extensions. In *Proc. 15th ACM Conf. on Comp. and Comm. Sec.*, CCS '08. ACM.

[do Brasil 2018] do Brasil, G. (2018). Governo digital. https://www.governodigital.gov.br/.

[Duflot et al. 2007] Duflot, L., Etiemble, D., and Grumelard, O. (2007). Using cpu system management mode to circumvent operating system sec. functions. https://tinyurl.com/y7mlduy9.

[DynamoRIO 2001] DynamoRIO (2001). Dynamic instrumentation tool platform. https://tinyurl.com/ybenfvw9.

[Embleton et al. 2008] Embleton, S., Sparks, S., and Zou, C. (2008). Smm rootkits: A new breed of os independent malware. In *Proc. 4th Intl. Conf. on Sec. and Priv. in Communication Netowrks*, SecureComm '08. ACM.

[Fattori et al. 2010] Fattori, A., Paleari, R., Martignoni, L., and Monga, M. (2010). Dynamic and transparent analysis of commodity production syst. In *Proc. IEEE/ACM Int. Conf. on Automated Software Engineering*, ASE '10. ACM.

[Felderer et al. 2016] Felderer, M., Büchler, M., Johns, M., D. Brucker, A., Breu, R., and Pretschner, A. (2016). Sec. testing: A survey.

[G1 2017] G1 (2017). Mobile banking se torna meio mais usado para transacoes bancarias, diz febraban. https://g1.globo.com/economia/seu-dinheiro/noticia/mobile-banking-se-torna-meio-mais-usado-para-transacoes-bancarias-diz-febraban.ghtml.

[Grégio et al. 2013] Grégio, A. R. A., Fernandes, D. S. o., Afonso, V. M., de Geus, P. L., Martins, V. F., and Jino, M. (2013). An empirical analysis of malicious internet banking software behavior. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1830–1835, New York, NY, USA. ACM.

[Intel 2013] Intel (2013). *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Intel.

[Intel 2015] Intel (2015). Pin - a dynamic binary instrumentation tool. https://tinyurl.com/m685m25.

[Intel 2018] Intel (2018). Pcie device security enhancements specification. https://www.intel.com/content/www/us/en/io/pci-express/pcie-device-security-enhancements-spec.html.

[Ionescu 2015] Ionescu, A. (2015). Battle of the skm and ium: How windows 10 rewrites os architecture. https://tinyurl.com/na375ur.

[ISECLAB 2010] ISECLAB (2010). Anubis - malware analysis for unknown binaries. https://anubis.iseclab.org/.

[Kang et al. 2009] Kang, M. G., Yin, H., Hanna, S., McCamant, S., and Song, D. (2009). Emulating emulation-resistant malware. In *Proc. 1st ACM Work. on Virtual Machine Sec.*, VMSec '09. ACM.

[Kaspersky 2015] Kaspersky (2015). Beaches, carnivals and cybercrime: Kaspersky lab shares insights on brazilian cyber underground. https://www.kaspersky.com/about/press-releases/2015_beaches-carnivals-and-cybercrime-kaspersky-lab-shares-insights-on-brazilian-cyber-underground.

[Khandelwal 2017] Khandelwal, S. (2017). First-ever data stealing malware found using intel amt tool to bypass firewall. https://tinyurl.com/y7e7kg8v.

[Kompalli and Sarat 2014] Kompalli and Sarat (2014). Using existing hardware services for malware detection. In *Proc. 2014 IEEE Sec. and Priv. Work.s*, SPW'14. IEEE Comp. Society.

[Ladakis et al. 2013] Ladakis, E., Koromilas, L., Vasiliadis, G., Polychronakis, M., and Ioannidis, S. (2013). You can type, but you can't hide: A stealthy gpu-based keylogger. https://tinyurl.com/cbzp42n.

[Lee et al. 2013] Lee, H., Moon, H., Jang, D., Kim, K., Lee, J., Paek, Y., and Kang, B. B. (2013). Ki-mon: A hardware-assisted event-triggered monitoring platform for mutable kernel object. In *22nd USENIX Sec. Symposium*. USENIX.

[Lee et al. 2017] Lee, S., Shih, M.-W., Gera, P., Kim, T., Kim, H., and Peinado, M. (2017). Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Sec. Symposium (USENIX Sec. 17)*. USENIX Association.

[Li et al. 2004] Li, K., Ding, S., McCreary, D., and Webb, S. (2004). Analysis of state exposure control to prevent cheating in online games. In *Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '04, pages 140–145, New York, NY, USA. ACM.

[LibVMI 2015] LibVMI (2015). Introduction to libvmi. https://tinyurl.com/y8d4xbq9.

[Liu et al. 2014] Liu, K., Lu, S., and Liu, C. (2014). Poster: Fingerprinting the publicly available sandboxes. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 1469–1471, New York, NY, USA. ACM.

[Mandt et al. 2016] Mandt, T., Solnik, M., and Wang, D. (2016). Demystifying the secure enclave processor.

[Martignoni et al. 2010] Martignoni, L., Fattori, A., Paleari, R., and Cavallaro, L. (2010). Live and trustworthy forensic analysis of commodity production syst. In *Proc. 13th Intl. Conf. on Recent Advances in Intrusion Detection*, RAID'10. Springer-Verlag.

[Martignoni et al. 2009] Martignoni, L., Paleari, R., Roglia, G. F., and Bruschi, D. (2009). Testing cpu emulators. In *Proc. 18th Intl Symp. on Software Testing and Analysis*, ISSTA '09. ACM.

[Moon et al. 2012] Moon, H., Lee, H., Lee, J., Kim, K., Paek, Y., and Kang, B. B. (2012). Vigilare: Toward snoop-based kernel integrity monitor. In *Proc. 2012 ACM Conf. on Comp. and Comm. Sec.*, CCS '12. ACM.

[More and Tapaswi 2014] More, A. and Tapaswi, S. (2014). Virtual machine introspection: towards bridging the semantic gap. *Journal of Cloud Computing*, 3(1).

[Myers and Youndt 2007] Myers, M. and Youndt, S. (2007). An introduction to hardware-assisted virtual machine (hvm) rootkits. `https://tinyurl.com/y8wfsye5`.

[Neugschwandtner et al. 2010] Neugschwandtner, M., Platzer, C., Comparetti, P., and Bayer, U. (2010). danubis - dynamic device driver analysis based on virtual machine introspection. In Kreibich, C. and Jahnke, M., editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 6201 of *Lecture Notes in Comp. Science*. Springer Berlin Heidelberg.

[Nguyen et al. 2009] Nguyen, A. M., Schear, N., Jung, H., Godiyal, A., King, S. T., and Nguyen, H. D. (2009). Mavmm: Lightweight and purpose built vmm for malware analysis. In *Proc. 2009 Annual Comp. Sec. Applications Conf.*, ACSAC '09. IEEE Comp. Society.

[Opsahl 2013] Opsahl, J. M. G. (2013). *Open-source virtualization : Functionality and performance of Qemu/KVM, Xen, Libvirt and VirtualBox*. PhD thesis, Oslo Univ.

[Paleari 2015] Paleari, R. (2015). Fast coverage analysis for binary applications. `https://tinyurl.com/y7obk3y5`.

[Paleari et al. 2009] Paleari, R., Martignoni, L., Roglia, G. F., and Bruschi, D. (2009). A fistful of redpills: How to automatically generate procedures to detect cpu emulators. In *Proc. 3rd USENIX Conf. on Offensive Technologies*, WOOT'09. USENIX Association.

[Pappas et al. 2013] Pappas, V., Polychronakis, M., and Keromytis, A. D. (2013). Transparent rop exploit mitigation using indirect branch tracing. In *Proc. 22Nd USENIX Conf. on Sec.*, SEC'13. USENIX Association.

[Petroni et al. 2004] Petroni, Jr., N. L., Fraser, T., Molina, J., and Arbaugh, W. A. (2004). Copilot - a coprocessor-based kernel runtime integrity monitor. In *Proc. 13th Conf. on USENIX Sec. Symp. - Volume 13*, SSYM'04. USENIX Association.

[Project 2015] Project, X. (2015). vmcs.c. `http://xenbits.xen.org/gitweb/?p=xen.git;a=blob;f=xen/arch/x86/hvm/vmx/vmcs.c`.

[Quynh and Suzaki 2010] Quynh, N. A. and Suzaki, K. (2010). Virt-ice: Next-generation debugger for malware analysis. `https://tinyurl.com/ybszcbxn`.

[Reina et al. 2012] Reina, A., Fattori, A., Pagani, F., Cavallaro, L., and Bruschi, D. (2012). When hardware meets software: A bulletproof solution to forensic memory acquisition. In *Proc. 28th Annual Comp. Sec. Applications Conf.*, ACSAC '12. ACM.

[Roemer et al. 2012] Roemer, R., Buchanan, E., Shacham, H., and Savage, S. (2012). Return-oriented programming: Syst., languages, and applications. *ACM Trans. Inf. Syst. Secur.*, 15(1).

[Rossow et al. 2013] Rossow, C., Dietrich, C., and Bos, H. (2013). Large-scale analysis of malware downloaders. In *Proc. of the 9th Inter. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'12. Springer.

[Rutkowska 2006] Rutkowska (2006). Subverting vista kernel for fun and for profit. `https://tinyurl.com/y86ltylh`.

[Rutkowska 2010] Rutkowska (2010). Qubes os project. `https://www.qubes-os.org/`.

[Rutkowska and Wojtczuk 2008] Rutkowska, J. and Wojtczuk, R. (2008). Preventing and detecting xen hypervisor subversions. `https://tinyurl.com/44denv2`.

[Schiffman and Kaplan 2014] Schiffman, J. and Kaplan, D. (2014). The smm rootkit revisited: Fun with usb. In *Availability, Reliability and Sec. (ARES), 2014 9th Intl. Conf. on*. IEEE.

[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. `https://arxiv.org/abs/1702.08719`.

[SeaBIOS 2015] SeaBIOS (2015). Seabios. `http://www.seabios.org/SeaBIOS`.

[Seshadri et al. 2007] Seshadri, A., Luk, M., Qu, N., and Perrig, A. (2007). Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *Proc. 21st ACM SIGOPS Symp. on Operating Syst. Principles*, SOSP '07. ACM.

[Shi et al. 2014] Shi, H., Alwabel, A., and Mirkovic, J. (2014). Cardinal pill testing of system virtual machines. In *23rd USENIX Sec. Symp. (USENIX Sec. 14)*. USENIX Association.

[Shinagawa et al. 2009] Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., Kono, K., Chiba, S., Shinjo, Y., and Kato, K. (2009). Bitvisor: A thin hypervisor for enforcing i/o device sec. In *Proc. ACM SIGPLAN/SIGOPS Intl. Conf. on Virtual Execution Environments*, VEE '09.

[Sikorski and Honig 2012] Sikorski, M. and Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition.

[Song et al. 2008] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G., Liang, Z., Newsome, J., Poosankam, P., and Saxena, P. (2008). Bitblaze: A new approach to comp. sec. via binary analysis. In *Proc. 4th Intl. Conf. on Information Syst. Sec.*, ICISS '08. Springer-Verlag.

[Thober et al. 2008] Thober, M., Pendergrass, J. A., and McDonell, C. D. (2008). Improving coherency of runtime integrity measurement. In *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, STC '08, pages 51–60, New York, NY, USA. ACM.

[van Prooijen 2016] van Prooijen, J. (2016). The design of malware on modern hardware. `https://tinyurl.com/y8rwfj5t`.

[Vasudevan and Yerraballi 2006a] Vasudevan, A. and Yerraballi, R. (2006a). Cobra: Fine-grained malware analysis using stealth localized-executions. In *Proc. 2006 IEEE Symp. on Sec. and Priv.*, SP '06. IEEE Comp. Society.

[Vasudevan and Yerraballi 2006b] Vasudevan, A. and Yerraballi, R. (2006b). Spike: Engineering malware analysis tools using unobtrusive binary-instrumentation. In *Proc. 29th Australasian Comp. Science Conf. - Volume 48*, ACSC '06. Australian Comp. Society, Inc.

[Wang et al. 2010] Wang, J., Stavrou, A., and Ghosh, A. (2010). Hypercheck: A hardware-assisted integrity monitor. In *Proc. 13th Intl. Conf. on Recent Advances in Intrusion Detection*, RAID'10. Springer-Verlag.

[Wang et al. 2011] Wang, J., Zhang, F., Sun, K., and Stavrou, A. (2011). Firmware-assisted memory acquisition and analysis tools for digital forensics. In *Proc. 2011 6th IEEE Intl. Wksp on Systematic Approaches to Digital Forensic Engineering*, SADFE '11. IEEE Comp. Society.

[Wecherowski 2009] Wecherowski, F. (2009). A real smm rootkit: Reversing and hooking bios smi handlers. `https://tinyurl.com/knoms4t`.

[Will et al. 2017] Will, N. C., Condé, R. C. R., and Maziero, C. A. (2017). Mecanismos de segurança baseados em hardware: uma introdução à arquitetura intel sgx. `https://sbseg2017.redes.unb.br/wp-content/uploads/2017/04/20171107-SBSeg2017-Livro_de_Minicursos.pdf`.

[Willems et al. 2012a] Willems, C., Hund, R., Fobian, A., Felsch, D., Holz, T., and Vasudevan, A. (2012a). Down to the bare metal: Using processor features for binary analysis. In *Proc. of the 28th Annual Comp. Sec. Applications Conf.*, ACSAC '12. ACM.

[Willems et al. 2012b] Willems, C., Hund, R., and Holz, T. (2012b). Cxpinspector: Hypervisor-based, hardware-assisted system monitoring. Technical report, Horst Gortz Institute for IT Sec.

[Xia et al. 2012] Xia, Y., Liu, Y., Chen, H., and Zang, B. (2012). Cfimon: Detecting violation of control flow integrity using performance counters. In *Proc. 2012 42nd Annual IEEE/IFIP Intl. Conf. on Depend. Syst. and Net. (DSN)*, DSN '12. IEEE Comp. Society.

[Xu et al. 2017] Xu, J., Mu, D., Xing, X., Liu, P., Chen, P., and Mao, B. (2017). Postmortem program analysis with hardware-enhanced post-crash artifacts. In *26th USENIX Sec. Symposium*. USENIX.

[Yan et al. 2012] Yan, L.-K., Jayachandra, M., Zhang, M., and Yin, H. (2012). V2e: Combining hardware virtualization and softwareemulation for transparent and extensible malware analysis. In *Proc. 8th ACM SIGPLAN/SIGOPS Conf. on Virtual Execution Environments*, VEE '12.

[Zhang 2013] Zhang, F. (2013). Iocheck: A framework to enhance the security of i/o devices at runtime. In *2013 43rd Annual IEEE/IFIP Conf. on Depend. Syst. and Net. Wksp (DSN-W)*.

[Zhang et al. 2015] Zhang, F., Leach, K., Stavrou, A., Wang, H., and Sun, K. (2015). Using hardware features for increased debugging transparency. In *2015 IEEE Symp. on Sec. and Priv.* IEEE.

[Zhang et al. 2013] Zhang, F., Leach, K., Sun, K., and Stavrou, A. (2013). Spectre: A depend. introspection framework via system management mode. In *Proc. 43rd Annual IEEE/IFIP Intl. Conf. on Depend. Syst. and Net. (DSN)*, DSN '13. IEEE Comp. Society.

Capítulo 4

Blockchain para Segurança em Redes Elétricas Inteligentes: Aplicações, Tendências e Desafios

Diogo M. F. Mattos (UFF), Dianne S. V. Medeiros (UFF), Natalia C. Fernandes (UFF), Marcela T. de Oliveira (UFF), Gabriel R. Carrara (UFF), Arthur A. Z. Soares (UFF), Luiz Claudio S. Magalhães (UFF), Diego Passos (UFF), Ricardo C. Carrano (UFF), Igor M. Moraes (UFF), Célio V. N. Albuquerque (UFF), Débora C. Muchaluat-Saade (UFF)

Abstract

The electric power grid is the world's largest engineering system, and its secure and reliable operation is vital to human activities. The introduction of intelligence in the electrical power grid imposes challenges that require new techniques and approaches to provide cybersecurity. In this chapter, we discuss the use of blockchain to provide security and reliability to smart grids. Blockchain allows, in a distributed peer-to-peer network, untrusted nodes to correctly and verifiably interact with each other, without any reliable intermediary. We explore smart contracts, codes resident in blockchain that automate multi-step processes as a way to automatically trade electric energy. We present the Hyperledger Fabric, Multichain, Parity and Corda platforms for the development of blockchain applications. We also discuss initiatives, challenges and research opportunities of blockchain technology in the electrical sector.

Resumo

A rede elétrica é o maior sistema de engenharia do mundo e o seu funcionamento seguro e confiável é vital para as atividades humanas. A introdução de inteligência nas redes elétricas impõe desafios que requerem novas técnicas e abordagens de segurança cibernética. Este capítulo discute o uso da tecnologia blockchain para prover segurança e confiabilidade às redes elétricas inteligentes. A tecnologia blockchain permite que, em uma rede par-a-par distribuída, nós não confiáveis interajam entre si, sem haver qualquer intermediário confiável, de maneira correta e verificável. O capítulo discute como os contratos inteligentes, códigos residentes na blockchain que automatizam processos de múltiplas etapas, podem ser usados na comercialização automática de energia. São apresentadas as plataformas Hyperledger Fabric, Multichain, Parity e Corda para o desenvolvimento de aplicações sobre blockchain. O capítulo apresenta ainda iniciativas, desafios e oportunidades de pesquisa da tecnologia blockchain no setor elétrico.

4.1. Introdução

A rede elétrica é o maior e mais bem-sucedido sistema de engenharia do mundo [Mo et al., 2012]. A confiabilidade alcançada pelas redes elétricas é muitas vezes superior à alcançada em sistemas de comunicação [Dütsch e Steinecke, 2017]. Contudo, nas últimas décadas, o desenvolvimento das redes elétricas não tem acompanhando os avanços industriais e sociais que aumentam as demandas de suprimento de energia [Wang e Lu, 2013]. Nesse sentido, para atender às demandas crescentes por energia, é necessário agregar novas fontes à rede elétrica e gerenciar de maneira eficiente as fontes tradicionais, como hidroelétrica e termoelétrica, como também as fontes renováveis e variáveis, como energia eólica e solar. Para tanto, o *National Institute of Standard and Technology* (NIST) reuniu esforços para definir a próxima geração das redes elétricas, referenciada como *Smart Grids* ou Redes Elétricas Inteligentes [Greer et al., 2014].

As redes elétricas inteligentes integram tecnologias de comunicação bidirecional nas redes elétricas [Mo et al., 2012, Guimarães et al., 2013] e, assim, os diversos equipamentos que fazem parte das redes elétricas passam a formar uma infraestrutura dinâmica e interativa, com a capacidade de gerenciar o consumo de energia, como através da infraestrutura avançada de medição (*Advanced Metering Infrastructure* – AMI) [Sui et al., 2009] e das aplicações de resposta à demanda [Gunter et al., 2008]. No entanto, o cenário das redes elétricas é composto por uma variedade de sistemas, com numerosos proprietários e sujeitos a diversos mecanismos e agências de regulação. A variedade de atores e sistemas se comunicando no mercado de energia elétrica suscitam um grande número de vulnerabilidades [Mo et al., 2012]. As novas vulnerabilidades relacionam-se com a agregação de fontes distribuídas de geração de energia à rede e com o fato de os novos atores, nós da rede, não confiarem totalmente nos demais, já que os nós podem pertencer a organizações distintas e não ter sua identidade e comportamento atestados. Dessa forma, o cenário de redes elétricas inteligentes é propício para a aplicações que se baseiam na tecnologia *Blockchain*, referenciada neste capítulo como Cadeia de Blocos [Christidis e Devetsikiotis, 2016, Mengelkamp et al., 2018b]. A cadeia de blocos tem o potencial de simplificar os processos de negociação no mercado de energia elétrica, que são impactados pela introdução de novas fontes de geração de energia elétrica renováveis e distribuídas, e de garantir a verificabilidade das ações na rede de comunicação [Dütsch e Steinecke, 2017].

Segurança e Privacidade em Redes Elétricas Inteligentes

As redes elétricas inteligentes agregam tecnologias de comunicação de dados às redes de geração, transmissão e distribuição de energia elétrica para permitir maior controle e monitoramento em tempo real da carga gerada e consumida na rede. Ademais, as redes elétricas inteligentes permitem o desenvolvimento de novas aplicações e agregam novas formas de geração à rede, muitas vezes de pequena escala e distribuídas [Lopes et al., 2016, Mengelkamp et al., 2018b]. A geração distribuída implica a necessidade de controle fino da rede elétrica e, em especial, a adoção de mecanismos automatizados para negociação de energia elétrica pelos consumidores finais, que passam a ser produtores e consumidores de energia, chamados de "prossumidores" (*prosumers*).

A Figura 4.1 mostra os quatro componentes principais em uma rede elétrica: a geração, a transmissão, a distribuição e o consumo. Vale ressaltar que paralelamente à

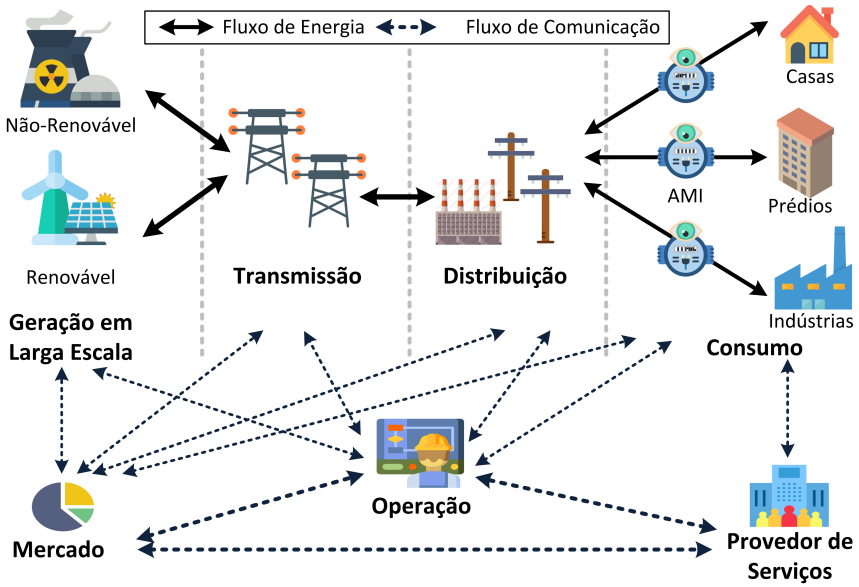


Figura 4.1. Fluxos de comunicação e os quatro principais componentes das redes elétricas. A Geração produz energia de diferentes maneiras. A Transmissão move a energia elétrica em altas voltagens através da infraestrutura. A Distribuição transforma a energia elétrica para média e baixa voltagem e distribui para o consumo. Os consumidores usam a energia elétrica.

infraestrutura dedicada para o fornecimento de energia, desde a geração até o consumo, há uma variedade de sistemas com diversos proprietários distintos que negociam, operam e fornecem a energia elétrica aos consumidores [Mo et al., 2012]. A ação dos sistemas paralelos à infraestrutura elétrica é muitas das vezes regulada por mecanismos governamentais para garantir o correto funcionamento da rede elétrica. Nas redes elétricas inteligentes, a rede de comunicação é subjacente à rede elétrica e tem por finalidade amparar a comercialização, a medição e o monitoramento da geração, transmissão, distribuição e consumo de energia elétrica. A rede de comunicação movimenta os dados de transações e monitoramento entre os diversos atores envolvidos no sistema, como os mercados de negociação de energia, os centros de controle e operação e os provedores de serviços aos consumidores finais.

Com a introdução das tecnologias de comunicação na rede elétrica, novos vetores de ataques aparecem. Destacam-se três vetores de ataque à segurança e à privacidade das redes elétricas inteligentes [Chen et al., 2012, Li et al., 2012]:

- **ataque de vulnerabilidade** causado pelo mau funcionamento de um dispositivo ou canal de comunicação ou simplesmente pela falta de sincronização entre informações de controle. As informações de controle podem ser comprometidas pela maneira como são fornecidas ou pelas condições de canal não confiáveis, levando a um processo de controle incorreto. As vulnerabilidades são provocadas principalmente pela confiabilidade intrínseca na rede de comunicação. A ausência de mecanismos para a realização de diagnósticos e a gravação de registros de ações dificultam a localização das vulnerabilidades. Vale ressaltar a existência de vulnerabilidades provocadas pela infiltração de dispositivos infectados no perímetro de segurança, como memórias USB [Langner, 2011];

- **ataque de injeção de dados** visa alterar as medidas de alguns medidores, a fim de manipular as operações da rede elétrica inteligente. O impacto desse ataque é principalmente a perda de receita. No entanto, dependendo do número de sensores controlados pelo atacante, é possível que o atacante force determinadas ações de controle sobre a rede elétrica, podendo levar a impactos catastróficos como a ocorrência de apagões em grandes áreas urbanas [Guimarães et al., 2013, Noce et al., 2017];

- **ataque intencional** é quando o atacante é capaz de ter total compreensão da topologia da rede e, portanto, pode utilizar totalmente a estrutura da rede para interromper as operações, paralisando alguma fração dos nós. O ataque intencional pode ser implementado por meio de ataque coordenado de negação de serviço (*Denial of Service* - DoS) e contribui para a interrupção da rede devido a desconexões de nós da rede de comunicação subjacente.

No novo cenário das redes elétricas inteligentes, torna-se imperativa a adoção de novas tecnologias capazes de assegurar a confiabilidade da rede elétrica, mesmo quando não há confiança entre os pares, de permitir o controle e a manutenção dos dados de produção e de consumo de energia de forma distribuída, de permitir a auditoria do histórico de transações de compra e venda de energia elétrica de maneira irrefutável e, por fim, de garantir que contratos de compra e venda de energia elétrica sejam executados corretamente, independentemente da cooperação dos participantes. Portanto, a tecnologia de cadeia de blocos é uma das viabilizadoras de aplicações seguras em redes elétricas inteligentes [Jesus et al., 2018].

Cadeia de Blocos para a Segurança entre Nós Não Confiáveis

A tecnologia de cadeia de blocos permite o desenvolvimento de aplicações sobre uma rede par-a-par, em que os membros não confiáveis interagem entre si, sem um intermediário confiável, mas de maneira verificável. A cadeia de blocos consiste em um histórico imutável de transações em uma estrutura de dados distribuída, em que cada nó da rede contém uma réplica de todos os blocos. Cada nó participante do sistema executa protocolos de consenso que validam as transações e as agrupam em blocos, que são encadeados usando uma referência ao bloco antecessor. A referência é um resumo criptográfico (*hash*), obtido através de algoritmos criptográficos unidirecionais. Essa propriedade torna improvável a recuperação dos dados originais a partir do resumo criptográfico gerado, garantindo a integridade do conteúdo e a segurança da cadeia. A cadeia de blocos é uma tecnologia capaz de atender às necessidades do sistema de coleta de dados, dando mais transparência e agilidade à comercialização de energia, na medida em que permite que as medições de fluxo de energia, assim que geradas, sejam disponibilizadas em um repositório distribuído e auditável. Paralelamente, um contrato inteligente (*smart contract*) [Bartoletti e Pompianu, 2017], uma evolução da cadeia de blocos para execução de códigos distribuídos, é essencial para a negociação segura de energia elétrica entre geradores, consumidores e "prossumidores", ao passo que assegura a execução dos contratos pela rede par-a-par subjacente à cadeia.

Contratos inteligentes (*smart contracts*) são definidos como a execução dos termos de um contrato através de um protocolo de transações computacionais [Wood, 2014,

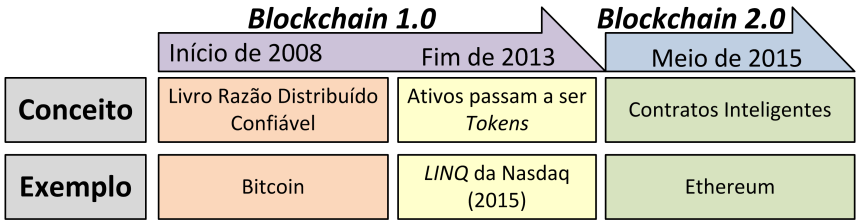


Figura 4.2. Gerações no desenvolvimento da tecnologia de cadeia de blocos. O conceito surge com o livro-razão confiável e distribuído implantado pela Bitcoin, evolui para a transação de ativos na forma de *tokens* e se concretiza como uma segunda geração após a introdução de contratos inteligentes na estrutura de dados da cadeia. Adaptado de [Dütsch e Steinecke, 2017].

Christidis e Devetsikiotis, 2016]. Assim, um contrato inteligente traduz as cláusulas de um contrato real para código que é executado em um ambiente capaz de forçar o seu correto funcionamento [Szabo, 1997]. Ao se considerar a execução dos contratos inteligentes como uma aplicação de cadeia de blocos, os contratos inteligentes são códigos armazenados na própria cadeia de blocos. Dessa forma, um contrato inteligente é uma aplicação armazenada na cadeia de blocos, acessível através de um endereço conhecido. Um contrato inteligente é ativado quando uma transação é disparada para o seu endereço, em que podem ser implementados diversos tipos de ação sobre diferentes ativos. No contexto de redes elétricas inteligentes, os contratos inteligentes são apontados como uma solução viável para realizar a negociação segura e descentralizada de energia [Aitzhan e Svetinovic, 2018].

As aplicações em cadeia de blocos ainda estão em fase inicial de industrialização. No desenvolvimento da tecnologia de cadeia de blocos, é possível diferenciar três momentos principais e definir duas gerações, mostrados na Figura 4.2. A *blockchain 1.0* refere-se à tecnologia de armazenamento de transações na cadeia, de forma distribuída e através da criação de ativos na forma de *tokens*. A primeira aplicação a ganhar notoriedade no uso de cadeia de blocos foi a criptomoeda Bitcoin [Nakamoto, 2008], introduzida em 2009. A ideia central é criar um livro-razão distribuído em uma rede par-a-par seguro e confiável. Um marco na evolução da tecnologia foi o uso da cadeia de blocos para gerir de forma segura a transferência de ativos sob a forma de *tokens*, a partir de 2013. Em 2015, essa tecnologia passou a ser usada pelo sistema LINQ da Nasdaq para armazenar transações privadas seguras[1]. A segunda geração da tecnologia, referenciada como *blockchain 2.0* [Greve et al., 2018], consiste na introdução dos contratos inteligentes. A plataforma Ethereum [Wood, 2014] foi a primeira a suportar os contratos inteligentes ao permitir o armazenamento de código de execução automática na cadeia. A principal evolução entre a primeira e a segunda gerações foi que, a exemplo da Bitcoin, a linguagem de programação das cadeias de blocos da primeira geração eram "orientadas a pilha" e não permitiam laços (*loops*) no código executável, a fim de evitar ciclos mortos (*deadlocks*) no sistema. Já em sistemas de segunda geração, como a Ethereum, são permitidas as execuções de laços nos código, que consomem créditos, no caso da Ethereum quantificados em *Ether*, ao executar contratos inteligentes. Assim, no caso de um ciclo morto, a execução é interrompida quando esgotam-se os créditos da conta usada [Merz, 2016].

[1]Acessível em http://ir.nasdaq.com/news-releases/news-release-details/nasdaq-linq-enables-first-ever-private-securities-issuance.

Objetivos do Capítulo

O objetivo principal deste capítulo é apresentar como a tecnologia de cadeia de blocos (*blockchain*) melhora a segurança, provendo a confiabilidade, a privacidade e a auditoria, em sistemas críticos, como as redes elétricas inteligentes. O capítulo foca na aplicabilidade da tecnologia de cadeia de blocos no setor elétrico, a fim de aprimorar os processos de monitoramento e faturamento da rede de geração e transmissão de energia elétrica. O uso da tecnologia de cadeia de blocos e contratos inteligentes tem o potencial de prover segurança ao sistema elétrico e reduzir os erros de armazenamento e processamento das medições no sistema. Adicionalmente, essa tecnologia reduz o risco de fraudes através de uma auditoria confiável e da garantia de execução dos contratos inteligentes. O capítulo conta com uma abordagem teórica sobre o tema, apresentando uma visão geral sobre a tecnologia de cadeia de blocos e o uso dos contratos inteligentes, discutindo especificamente os desafios encontrados no setor elétrico para monitorar ativos e realizar o faturamento entre empresas parceiras e consumidores de forma confiável. Apresentam-se, ainda, as tendências e contribuições da tecnologia de cadeia de blocos particularmente para essa nova área de pesquisa. Compara-se o desempenho do modelo baseado nessa tecnologia e dos modelos atuais utilizados para monitoramento e faturamento no sistema elétrico. Por fim, o capítulo também analisa e discute os desafios em aberto para motivar os participantes a desenvolver pesquisas na área de segurança para redes elétricas.

Organização do Capítulo

O restante do capítulo está organizado da seguinte forma. A Seção 4.2 discute os principais desafios em segurança e privacidade nas redes elétricas inteligentes. A Seção 4.3 apresenta as principais tecnologias para o desenvolvimento de aplicações sobre cadeia de blocos. O controle distribuído e a necessidade de se atingir o consenso em uma cadeia de blocos com nós não confiáveis são analisados na Seção 4.4. Os contratos inteligentes e a sua aplicação no mercado de energia elétrica são explorados na Seção 4.5. As aplicações de cadeias de blocos em redes elétricas inteligentes são abordadas na Seção 4.6. A Seção 4.7 discute as principais tendências e desafios de pesquisa da tecnologia de cadeia de blocos em redes elétricas inteligentes. A Seção 4.8 conclui o capítulo.

4.2. Segurança e Privacidade em Redes Elétricas Inteligentes

Novas vulnerabilidades são inseridas quando se introduzem as técnicas de comunicação nas redes elétricas inteligentes. Nesse novo cenário, dispositivos autônomos, como medidores inteligentes, tornam-se susceptíveis a ataques. Os atacantes de uma rede elétrica inteligente podem ser atacantes individuais; grupos criminosos; desenvolvedores de *spyware* / *malware*; *phishers*; espiões; sabotadores; terroristas ou agentes de serviços de inteligência estrangeiros [Guimarães et al., 2013, Lopes et al., 2016]. Os principais requisitos de segurança para as redes elétricas inteligentes são a disponibilidade, a integridade, a privacidade, a autenticação, a autorização, a auditoria, o não repúdio e a confiança entre os componentes da rede [Mo et al., 2012, Wang e Lu, 2013]. No contexto das aplicações de cadeia de blocos para redes elétricas inteligentes, destacam-se as propriedades de privacidade, auditoria e não repúdio que são garantidas naturalmente pelas caracterís-

ticas de formação da cadeia de blocos. Ademais, como a confiabilidade entre nós da rede não é garantida, a cadeia de blocos fornece mecanismos para execução de código entre pares que não possuem confiança mútua através dos contratos inteligentes.

A rede de comunicação das redes elétricas inteligentes interconecta os diversos atores do sistema elétrico, como consumidores, concessionárias, distribuidoras, entre outros. Assim, um ataque à rede de comunicação pode se originar de qualquer dispositivo conectado à rede [Igure et al., 2006]. A interconexão dos atores do sistema ocorre tanto por enlaces de rede cabeada quanto por enlaces de rede sem-fio, para reduzir os custos de implantação e manutenção [Zhu et al., 2011]. Dessa forma, além dos diversos pontos de entrada para um atacante[2], os ataques às redes elétricas inteligentes passam a poderem ser realizados de duas maneiras: (1) o acesso direto a um dispositivo físico, como o medidor inteligente de uma residência; ou (2) através da comunicação sem-fio, devido à natureza compartilhada do meio de transmissão da tecnologia. Portanto, para dificultar ações maliciosas nesse ambiente hostil proveniente da introdução das redes de comunicações, deve-se levar em conta os requisitos de segurança para as redes elétricas inteligentes.

A **disponibilidade** refere-se ao tempo e à acessibilidade aos dispositivos da rede. Através de uma falha na rede ou um ataque intencional [Neuman e Tan, 2011], um servidor pode tornar-se indisponível, interrompendo assim os serviços providos por este e implicando danos em equipamentos eletrônicos ou perda financeira [Rahman et al., 2013]. Um nó malicioso, por exemplo, pode comprometer o medidor inteligente de uma residência para interromper o serviço de coleta de informações de energia. Ainda em serviços críticos que necessitam da troca de mensagens dentro de um tempo mínimo aceitável, um simples ataque DoS que degrade a qualidade da comunicação pode gerar o atraso necessário no envio dos pacotes para causar danos à infraestrutura da rede elétrica [Guimarães et al., 2013].

A **integridade** refere-se à consistência e à exatidão do dado durante todo seu ciclo de vida, ou seja, a confiança de que uma mensagem enviada pela fonte chegou em seu destino sem sofrer alterações indevidas no meio do caminho. Um exemplo de quebra de integridade é a modificação das informações de cobrança por uso da energia elétrica [Neuman e Tan, 2011]. Um atacante poderia, por exemplo, modificar o valor de consumo enviado para a distribuidora de energia, a fim de se beneficiar reduzindo o valor da cobrança do seu consumo. Um ataque de vulnerabilidade para a adulteração das informações pode impactar também a disponibilidade de outros dispositivos ou serviços da rede e, portanto, deve-se adotar métodos para impedir ataques à integridade.

A **privacidade** é outro ponto importante nas redes elétricas inteligentes. Por exemplo, os medidores inteligentes mantêm informações sobre a utilização de energia por parte do consumidor [McDaniel e McLaughlin, 2009], que descrevem partes de sua rotina como os horários em que o consumidor esteve em casa ou os cômodos que ele mais utiliza. É possível descobrir quais equipamentos elétricos um consumidor possui, em qual horário usou o chuveiro elétrico, ou qual é o nível de carga do veículo elétrico desse consumidor ao chegar em casa, podendo calcular possíveis destinos, entre outros dados pessoais. Esse conjunto de informações pode facilitar ações criminosas contra re-

[2]Atacante, entidade maliciosa e nó malicioso são termos usados intercaladamente como sinônimos neste capítulo.

sidências ou fornecer inteligência de negócios para os concorrentes [Hadley et al., 2010]. A cadeia de blocos pode proporcionar uma solução confiável para garantir a privacidade dessas informações.

A **autenticação** diz respeito à identificação de uma entidade, certificando que o indivíduo é quem realmente diz ser. A autenticação assegura a integridade das ações tomadas e mensagens compartilhadas na rede, uma vez que sistemas sensíveis podem sofrer ataque de injeção de dados resultando em dano a equipamentos, perda financeira ou, até mesmo, roubo de energia [Hadley et al., 2010]. No cenário de redes elétricas inteligentes com geração distribuída, a confiança entre os nós da rede é um desafio.

A **autorização** tem como objetivo liberar o acesso de um dispositivo a um limitado conjunto de políticas. Nas redes elétricas inteligentes, as políticas de autorização são responsáveis por garantir que os serviços, de leitura ou de gerenciamento, sejam executados apenas por um seleto grupo de dispositivos. É importante ressaltar que autenticação e autorização são conceitos diferentes, mas intrinsecamente relacionados. Enquanto a autenticação garante a identidade do usuário, a autorização, após a autenticação, concede acesso às funções específicas do seu nível de permissão. Uma rede sem políticas bem definidas é suscetível a ataques à privacidade. Por exemplo, uma política de acesso que permite a leitura de informações partindo de qualquer entidade autenticada [Yan et al., 2011] facilitaria ações criminosas. O mesmo vale para políticas de gerenciamento dos dispositivos da rede elétrica, um atacante (com autorização de gerenciamento, ou por falta de políticas bem definidas) pode desligar remotamente o fornecimento de energia de uma residência [Neuman e Tan, 2011].

A **auditoria** diz respeito à capacidade de averiguar a integridade dos dispositivos e o registro de suas ações na rede elétrica inteligente [Hadley et al., 2010]. Uma área que se beneficia do requisito da auditoria é a documentação de eventos incomuns. Esse registro é importante para análise e elucidação de eventuais falhas ou tentativas de ataques à rede elétrica, podendo atribuir uma pontuação para cada dispositivo de acordo com suas ações na rede [Wang e Lu, 2013].

O requisito de **não repúdio** é importante nas redes elétricas inteligentes para garantir que ações acordadas entre as partes não possam ser contestadas futuramente. Um ataque direcionado ao não repúdio pode se aproveitar de vulnerabilidades do dispositivo para alteração dos dados auditáveis e contestação de cobranças por consumo de energia elétrica [Neuman e Tan, 2011]. Em cenários de geração distribuída de energia, esse é um grande desafio devido às transações de compra e venda de energia ocorrerem sem um intermediador. O histórico de transações permite a validação e o estudo das operações, podendo criar estratégias para maximizar o lucro ou minimizar o gasto [Ramachandran et al., 2011].

A **confiança** diz respeito à crença entre os dispositivos da rede elétrica inteligente de que os dispositivos se comportam da forma esperada. Este requisito considera que os dispositivos da rede estão autenticados e autorizados de forma correta e suas ações são bem intencionadas [Wang e Lu, 2013]. Em um ataque direcionado à confiança, o atacante aproveita-se da crença dos outros dispositivos da rede para efetuar ações danosas como a injeção de informações incorretas que podem causar interrupções de energia em menor escala [Hadley et al., 2010].

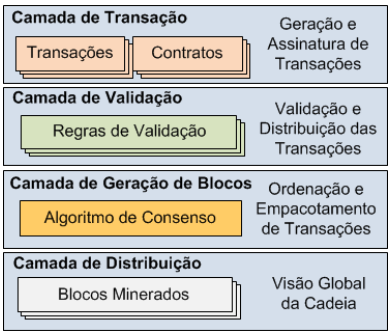
É evidente a necessidade de tecnologias para evitar ou reduzir possíveis falhas e ataques à rede. Através da tecnologia de cadeia de blocos, é possível atender aos requisitos de segurança, em especial nas redes de geração distribuída, nos quais não existe uma organização centralizadora para garantir a legitimidade das transações. Uma das características das cadeias de blocos é a garantia de imutabilidade dos dados contidos nos blocos. Assim, todas as ações dos usuários ficam gravadas na cadeia, formando um histórico incontestável. Através da auditoria dos blocos, é possível detectar falhas ou ataques e tomar as devidas ações para correção destes problemas, evitando a probabilidade de reocorrência no futuro. Os contratos inteligentes possibilitam a integridade, confiança e não repúdio, garantindo que as transações sejam concluídas apenas se ambas as partes atenderem aos requisitos acordados anteriormente, reforçados pela rede, e assegurando que, após finalizada, a transição não é desfeita.

4.3. Tecnologia de Cadeia de Blocos

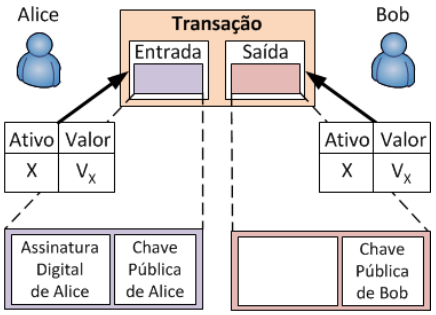
A tecnologia de cadeia de blocos consiste em uma rede par-a-par com uma estrutura de dados capaz de armazenar transações de forma ordenada e distribuída. Dessa forma, a tecnologia de cadeia de blocos é definida por dois elementos básicos, a estrutura de dados de encadeamento dos blocos e a rede par-a-par composta pelos nós participantes. O diferencial que a tecnologia de cadeia de blocos oferece em relação aos sistemas de dados distribuídos é a não necessidade da terceira entidade centralizadora, âncora de confiança, para garantir a segurança entre transações na rede [Nakamoto, 2008]. Ao se introduzir uma terceira entidade centralizadora gera-se um ponto único de falha que prejudica a segurança e a privacidade das transações realizadas, quando são considerados os conflitos de interesses entre as partes envolvidas. Assim, a tecnologia de cadeia de blocos tem sido amplamente empregada em diversos ramos de negócios, a fim de garantir disponibilidade, integridade, privacidade e não repúdio sem a necessidade de uma organização centralizadora controlando os dados.

A tecnologia de cadeia de blocos beneficia a **transparência** das transações, pois permite velocidades de liquidação próximas a tempo real e construindo a base para a auditoria, rastreabilidade e confiança entre participantes; a **confiança** entre os pares, pois elimina os intermediários e garante que a base de dados nos nós participantes converge para uma única versão coerente; a **eficiência** do sistema, já que, com a eliminação de intermediários, há a redução de custos com reconciliação e conformidade, assegurando a viabilidade de transacionar quantias menores; o **controle** e a **segurança** dos ativos transacionados ao reduzir os riscos de fraude e de liquidação, pois usa inerentemente funções criptográficas e evita a criação de pontos de centralização do controle [Dütsch e Steinecke, 2017].

A primeira geração de cadeia de blocos, representada pela *Bitcoin* [Nakamoto, 2008], foi idealizada para transferências monetárias entre nós em uma rede pública, que representam transações de pequenas quantidades de dados em uma rede hostil, em que os nós não confiam uns nos outros. Posteriormente, a segunda geração de cadeia de blocos, representada pela rede *Ethereum*, propôs que a estrutura de dados da cadeia de blocos fosse usada para representar transações mais complexas que executam um determinada aplicação, os chamados contratos inteligentes. Contratos inteligentes são estruturas de computação de mensagens de objeto confiável, autoexecutáveis. Um



(a) A tecnologia de cadeia de blocos em camadas.



(b) Exemplo de transação de transferência de ativos.

Figura 4.3. Elementos que compõem a tecnologia de cadeia de blocos. a) Divisão da cadeia de blocos em camadas. As transações dos usuários são geradas nas camada de transação, validadas pela rede na camada de validação, inseridas em blocos na camada de geração de blocos e os blocos são distribuídos na camada de distribuição. b) Transação típica em que Alice transfere um ativo da sua posse para Bob. A transação é identificada pelas chaves públicas e validada pela assinatura digital de Alice. Adaptado de [de Oliveira et al., 2018].

contrato inteligente é determinístico. Isso quer dizer que, para uma mesma entrada, sempre produzirá uma mesma saída. Caso contrário, um contrato não determinístico geraria resultados aleatórios para os diferentes nós da rede [Christidis e Devetsikiotis, 2016], impossibilitando o alcance de uma consenso sobre dados que devem ser armazenados na cadeia de blocos. A característica determinística dos contratos inteligentes é o que garante a convergência da visão global da rede. O contrato inteligente reside na cadeia de blocos e, como tal, seu código pode ser inspecionado por todos os participantes da rede. Como todas as interações com um contrato ocorrem via mensagens assinadas, é possível rastrear todos os participantes envolvidos na operação do contrato. Sendo assim, a aplicação de contratos inteligentes possibilitou a automatização de regras executáveis com o consentimento das várias partes envolvidas [Wood, 2014].

A segurança oferecida pela tecnologia de cadeia de blocos reside em todos os nós participantes da rede par-a-par acessarem uma réplica idêntica da cadeia de blocos armazenada localmente, mesmo em um ambiente de desconfiança mútua entre participantes. Portanto, é necessário adotar mecanismos de validação e de consenso para realizar a distribuição e a réplica coerente dos dados e adotar mecanismos de assinatura digital e resumos criptográficos para garantir a auditoria distribuída sobre as transações executadas na rede.

Para que as transações sejam efetivadas como parte da cadeia, as transações são processadas por quatro camadas, transações, validação, geração de blocos e distribuição, mostradas na Figura 4.3(a) [de Oliveira et al., 2018].

A camada de **transações** representa a geração da informação que se deseja armazenar em uma cadeia de blocos. Estas informações são por natureza não editáveis, isto é, contratos, transferências bancárias, compra e venda etc. Assim como em bancos de dados tradicionais, na cadeia de blocos, as transações seguem a semântica ACID (Atomicidade, Consistência, Isolamento e Durabilidade) [Dinh et al., 2017]. Além disso, a interação direta dos usuários com a cadeia ocorre na camada de transações. O controle de acesso à rede acontece a partir da concessão de um par de chaves assimétricas para que o

usuário possa assinar digitalmente uma transação na rede. Os usuários são identificados somente pelas chaves públicas geradas ao ingressarem na rede, permitindo uma pseudo-anonimização dos participantes [Nakamoto, 2008]. Vale ressaltar que, por padrão, não há um esquema para autenticação de usuários, já que os usuários são apenas identificados por suas chaves públicas, não há um mecanismo que relacione uma chave pública com uma entidade conhecida, como realizado por uma infraestrutura de chaves públicas (*Public Key Infrastructure* - PKI). Assim, o usuário assina suas transações com a chave privada e pode ser endereçado na rede por meio da chave pública. Seguindo critérios definidos na rede, como organização e linguagem de codificação pré-estabelecidas para a elaboração da transação e a assinatura, o usuário transmite a transação para todos os nós vizinhos, conforme mostrado na Figura 4.3(b). A camada de **validação** é determinada pela verificação das transações. Os nós vizinhos são responsáveis por verificar se as transações seguem os critérios predeterminados pela rede. A validação analisa se a transação obedece a todas as regras da rede no desenvolvimento da cadeia de blocos. Por exemplo, na *Bitcoin*, a regra fundamental para executar uma transação é a disponibilidade da quantia enviada em posse da chave pública que a emite. As transações são somente transmitidas para os nós seguintes se forem consideradas válidas pelos nós que realizam o processo de validação das novas transações. Caso a transação descumpra algum dos critérios da rede, deve ser descartada e não passada adiante.

Na camada de **geração de blocos**, as transações validadas na camada de validação estão disponíveis para a formação do novo bloco da cadeia, *pool* de transações válidas. Essas transações são coletadas, ordenadas e empacotadas em um bloco candidato a ser inserido na cadeia, com a estampa de tempo correspondente (*timestamp*). A geração do bloco é chamada de processo de mineração, no qual o nó minerador toma a responsabilidade de gerar o bloco e introduzi-lo efetivamente na cadeia. Contudo, a escolha do nó minerador depende diretamente do mecanismo de consenso empregado na rede. Cada bloco minerado contém uma referência ao bloco antecessor, formando assim o encadeamento de blocos. Essa referência é feita através de resumos criptográficos (*hash*) [Nakamoto, 2008, Chicarino et al., 2017, Christidis e Devetsikiotis, 2016], como ressaltado na Figura 4.4. O bloco inicial da cadeia é o bloco *genesis*, que armazena um valor de resumo criptográfico determinado na pela regra de formação da cadeia de blocos. O bloco B_n, com transações válidas, possui junto ao seu conteúdo o resumo criptográfico do bloco anterior B_{n-1}. O conteúdo completo do bloco B_n será usado para gerar o resumo criptográfico que será incluído como referência no próximo bloco B_{n+1}. Como o algoritmo que computa o resumo criptográfico é unidirecional, é improvável a recuperação dos dados originais a partir do resumo gerado, assim como é improvável a geração de um novo conteúdo que gere o mesmo resumo. Isso garante a integridade dos dados na cadeia. Caso haja uma mudança indevida no conteúdo de um dos blocos armazenados em um nó, tal mudança é evidenciada pela alteração do valor do *hash* desse bloco que, por sua vez, é propagada para todos os demais blocos da cadeia, devido ao encadeamento de valores dos *hashes* do blocos seguintes, a exemplo do $Bloco_{n-2}$ no *Nó 2* da Figura 4.4.

Na camada de **distribuição**, o bloco minerado é adicionado à estrutura de dados da cadeia de blocos de cada nó da rede par-a-par. Destaca-se a necessidade da réplica atualizada da cadeia localmente para que seja alcançado o consenso na rede. Assim sendo, as transações associadas aos blocos são executadas para atualizar a visão global da cadeia.

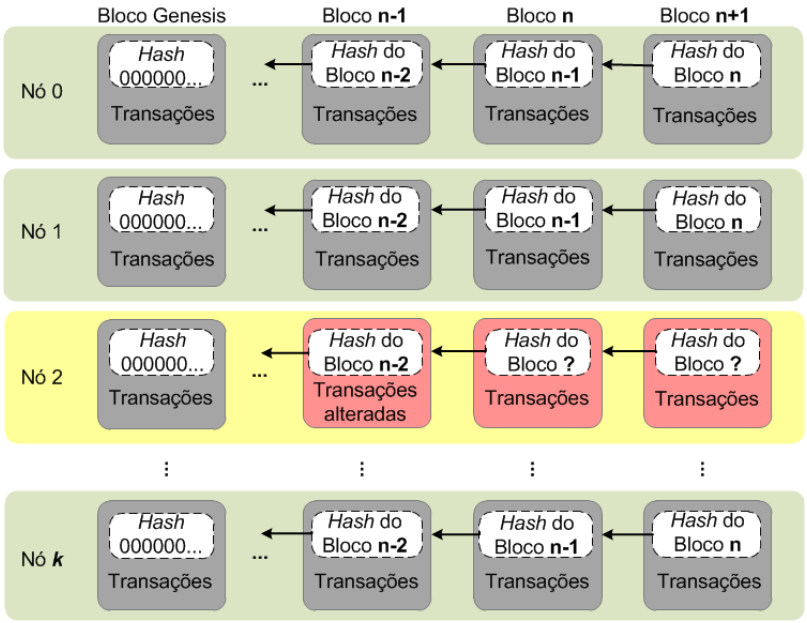


Figura 4.4. Visão esquemática da estrutura de dados em uma cadeia de blocos. O bloco *genesis* representa o primeiro bloco da cadeia. Cada bloco tem o resumo criptográfico do bloco anterior, gerando um encadeamento de resumos criptográficos. A alteração de um bloco gera a inconsistência de todos os blocos seguintes da cadeia.

Ressalta-se que a execução das transações determina uma mudança de estado global na cadeia, seja a transferência de ativos, seja a execução de um contrato inteligente. Contudo, o encadeamento de um novo bloco só ocorre nos nós adjacentes se o resumo criptográfico do bloco minerado estiver correto. Essa verificação é feita pela comparação entre o conteúdo do bloco e o resumo criptográfico apresentado. Caso contrário, o bloco minerado é descartado. Se todos os nós da rede possuírem o mesmo estado global da cadeia, com o mesmo conteúdo e blocos organizados na mesma ordem, os nós estão em consenso. Ao atingi-lo, todos os nós passam a ter acesso à mesma informação. A visão global distribuída da cadeia permite a disponibilidade e a auditoria das informações.

4.3.1. Taxonomia de Plataformas de Cadeia de Blocos

Christidis e Devetsikiotis classificam as cadeias de blocos segundo os aspectos de controle de acesso ao conteúdo da cadeia e quanto às permissões sobre as funções que os nós da rede exercem [Christidis e Devetsikiotis, 2016]. Contudo, não há um consenso sobre uma definição formal de taxonomia para classificar as redes de cadeia de blocos. Outros trabalhos classificam as redes como pública, privada, permissionada e híbrida [Pilkington, 2016, Gupta e Sadoghi, 2018]. Apesar dos conceitos de rede pública e privada serem bem conhecidos e antagônicos, os conceitos de rede permissionada e híbrida não são autoexplicativos. Neste capítulo, adota-se a taxonomia em diferentes tipos de visão da rede, *pública não permissionada*, *pública permissionada*, *privada não permissionada* e *privada permissionada*, como evidenciado na Figura 4.5.

Redes públicas e privadas se distinguem em relação ao controle de acesso à rede e ao conteúdo da cadeia, visto que uma vez que o nó é participante da rede par-a-par, o nó acessa a sua réplica da cadeia armazenada localmente. Em uma rede pública, de conteúdo

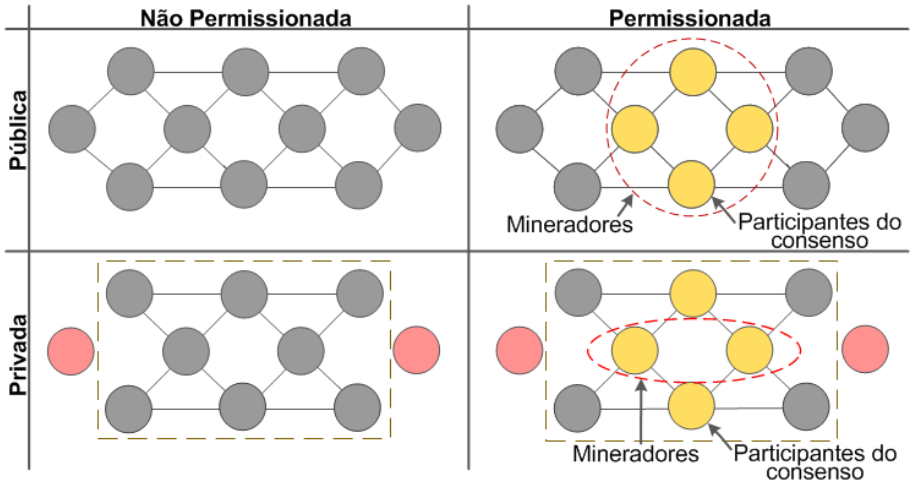


Figura 4.5. Taxonomia aplicada a redes de cadeia de blocos. A classificação entre pública e privada relaciona-se com a participação de nós na rede. A classificação entre permissionada e não permissionada relaciona-se com o papel desempenhado pelos nós da rede nos mecanismos de consenso e de geração de novos blocos.

aberto, não há qualquer mecanismo de controle de acesso e os nós podem ingressar e deixar a rede sem qualquer prejuízo para o mecanismo de consenso ou para a geração de novos blocos. Em redes privadas, em que o conteúdo é fechado, há medidas de controle de acesso e apenas nós autorizados podem acessar a rede par-a-par e ter acesso ao conteúdo da cadeia. Em paralelo, redes permissionadas e não permissionadas se diferenciam pelo critério de atividades desempenhadas na rede. Em redes não permissionadas, todos os nós desempenham o mesmo papel na rede, podendo gerar transações, competirem na mineração de blocos e participarem do mecanismo de consenso. Em contraste, nas redes permissionadas, os nós podem possuir papéis distintos, de acordo com a necessidade da aplicação, como por exemplo, uma aplicação em que todos os nós podem criar transações, um grupo de nós da rede é responsável por realizar o consenso e apenas um subgrupo é autorizado a minerar novos blocos.

Em **redes públicas não permissionadas** de cadeias de blocos há desconfiança mútua entre os usuários da rede e, por isso, os mecanismos de consenso são rígidos. A fim de evitar ataques de personificação (*Sybil Attack*) [Douceur, 2002], o consenso em redes públicas não permissionadas é oneroso, exigindo-se a resolução de um desafio computacional como prova da participação no consenso e, assim, evitando que um nó apresente diversas identidades. O incentivo aos nós a participarem desse mecanismo de consenso consiste em um incentivo econômico dado aos nós mineradores na forma de criptomoeda. Isso é justificável pelas características principais de uma rede pública não permissionada, tais como conteúdo aberto e igualdade entre os nós. Todo nó pode ingressar e sair da rede. O participante da rede gera um par de chaves criptográficas para assinar e realizar transações quando ingressa pela primeira vez na rede. Além disso, qualquer nó pode ser um minerador e fazer parte do mecanismo de consenso da rede. Os problemas associados às redes públicas não permissionadas estão relacionados à escalabilidade e ao tempo efetivo desde a emissão da transação até a execução na cadeia. Essas redes constituem ambientes colaborativos e, portanto, dependem do comportamento benigno dos nós. Além disso, a

competição entre os nós mineradores pelas recompensas da geração do bloco torna o processo mais lento e custoso. *Bitcoin* e *Ethereum* são exemplos de plataformas que oferecem configuração de rede pública não permissionada.

As **redes públicas permissionadas** foram desenvolvidas para aplicação de mecanismos de consenso menos custosos em redes públicas. A diferença entre as redes públicas não permissionadas e as permissionadas é a desigualdade de atuação dos nós na rede. Em uma rede pública permissionada, todos os dados são disponíveis para auditoria pública e não se restringe a entrada de novos nós. Contudo, um nó só participa da rede após a verificação adequada de sua identidade e, assim, alocam-se as permissões que determinam quais atividades o nó pode executar na rede. Este tipo de rede é utilizada para gerenciar transações entre empresas ou em processos que envolvem várias entidades, permitindo que somente alguns nós de cada entidade fiquem responsáveis pela geração de blocos, aplicando mecanismos de consenso mais eficientes e sustentáveis.

As **redes privadas não permissionadas** se diferenciam das redes púbicas por restringirem a entrada de nós e, portanto, só fornecem a réplica da cadeia a nós identificados por uma chave pública autorizada. Existe uma ou um conjunto de instituições que determinam quem são os nós autorizados a participarem da rede. Destaca-se que o conceito de rede privada se delimita ao controle de acesso aos dados da cadeia. Os nós que participam da rede têm funções iguais e exercem a mesma importância na rede, pois não se determinam permissões diferenciadas aos nós da rede. Uma vez autorizado a participar da rede, o nó pode gerar transações, gerar blocos e participar do consenso. Esta característica é interessante às aplicações em que nós, mesmo autorizados a participarem da rede, oferecem um comportamento hostil. Nesses casos, empregam-se mecanismos de consenso tolerantes a falhas Bizantinas (*Byzantine-Fault Tolerant* - BFT), exigindo que todos os nós participem do consenso. O *Hyperledger Fabric* é um exemplo de plataforma com opção de configuração de redes privadas não permissionadas.

As **redes privadas permissionadas** oferecem a oportunidade de aplicações de mecanismos de consenso mais eficientes e menos custoso em termos de processamento. A característica privada limita a entrada e permanência de nós indesejáveis na rede. As permissões possibilitam configurar diferentes papéis para os nós participantes da rede como, por exemplo, oferecer flexibilidade para aplicações permitirem que apenas alguns nós façam parte do processo de consenso e apenas um subconjunto desses nós possam gerar o próximo bloco. As aplicações de cadeia de blocos em redes elétricas inteligentes podem aproveitar esta flexibilidade para poupar processamento e energia de dispositivos simples, como medidores inteligentes, impedindo a participação desses nós em mecanismos de consenso e geração de blocos. Dispositivos com mais processamento e segurança exercem essas atividades na rede. A *Parity* e a *MultiChain* são exemplos de plataformas que permitem configurações de redes privadas permissionadas.

4.3.2. Plataformas para Desenvolvimento de Cadeias de Blocos

A **Bitcoin**[3] é uma plataforma para desenvolvimento de aplicações para cadeias de blocos proposta por Satoshi Nakamoto [Nakamoto, 2008]. Ela permite a criação de redes públicas não permissionadas nas quais um nó pode participar e exercer qualquer

[3]Disponível em https://bitcoin.org/.

função na rede. Essa característica faz com que certos critérios de confiabilidade sejam exigidos de seus participantes, por isso esta plataforma utiliza a Prova de Trabalho como mecanismo de consenso. Antonopoulos define que as funções exercidas por cada nó podem ser divididas em: (i) cliente de referência que possuem todas as funções, (ii) nó completo que não exerce a função de mineração, (iii) nó leve que apenas interage com a rede para enviar e receber transações e (iv) nó minerador que é apenas responsável por executar a prova de trabalho [Antonopoulos, 2014].

A Bitcoin permite aos seus usuários criarem regras que restringem o modo como os valores enviados através das transações são gastos. Apenas usuários capazes de satisfazer essas regras podem ter acesso aos valores a eles relacionados. Para expressar essas regras, a Bitcoin implementa uma linguagem denominada *Script*. Um usuário ao criar uma transação acrescenta um código executável que define as condições para gastar aquele valor. Ao utilizar esse valor para criar outra transação, deve-se acrescentar outro código executável que, quando executado juntamente ao anterior, resulte em uma execução bem sucedida. A *Script* é uma linguagem baseada em pilha e não é do tipo *Turing* completa, ou seja, ela não possui estruturas de laços. Essa restrição se faz necessária para impedir que ataques de negação de serviço sejam feitos contra um nó através da criação de laços infinitos. A regra mais utilizada para criar transações é a que atrela um dado valor à chave pública de outro usuário e, assim, garante que apenas ele possa gastá-lo. Em suas primeiras versões, a Bitcoin permitia somente a criação de soluções financeiras baseadas na troca de valores. No entanto, as versões mais recentes da plataforma permitem o envio de dados no lugar de apenas valores. Aplicações não financeiras foram propostas sobre a Bitcoin, como o *FairAccess* [Ouaddah et al., 2016], cujo objetivo é prover controle de acesso a recursos, utilizando a estrutura das transações como meio para enviar as permissões. A linguagem *Script* é usada para expressar as condições de acesso aos recursos.

A **MultiChain**[4] é uma plataforma para desenvolvimento de aplicações utilizando cadeias de blocos, cujo projeto foi desenvolvido baseado na implementação da Bitcoin. A plataforma é totalmente compatível com o protocolo e capaz de funcionar como um nó da rede Bitcoin. A MultiChain permite a criação de redes privadas permissionadas [Greenspan, 2015], o que exige a participação de um administrador na rede, que é responsável por permitir a entrada e gerenciar as permissões dadas aos nós. As permissões variam desde capacidade de executar buscas na cadeia, até permissão para um nó ser minerador ou tornar outro nó administrador. O papel de administrador é concedido ao nó responsável por criar o bloco *gênesis* da rede. Sempre que um administrador concede ou revoga uma permissão de um nó, esse evento fica registrado na cadeia através de uma transação especial. Dessa forma, todos os demais nós são capazes de verificar quais permissões estão vinculadas a cada chave pública da rede.

Usuários de uma rede MultiChain são capazes de criar e gerenciar seus próprios ativos, através de uma transação especial, chamada "Transação Gênesis", que contém os metadados necessários para registrar o canal na cadeia de blocos [Greenspan, 2015]. Para tanto, o usuário deve possuir a permissão concedida por um administrador da rede. Após a criação do canal, o criador assume o papel de administrador, podendo decidir quem pode

[4]Disponível em https://www.multichain.com/.

enviar, receber e criar novos ativos naquele canal. A criação de canais privados permite que apenas os usuários que tenham interesse em um certo tipo de ativo obtenham acesso ao canal e às informações presentes na cadeia relativa ao ativo.

Ao contrário da plataforma Bitcoin, a MultiChain não possui suporte à criação de regras através da linguagem *Script*. Logo, um usuário é apenas capaz de enviar e receber ativos pela rede. No entanto, quando utilizada para se conectar à rede Bitcoin, a plataforma fornece suporte à criação de regras através da *Script*. O mecanismo de consenso oferecido pela MultiChain permite que, através da configuração de parâmetros no momento da criação da rede, o consenso funcione como uma prova de trabalho ou que cada minerador alterne na criação de um novo bloco, sem a necessidade de haver competição. Isso permite que haja maior flexibilidade na criação de novas redes e elimina os altos custos computacionais da prova de trabalho. A manutenção da prova de trabalho permite a retrocompatibilidade com a rede Bitcoin.

A **Ethereum**[5] é uma plataforma criada e mantida pela *Ethereum Foundation* e representa o início da segunda geração das plataformas de cadeias de blocos [Buterin et al., 2013]. Essa plataforma foi desenvolvida com o intuito de possibilitar a realização de novas transações além de somente as transações de trocas de posse de ativos. A plataforma possibilita a criação de aplicações que, através da invocação de contratos inteligentes, podem exercer diferentes funções na redes. A utilização de contratos inteligentes estabeleceu uma nova geração de aplicações de cadeias de blocos, pois o desenvolvedor de aplicações na plataforma Ethereum é capaz de criar aplicações que interagem com a rede Ethereum de maneira transparente ao usuário final. Essas aplicações usufruem de todas as vantagens das cadeias de blocos, porém sem a necessidade de conhecimento específico sobre a tecnologia. A principal proposta da plataforma Ethereum é a criação de aplicações que utilizam a cadeia de blocos da própria rede Ethereum. Contudo, a plataforma possibilita a criação de redes públicas não permissionadas e, assim como na rede Bitcoin, a rede Ethereum possui sua própria moeda corrente, o *Ether*. A plataforma possibilita a realização de transações financeiras simples, sem a necessidade da utilização de contratos complexos. O mecanismo de consenso utilizado pela Ethereum é prova de trabalho, mas a versão implementada por esta plataforma impede a utilização de *hardware* especializado para a otimização do tempo de resolução do desafio computacional. Isso garante uma melhor distribuição das capacidades de mineração através da rede [Buterin et al., 2013]. Segundo seus desenvolvedores, em uma versão futura, será acrescentado suporte à utilização da prova de participação (*Proof of Stake*) como uma alternativa à prova de trabalho.

A **Parity**[6] inicialmente foi proposta como um meio de interação com a rede Ethereum. Assim, a Parity possui todas as características presentes na plataforma Ethereum. Além disso, a Parity agrega uma ferramenta para desenvolvimento e depuração de aplicações baseadas em contratos inteligentes utilizando a linguagem nativa da Ethereum, a *Solidity*, além de possuir suporte para a utilização de carteiras digitais e para o gerenciamento de chaves de usuários. A Parity estende as funções padrão da plataforma Ethereum permitindo a criação de redes privadas permissionadas, que possuem compati-

[5]Disponível em https://www.ethereum.org/.
[6]Disponível em https://www.parity.io/.

bilidade com a rede Ethereum e permitem que aplicações desenvolvidas para a Ethereum funcionem normalmente na rede privada Parity. Para a criação de redes privadas, a Parity fornece suporte ao mecanismo de consenso original da Ehtereum, a prova de trabalho, porém também possui uma implementação do mecanismo baseado na prova de autoridade. Nessa implementação, os nós mineradores se revezam para criar blocos dentro de uma janela de tempo determinada no momento da criação da rede.

O **Hyperlegder**[7] é um projeto formado por diversas corporações, como a Linux Foundation, a IBM e a Intel, que visam desenvolver soluções para promover a aplicação comercial e incentivar os estudos sobre a tecnologia de cadeia de blocos [Greve et al., 2018]. Um dos desdobramentos do projeto é a plataforma **Hyperledger Fabric**, proposta pela IBM [Cachin, 2016]. A plataforma permite a criação de soluções empresariais baseadas no uso de cadeias de blocos privadas permissionadas. A arquitetura modular permite que serviços específicos de rede sejam distribuídos entre nós especializados, o que permite que sejam oferecidos altos graus de confiabilidade, resiliência, flexibilidade e escalabilidade [Greve et al., 2018]. Os nós da rede podem fornecer serviços de validação, consenso, controle de credenciais e armazenamento. Nós que fornecem serviços de validação são responsáveis por conectar clientes aos serviços de consenso, através da emissão de transações. Esses nós não possuem capacidade para executar as transações, apenas de verificá-las. O serviço de consenso é oferecido pelos nós responsáveis por executarem o mecanismo de consenso, que nesta plataforma consiste de uma implementação do modelo prático de tolerância a falhas bizantinas (*Practical Byzantine Fault Tolerance* - PBFT). Esses nós também são responsáveis por validar as transações. O serviço de consenso também é responsável por atualizar o estado da cadeia de blocos. O controle de credenciais é o serviço responsável por criar os certificados que identificam os usuários da rede. Através dessa identificação, é possível exercer o controle de permissão da rede. Ao realizar uma transação, um usuário deve obrigatoriamente se identificar através de seu certificado. O serviço de armazenamento é oferecido por todos os nós da rede, com exceção dos nós que realizam controle de credenciais. O serviço consiste em armazenar uma cópia da cadeia de blocos e permitir a realização de consultas. As informações da cadeia de blocos são armazenados em um banco de dados não relacional e apenas as referências aos dados ficam na cadeia. Esse procedimento torna a rede escalável, pois reduz o espaço necessário para armazenar uma cópia local da cadeia e reduz o tempo para realizar buscas por informações. A Hypeledger Fabric possui suporte para a criação de contrato inteligente, denominado *ChainCode*. Esses contratos ficam armazenados na cadeia e, quando invocados, são executados pelos nós que implementam o serviço de consenso. A linguagem utilizada para o desenvolvimento dos contratos é a GO[8]. Para adicionar um contrato a uma cadeia, um usuário executa uma transação, ficando assim também registrado o momento da criação do contrato.

A plataforma **R3 Corda**[9] foi desenvolvida com o propósito de se diferenciar das demais plataformas de desenvolvimento de cadeias de blocos, já que propõe o isolamento dos dados de seus usuários. O isolamento ocorre através da criação de canais de comunicação entre usuários que estejam interessados em realizar transações entre si, formando

[7]Disponível em http://www.hyperledger.org.
[8]Disponível em https://golang.org/.
[9]Disponível em http://www.corda.net/.

pequenas cadeias de blocos acessíveis apenas entre eles. A Corda possibilita a criação de cadeias privadas permissionadas. Para que um usuário possa fazer parte de uma rede, é necessário que ele possua os certificados gerados pela entidade responsável por administrar aquela rede. Uma vez na rede, o usuário pode se associar a canais com participantes com os quais deseja realizar transações. Contratos inteligentes são tratados de maneira diferente pela Corda, uma vez que esse contratos são disponibilizados na forma de aplicativos (*Cordapps*) que podem ser instalados em nós específicos da rede, sem a necessidade de estarem disponíveis em todos os nós. Esse aplicativos são responsáveis por realizarem a interação dos usuários com a rede. Para o desenvolvimento das aplicações, a Corda oferece bibliotecas nas linguagens *Java* e *Kotlin*, porém qualquer linguagem compatível com a máquina virtual Java (JVM) pode ser utilizada. O desenvolvedor pode criar uma interface gráfica e disponibilizá-la através de um servidor *web* integrado à sua aplicação. Isso torna a utilização das aplicações mais amigáveis aos usuários. Na versão atual, versão 3.0, a Corda disponibilizada três opções para mecanismos de consenso, o *Raft*, o *bftSMaRt* e o consenso personalizado. O *Raft* é um protocolo de consenso por votação simplificado, mas que não resiste a falhas bizantinas [Ongaro e Ousterhout, 2014]. O *bftSMaRt* consiste de uma implementação do modelo prático de tolerância a falhas bizantinas [Bessani et al., 2014]. A aplicação (*Cordapp*) pode fornecer uma implementação própria de um mecanismo de consenso personalizado. O armazenamento dos dados da rede é feito através da utilização de bases de dados externas à cadeia em cada nó. As bases de dados armazenam apenas as transações em que eles tiveram participação, restringindo os acesso aos dados apenas a usuários que utilizam os canais aos quais o nó pertence.

4.4. Controle Distribuído e Consenso em Cadeias de Blocos

Os nós participantes de um sistema que utilize cadeia de blocos devem possuir uma visão global comum sobre a rede, para que não existam divergências nas cópias das cadeias de blocos presentes em cada nó. Os blocos da cadeia são compostos por uma sequência de transações a serem executadas. Antes de serem executadas, os nós precisam alcançar um consenso, concordando com as transações inseridas no bloco e com a ordem em que serão executadas. O consenso consiste em regras para validação e difusão de transações e blocos, resolvendo potenciais conflitos [Xu et al., 2017], e alcançando uma consistência eventual da informação presente na rede. Ao se alcançar o consenso, garante-se a integridade, a consistência e a imutabilidade da cadeia de blocos.

O consenso é alcançado de forma distribuída, eliminando a necessidade de um agente central intermediário confiável. O tipo de mecanismo de consenso utilizado depende do tipo de rede de cadeia de blocos e do tipo de vetor de ataque esperado. São duas as principais classes de mecanismos de consenso: protocolos probabilísticos de consistência eventual e protocolos baseados em votação por maioria [Christidis e Devetsikiotis, 2016, Cachin e Vukolic, 2017]. Nos mecanismos baseados em consistência eventual, não é necessário saber o número de participantes disponíveis no consenso e eventualmente existe convergência sobre a cadeia de blocos, com base na disseminação da informação sobre o que cada participante enxerga como verdade. Já nos protocolos baseados em votação, é necessário conhecer todos os participantes do mecanismo. Dessa forma, consenso baseado em consistência eventual é adequado para cadeias de blocos públicas, enquanto os baseados em votação são mais adequados para ca-

deias de blocos privadas [Christidis e Devetsikiotis, 2016, Cachin e Vukolic, 2017]. Dentre os mecanismos de consenso utilizados nas redes de cadeia de blocos públicas estão as provas de trabalho (*Proof of Work* - PoW), de participação (*Proof of Stake* - PoS) e de capacidade (*Proof of Capacity* - PoC). Em redes de cadeia de blocos privadas, a necessidade por mecanismos de consenso custosos em termos computacionais, como a PoW, é reduzida [Christidis e Devetsikiotis, 2016] e, portanto, outros mecanismos de consenso podem ser utilizados, como a prova de autoridade (*Proof of Authority* - PoA), o protocolo prático de tolerância a falhas bizantinas (*Practical Byzantine Fault Tolerance* - PBFT) [Castro e Liskov, 1999] e os algoritmos Paxos [Lamport, 2001], Raft [Ongaro e Ousterhout, 2014] e Ripple [Schwartz et al., 2014].

4.4.1. Prova de Trabalho – PoW

Em redes públicas, tais como *Bitcoin* e *Ethereum*, uma única entidade pode participar da rede com múltiplas identidades para influenciar a votação sobre a validação de um determinado bloco. A consequência imediata dessa possibilidade é o controle da rede por uma minoria [Christidis e Devetsikiotis, 2016]. Para desestimular essa prática, Nakamoto [Nakamoto, 2008] propõe o uso de um mecanismo de consenso denominado Prova de Trabalho (*Proof of Work* - PoW). Na PoW, para que um bloco seja inserido na cadeia de blocos, os nós mineradores devem resolver um desafio não trivial, que exige grande poder de processamento para ser resolvido, mas cujo resultado pode ser facilmente verificado por qualquer nó da rede que seja participante do consenso. O bloco candidato a ser inserido na rede é composto pelas transações que foram submetidas, mas ainda não foram validadas. Quando o desafio é solucionado, o bloco é minerado, isto é, inserido na cadeia de blocos global, e as transações são validadas e executadas. O nó minerador recebe uma recompensa por ter empenhado seu poder computacional para a resolução do desafio.

O desafio computacional exigido na PoW consiste em encontrar um número aleatório (*nonce*) que faz com que o resumo criptográfico (*hash*) do bloco tenha o número esperado de zeros iniciais para a dificuldade definida para aquela rede [Nakamoto, 2008]. O *nonce* está contido no cabeçalho do bloco, juntamente com o resumo criptográfico do bloco anterior. Quanto maior o número de zeros iniciais requeridos, mais complexo é o desafio computacional. Na *Bitcoin*, a rede ajusta a dificuldade do desafio a cada 2.016 blocos para levar em consideração mudanças no poder de processamento dos nós e para garantir que os blocos sejam gerados a uma taxa constante [Christidis e Devetsikiotis, 2016] e, atualmente, são exigidos resumo criptográfico que iniciam com 8 zeros.

Ao resolver o desafio computacional, o nó gera uma prova de trabalho e pode adicionar o novo bloco à cadeia de blocos, recebendo a recompensa pela resolução do desafio. Esse bloco é disseminado através da rede para que os outros nós possam adicioná-lo a suas cópias da cadeia de blocos. Os outros nós que estavam trabalhando para solucionar o mesmo desafio param de tentar, uma vez que não haverá mais recompensa caso resolvam o desafio após o bloco ter sido minerado [Nakamoto, 2008]. Além disso, os blocos que os outros nós estão tentando minerar agora referenciam o resumo criptográfico do bloco errado e podem ser compostos por transações que já foram mineradas, isto é, que estão no bloco recentemente inserido na cadeia pelo nó vencedor. Ao desistirem e aceitarem que o desafio foi solucionado por um minerador vencedor, o consenso é alcançado. Na PoW, é possível que vários nós reivindiquem o próximo bloco a ser adicionado à cadeia.

Isso ocorre porque a PoW é um mecanismo de consenso probabilístico, no qual cada nó tem uma determinada probabilidade de concluir o desafio computacional antes de todos os outros nós da rede. Quando mais de um nó minera um bloco, ocorre uma ramificação da cadeia de blocos. Na PoW, a ramificação que carregar a maior quantidade de trabalho deve ser seguida. A ramificação da cadeia de blocos que crescer primeiro, através da inserção de novos blocos será adotada como a cadeia correta, levando à poda dos outros ramos [Nakamoto, 2008]. Isso permite que a rede alcance novamente o consenso sobre a ordem de ocorrência dos eventos e, então, obtenha uma visão global da cadeia consistente.

Apesar de o mecanismo tender probabilisticamente à convergência, a Prova de Trabalho apresenta desvantagens como a crítica à sustentabilidade do processo de mineração, em que há um gasto exacerbado de energia para criação de um bloco. Além disso, é observada uma alta latência para alcançar o consenso na rede. Como consequência, há uma baixa vazão na quantidade de transações validadas no tempo. Além disso, o mecanismo de Prova de Trabalho pode ser comprometido, teoricamente, por um usuário que controle pelo menos mais que 50% dos recursos computacionais da rede. Apesar da PoW desestimular uma entidade única a possuir diversas identidades na rede devido ao custo computacional para gerar a prova de trabalho, um grupo de nós mineradores pode compartilhar recursos para gerar blocos mais rapidamente, distorcendo a natureza descentralizada da rede. Atualmente, quatro aglomerados de poder computacional, *BTC.com*, *SlushPool*, *AntPool* e *BTC.TOP*, são responsáveis por mais de 50% da taxa de resumos criptográficos gerados na *Bitcoin*[10].

4.4.2. Prova de Participação – PoS

A Prova de Participação (*Proof of Stake* - PoS) é uma alternativa ao alto custo computacional da Prova de Trabalho para alcançar o consenso na rede, preservando a natureza descentralizada da rede pública. Na PoW, a probabilidade de um nó conseguir minerar um bloco depende do poder computacional de cada nó. Já na PoS, essa probabilidade passa a depender da participação dos nós na rede. Os nós mineradores precisam encontrar um valor de resumo criptográfico menor ou igual a um valor alvo para que possam minerar um bloco. A dificuldade para encontrar esse resumo criptográfico é inversamente proporcional à riqueza acumulada (*coin age*) daquele nó, definida como a quantidade de recursos do nó multiplicada pelo período em que o nó reteve aquele recurso. Por exemplo, se Bob recebeu 10 recursos de Alice e manteve a posse desses recursos por 90 dias, Bob acumulou uma riqueza igual a 900 $recursos \times dias$. Quando Bob gasta esses 10 recursos provenientes de Alice, ele consumiu, ou destruiu, o valor de riqueza acumulada devido a esses 10 recursos. A consequência imediata de se utilizar o conceito de riqueza acumulada na PoS é atribuir ao nó com maior participação, riqueza acumulada, a oportunidade de gerar o próximo bloco [Tschorsch e Scheuermann, 2016].

Na PoS, é necessário que cada transação possua um campo de marcação do tempo de geração para que o valor da riqueza acumulada seja calculado. O conjunto de nós que deseja atuar como minerador precisa necessariamente bloquear seus recursos por um determinado tempo. Isso é feito através da construção de um bloco especial, *coinstake*, no qual o proprietário do recurso emite uma transação para si mesmo, adicionando uma deter-

[10]Dados disponíveis em https://www.blockchain.com/pt/pools.

minada taxa de transação como recompensa [Tschorsch e Scheuermann, 2016]. No momento em que o minerador paga a si mesmo, o valor da sua riqueza acumulada (*coin age*) é consumido. Dessa forma, na próxima rodada de mineração, outros nós têm a chance de conseguir minerar o novo bloco. A primeira entrada do bloco *coinstake* é composta por um *kernel* que deve obedecer a um protocolo de geração de resumos criptográficos específico da rede. As tentativas de geração do resumo criptográfico correto ocorrem a uma taxa de uma tentativa por unidade de riqueza acumulada. Assim, quanto mais recursos o nó disponibilizar para tentar encontrar o *kernel* correto, maior será a sua chance de reivindicar a oportunidade para minerar o próximo bloco. Por exemplo, se Alice possui uma riqueza acumulada de 100 $recursos \times dias$, para a qual se espera a geração de um *kernel* em 2 dias, então Bob, que possui uma riqueza acumulado de 200 $recursos \times dias$, pode esperar que o *kernel* seja gerado na metade do tempo. Com a PoS, a probabilidade de gerar o *kernel* independe do poder computacional dos nós da rede [King e Nadal, 2012]. Caso exista uma ramificação da cadeia, aquela que será declarada como principal é a que possui a maior soma de riqueza acumulada consumida [King e Nadal, 2012]. Existe a possibilidade de as ramificações crescerem na PoS quando a implementação não fornece incentivo para que os nós adicionem blocos à cadeia correta, originando o problema conhecido como "nada a perder" (*nothing-at-stake*) [Kiayias et al., 2017]. Dessa forma, os nós adicionam blocos a múltiplos ramos para maximizar a probabilidade de receber uma recompensa, prejudicando a obtenção de uma visão global única da cadeia de blocos.

4.4.3. Prova de Capacidade – PoC

O protocolo de consenso de prova de capacidade foi desenvolvido como uma alternativa à prova de trabalho. Introduzido com a criptomoeda *Burstcoin*[11], o algoritmo permite utilizar o espaço de armazenamento de memória secundária, ao invés de poder computacional bruto, para determinar o nó minerador. Assim, a PoC é baseada no espaço disponível do disco rígido do nó. O mecanismo funciona configurando o disco rígido para reservar um espaço de armazenamento em um processo chamado "plotagem"(*plotting*). Com a plotagem no disco rígido, os cálculos são feitos de antemão e as possíveis soluções são armazenadas em disco. Algumas dessas soluções, ou parcelas das soluções, permitem alcançar a solução final mais rapidamente do que outras e o nó que alcançar primeiro a solução final para o bloco mais recente será recompensado pela mineração. Isso essencialmente permite que o nó gere uma receita passiva utilizando o espaço de armazenamento disponível. Em outras palavras, os nós mineradores pré-geram pedaços de dados conhecidos como grafos que são salvos no disco. O número de parcelas que o nó armazena é efetivamente sua velocidade de mineração. A cada bloco, o nó minerador correrá as parcelas salvas e obterá uma quantidade de tempo até poder extrair um bloco, se outro bloco ainda não tiver sido encontrado. Depois de ler as plotagens, o *hardware* permanece ocioso até o próximo bloco. Quanto mais grafos o nó obtiver em seu disco rígido, maiores serão suas chances de resolver o próximo bloco. O algoritmo de prova de capacidade mostra um potencial para a evolução das criptomoedas, pois requer menos energia que a PoW baseada em ASIC (*Application Specific Integrated Circuits*) [Gauld et al., 2017].

[11] Disponível em https://bitcointalk.org/index.php?topic=731923.0.

4.4.4. Prova de Autoridade – PoA

No contexto das redes privadas, em vez da prova de trabalho ou de participação, propõe-se o uso da Prova de Autoridade (*Proof of Authority* – PoA). Nas redes privadas, existe uma entidade responsável pela rede que pode pré-determinar o papel de alguns nós. Assim, na prova de autoridade, a ideia é designar um conjunto de nós com autoridade para participar do consenso. Esses nós são encarregados da tarefa de gerar novos blocos e validar as transações. A PoA endossa um bloco como parte da cadeia se ele for assinado por pelo menos um nó com autoridade. O modelo de incentivo na PoA destaca que é do interesse de um nó de autoridade manter sua reputação para permanecer como nó de autoridade. Deve existir, então, um mecanismo confiável que permita a avaliação do comportamento dos nós de autoridade na rede para definir a reputação de cada nó. Vale ressaltar que PoA mantém a natureza distribuída da rede pelo fato de que todos os nós de autoridade devem concordar sobre o estado global da cadeia. Plataformas que se baseiam em PoA como mecanismo de consenso aplicam um tipo de rotação entre os nós de autoridade para que cada um tenha um tempo para gerar blocos alternadamente, sem disputa e desperdício de recursos por mais de um nó. Após o nó minerador da rodada minerar o último bloco, todos os nós de autoridade devem concordar e adicioná-lo ao final da cadeia. Caso seja observada alguma falha do nó de autoridade, é preciso que a plataforma ofereça recursos para fiscalizar e retirar a autoridade desse nó e, como consequência, desconsiderar seus blocos minerados, retornando as transações para o conjunto de transações não mineradas [Cachin e Vukolic, 2017].

4.4.5. Protocolos de Consenso Baseados em Votação

Todos os mecanismos de consenso descritos anteriormente eventualmente alcançam a consistência da visão global da cadeia de blocos existente na rede. A consistência eventual é alcançada sem a necessidade de conhecer o número de participantes do consenso. Outro grupo de mecanismos de consenso é composto por protocolos baseados em votação. Para alcançar o consenso na rede, é necessário que uma determinada fração dos participantes do consenso entrem em comum acordo quanto à adição do bloco na cadeia de blocos da rede. Para tanto, um grupo de nós da rede participa do consenso, votando a favor ou contra qualquer proposta de modificação nos dados do sistema. Alguns exemplos desse grupo de protocolos são PBFT [Castro e Liskov, 1999], Ripple [Schwartz et al., 2014], Paxos [Lamport, 2001] e Raft [Lamport, 2001].

Os protocolos baseados no **Modelo Prático de Tolerância a Falhas Bizantinas** (PBFT) consideram que os nós da rede podem exercer comportamentos arbitrariamente maliciosos ou falhas que fogem do protocolo predefinido [Bessani et al., 2014, Castro e Liskov, 2002, Alvarenga et al., 2018]. Apesar da participação de nós maliciosos, os protocolos baseados no PBFT garantem o consenso entre os nós da rede até o número limite de nós maliciosos, chamados de nós bizantinos, atingir f, em que $f \leq \frac{n+1}{3}$ e n representa o número total de nós da rede. O mecanismo de consenso PBFT pode ser resumido em quatro fases. A primeira fase é determinar o nó líder da rodada de mineração, que geralmente segue um rodízio entre os nós da rede, considerando que todos os nós são iguais e participam do consenso. Na segunda fase, o nó líder gera um novo bloco e o encaminha para todos os nós da rede. Na terceira fase, o nó líder aguarda a resposta de no mínimo $f+1$ nós com o mesmo resultado para o novo bloco. A quarta fase é a execução das tran-

sações contidas no bloco, visto que o consenso foi alcançando e os nós compartilham da mesma visão da cadeia. Como resultado final, todos os nós legítimos chegam a um acordo sobre a ordem das transações e as aceitam ou rejeitam. Além disso, algumas aplicações de PBFT oferecem opções nas quais a maioria absoluta de nós legítimos pode decidir se um líder está com defeito, ou sendo desonesto. A maioria pode votar para removê-lo da rotação de líder nas próximas rotações para a geração de blocos. Em comparação à PoW, os protocolos baseados em PBFT são vantajosos em termos de processamento. Por outro lado, estes protocolos exigem uma grande complexidade de mensagens, $O(n^2)$, o que gera um problema de escalabilidade para a rede. Consequentemente, protocolos baseados em PBFT são adequados para redes com poucos nós.

Schwartz et al. propõem o protocolo **Ripple** como mecanismo de consenso distribuído para cadeias de blocos federadas[12] [Schwartz et al., 2014]. O Ripple é tolerante a falhas bizantinas e é robusto contra ataques de conluio. Essa robustez advém da criação de subconjuntos de zonas confiáveis, nas quais não se espera uma conspiração entre os nós para atacar o sistema. Dessa forma, os nós consultam apenas o subconjunto e nós confiáveis para alcançar o consenso. Um dos problemas do Ripple é a escalabilidade quanto ao número de nós designados para alcançar o consenso [Cachin e Vukolic, 2017].

Paxos [Lamport, 2001] e **Raft** [Ongaro e Ousterhout, 2014] são protocolos equivalentes, com o objetivo de gerenciar registros replicados de entradas de dados. Primeiramente é eleito um líder, que recebe todas as propostas de modificação de dados no sistema. O líder torna-se responsável por compartilhar todas as modificações com todos os outros nós, para que eles possam votar. Em seguida, o líder compartilha a decisão coletiva com todos os outros nós participantes do consenso. Ambos os protocolos são resilientes a f falhas, em que $f \leq \frac{n+1}{2}$ e n representa o número total de nós da rede. Esses protocolos foram desenvolvidos para serem usados em ambientes confiáveis, uma vez que não consideram comportamento malicioso de nós que participam do consenso. Dessa forma, só devem ser utilizados em cadeias de blocos privadas. Variações dos protocolos Paxos e Raft consideram a redução do número de fases para alcançar o consenso com menor número de mensagens e a assinatura das mensagens trocadas permite extrapolar o uso desses protocolos em ambientes não confiáveis [Mattos et al., 2018, Cachin e Vukolic, 2017].

4.5. Contratos Inteligentes no Mercado de Energia Elétrica

A popularização dos contratos inteligentes é evidente com o crescimento exponencial do número de transações realizadas por esses contratos nos últimos anos, acarretando dezenas de milhares de contratos armazenados na plataforma Ethereum, responsáveis pelas movimentações de milhões de dólares [Luu et al., 2016]. A Figura 4.6 mostra o crescimento acelerado do número de contratos na Ethereum entre 2015 e 2018. Como consequência dessa popularização, hoje já existem diversas outras plataformas disponíveis para os contratos inteligentes [Bartoletti e Pompianu, 2017].

A primeira definição de contrato inteligente foi proposta por Nick Szabo em 1994 e colocava que o contrato inteligente se tratava de um protocolo de transação computado-

[12]Cadeias de blocos federadas são cadeias privadas sob a liderança de um grupo ou consórcio de nós.

[14]Gráfico gerado com base nos dados disponíveis em https://hackernoon.com/ethereum-smart-contracts-most-of-them-are-rarely-used-f45749730d3e.

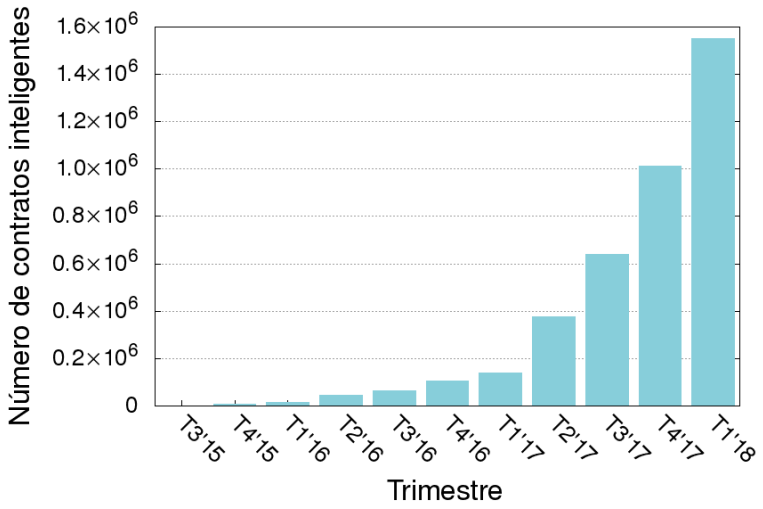


Figura 4.6. Crescimento exponencial do número de contratos inteligentes na plataforma Ethereum[14] entre os anos de 2015 e 2018.

rizado que executa os termos de um contrato. Na época, não existiam estruturas ou poder computacional para dar amplo suporte a aplicação da ideia, o que a fez cair em desuso por mais de vinte anos [Christidis e Devetsikiotis, 2016, Giancaspro, 2017].

Os contratos inteligentes oferecem transações eficientes, com baixo custo e seguras sem a necessidade de intermediários, tais como bancos e companhias de crédito, que encarecem muito o custo do negócio [Giancaspro, 2017]. Os contratos inteligentes automatizam operações, tornando a sua efetividade e as transferências de crédito mais céleres. Para tanto, o programa definido pelo contrato inteligente deve ser executado sobre uma cadeia de blocos e ser garantido por um protocolo de consenso, que pode ser o mesmo da cadeia de blocos ou definido em uma plataforma a parte. Com isso, o contrato tem acesso a movimentações financeiras por meio da criptomoeda da cadeia [Giancaspro, 2017]. O contrato deve respeitar um conjunto de regras definidas em uma linguagem de programação específica e a lógica do negócio pode ser expressa em código e validada a cada novo pedido de transação [Christidis e Devetsikiotis, 2016]. Com isso, os contratos inteligentes podem ser usados para uma vasta gama de aplicações.

O contrato inteligente possui um estado, está associado a um conjunto de dados e a uma quantidade de criptomoedas e possui um endereço próprio[15]. O endereço do contrato é usado pelas mensagens ou transações que ativam a execução do contrato. Assim, um contrato é executado ao ser invocado por uma transação ou pelo envio de uma mensagem para o seu endereço. Com isso, se a transação é aceita na cadeia de blocos, todos os mineradores executam o contrato baseados no estado atual da cadeia. Para tanto, ao se considerar o uso de cadeias públicas não permissionadas, o contrato deve ser sempre associado a uma quantidade de criptomoedas, que deve ser suficiente para financiar a execução do contrato, além de cobrir eventuais transações financeiras que sejam inerentes ao

[15]Variações nos conjunto de informações associadas ao contrato ocorrem dependendo da tecnologia utilizada. Por exemplo, diferentemente da Bitcoin, a Ethereum permite contratos com estados que podem ser modificados a cada nova transação [Luu et al., 2016].

contrato. A cada linha de execução do contrato, uma determinada quantia de criptomoedas é descontada do valor associado ao contrato. Esse mecanismo garante que o contrato só pode executar enquanto houver crédito suficiente, evitando ciclos mortos (*deadlocks*) ou execuções infinitas.

Por ser baseado em um protocolo de consenso, o contrato precisa ser determinístico. Logo, qualquer um dos mineradores que execute o contrato, com um certo conjunto de entradas, obterá o mesmo resultado como saída. Por estar armazenado na cadeia de blocos, todos os participantes da cadeia podem inspecionar o código do contrato. Além disso, como todas as transações sobre o contrato são registradas na cadeia, é possível fazer uma auditoria de todas as operações realizadas por meio daquele contrato [Christidis e Devetsikiotis, 2016]. Vale ressaltar que o contrato inteligente é armazenado dentro da cadeia e, então, após criado, ele não pode ser modificado ou cancelado [Giancaspro, 2017], o que pode trazer complicações legais. Como nenhum bloco pode ser removido ou alterado na cadeia, não é possível alterar um contrato já criado. Dessa forma, para alterar o contrato, em caso de observação de alguma falha ou vulnerabilidade, a cadeia precisaria ser reescrita, o que não é viável na maioria dos casos. Outro recurso que vem sendo usado é a utilização de contratos que guardam listas de contratos já cancelados e que são invocados para validação antes da execução de um código.

Os contratos inteligentes podem ser usados para diversos fins. Bartoletti e Pompianu propõem uma classificação das aplicações típicas dos contratos inteligentes com base em uma pesquisa nos contratos existentes na plataforma Ethereum [Bartoletti e Pompianu, 2017]. Os contratos são classificados em contratos de finanças, cartório, jogo, carteira e biblioteca.

Contratos de finanças são os que gerenciam, coletam ou distribuem criptomoedas. Alguns desses contratos verificam quem são os donos de um determinado ativo, verificam valores e lidam com a comercialização, enquanto outros realizam coleta de dinheiro para financiamento de projetos (*crowdfunding*). Nessa segunda categoria, merece destaque o DAO, que era o serviço de financiamento solidário mais proeminente da Ethereum, mas deixou de existir devido a um ataque em 2016 que desviou milhões de dólares. Existem ainda os esquemas Ponzi, nos quais os usuários investem dinheiro com a promessa de receber o valor com juros de volta no caso de novos investidores participarem do esquema (Ex. *King of the Ether Throne*). Existem ainda os contratos de seguradoras que podem ser comprovados digitalmente e ainda contratos para publicação de propaganda (Ex. *PixelMap*). Os contratos do mercado de energia são classificados nessa categoria.

Contratos de cartório servem para armazenar dados de forma permanente, registrando dados de origem e posse. Uma vez que os dados podem ser grandes, uma proposta é armazenar apenas o *hash* do documento na cadeia, o que prova a integridade e existência do documento, mas não pode garantir que o dado não será destruído, restando apenas a evidência que o documento deveria existir. Esse tipo de contrato também é usado para registro de direitos autorais (Ex. *Monegraph*), para registro de identidades por meio da associação de uma entidade com uma chave pública (Ex. *Physical Address*) e para aplicações mais livres, como o registro de mensagens por usuários para a comunidade (Ex. *Eternity Wall*). No setor elétrico, esses contratos podem ser utilizados para registrar as entidades do sistema e para registrar o consumo, a geração e o preço da energia.

Contratos de carteira lidam com chaves, enviam transações e gerenciam dinheiro e outros contratos, sendo gerenciados por um ou mais donos. Assim como os contratos financeiros, os contratos de carteira podem ser usados no setor elétrico, em especial no que diz respeito a transações relacionadas a grupos de clientes ou consórcios de empresas.

Os contratos de jogos servem para registros de apostas em jogos de azar (Ex. *LooneyLotery*) ou jogos de habilidades (Ex. *Etherization*). Já os contratos de biblioteca trazem funcionalidades de uso geral, como funções matemáticas úteis, e são exclusivamente criados para serem usados por outros contratos.

4.5.1. Plataformas de Contratos Inteligentes

Algumas das principais plataformas de contratos inteligentes, que estão atualmente disponíveis e apresentam um determinado grau de maturidade, são a Bitcoin, a Ethereum, a Counterparty, a Monax e a Lisk. As plataformas e o funcionamento de cada uma são detalhados a seguir.

O principal objetivo da **Bitcoin** é criar uma plataforma para a transferência de criptomoeda em um ambiente aberto e com conteúdo imutável. Dentro desse cenário, é possível criar formas limitadas de contratos inteligentes [Bartoletti e Pompianu, 2017]. A limitação se dá pelo fato de a Bitcoin utilizar uma linguagem de códigos executável que não é uma linguagem *Turing* completa, que permite apenas colocar algumas condições que devem ser verificadas antes de executar uma transação. A execução na Bitcoin é orientada a pilha e, portanto, não permite a execução de códigos iterativos. Entre as operações disponíveis, têm-se aritmética básica, lógica e operações criptográficas. Outra limitação importante é que apenas poucos nós da rede Bitcoin realizam o processamento de códigos executáveis que usem operações que tenham códigos que dependam de mais do que uma verificação de assinatura [Banasik et al., 2016].

A **Ethereum** é uma plataforma com mais suporte a contratos inteligentes que a Bitcoin, disponibilizando uma linguagem *Turing* completa de *bytecode* baseada em pilha para a codificação dos contratos. Para criação dos códigos, utilizam-se linguagens de alto nível, sendo a mais utilizada a *Solidity* [Ethereum, 2018]. Os usuários criam os seus códigos de contratos e os submetem, por meio de uma transação, para a cadeia de blocos. Para usar esse contrato, uma transação deve ser direcionada a esse contrato ou, ainda, pode-se chamar um contrato a partir de outro contrato da cadeia. O resultado da transação é validado pela rede. Como um outro diferencial com relação à Bitcoin, na Ethereum, tanto os usuários quanto os contratos podem armazenar, enviar e receber criptomoeda, tanto de ou para outros usuários quanto de ou para outros contratos. Vale destacar o sistema de gás (*gas*), o "combustível" para a execução do contrato. Como a execução de um contrato leva a um esforço computacional, que varia de contrato para contrato, a Ethereum paga aos mineradores uma quantia pelo processamento de um determinado contrato proporcional ao esforço aplicado. Cada instrução que pode ser usada em um contrato tem um preço em gás pré-determinado. Ao chamar um contrato, o usuário deve dizer o quanto ele quer pagar por cada unidade de gás gasto na execução do contrato (*gasPrice*) e um gasto máximo que ele aceita pagar pelo contrato (*gasLimit*). Assim, com a execução do contrato, o minerador irá receber a quantia relativa à quantidade de gás usada na execução multiplicada pelo *gasPrice*. Caso o valor exceda o *gasLimit*, a

execução é pausada e o minerador recebe *gasLimit* sem ter que enviar o resultado final do contrato ao remetente. Esses casos geram uma exceção e levam à reversão das operações realizadas até a finalização do gás [Luu et al., 2016].

O **Counterparty** [Counterparty, 2018] é um protocolo desenvolvido especificamente para criar contratos inteligentes sobre a cadeia de blocos Bitcoin. Para tanto, as informações que devem ser trocadas para a realização do contrato são inseridas em transações da Bitcoin com o prefixo CNTRPRTY, em campos que são ignorados pela Bitcoin, como, por exemplo, a saída do OP_RETURN ou como *hash* falso de chave pública. Assim, a rede Bitcoin ignora a informação, que é coletada e interpretada por nós do Counterparty. A tarifação padrão de contratos Counterparty é similar àquela proposta na Ethereum, sendo que diferentes tipos de operação serão associados a diferentes tarifas. Contudo, cabe observar que o Counterparty possui uma criptomoeda própria, o XCP, que é usada para pagar a execução do contrato por meio de *proof of burn*[16]. Além disso, os contratos utilizam a mesma linguagem que a Ethereum, sendo o diferencial a ausência de consenso na validação do resultado.

Diferentemente do Counterparty, a plataforma **Stellar** [Stellar, 2018] provê a sua própria cadeia de blocos com sua própria criptomoeda, chamada *lumen*, a qual é regida usando o acordo Bizantino federado[17]. Para a Stellar, o contrato inteligente não define uma linguagem específica e, além das contas individuais, é possível também ter contas com vários donos (*multisignature*).

A **Monax** [Monax, 2018] é uma plataforma para criar contratos da Ethereum, mas sem estar associado à cadeia ou à criptomoeda da Ethereum. A plataforma disponibiliza modelos padrão (*templates*) de contratos que podem ser configurados, facilitando a criação de novas regras de negócio. A plataforma permite a criação de cadeias privadas com regras de autorização configuradas por cada cadeia, com consenso baseado votação.

A plataforma **Lisk** [Lisk, 2018] provê a sua própria cadeia de blocos pública baseada em prova de participação, com sua própria criptomoeda. Permite a execução de contratos baseados em máquinas de Turing-completas nas linguagens *JavaScript* ou *Node.js*. Apesar de ter uma cadeia principal, cada contrato nessa plataforma é executado em uma cadeia a parte e a criptomoeda pode ser transferida entre a cadeia principal e as cadeias de contrato. Nas cadeias específicas de contrato, o dono do contrato pode especificar, entre outros, quais nós podem participar do mecanismo de consenso. A Tabela 4.1 apresenta um resumo das principais plataformas para contratos inteligentes.

4.5.2. Exemplos de Contratos Inteligentes na Ethereum

A Ethereum é o exemplo mais relevante de plataforma de contratos inteligentes e, portanto, será usada para exemplificar a estrutura de um contato inteligente. O contrato na Ethereum é uma sequência de funções que podem, entre outras ações, transferir *Ether* entre usuários e para outros contratos. As transações são usadas para criar novos contratos, chamar funções de contratos e transferir *Ether*. Um contrato pode receber *Ether*

[16]Mecanismo de consenso em que um nó envia uma determinada soma de ativos para um endereço a partir do qual não é mais possível consumir aquele ativo. Funcionamento semelhante ao *proof of stake*.

[17]O consenso é feito entre a vizinhança, a qual é composta pelos nós mais confiáveis para aquele nó. Semelhante ao protocolo Ripple.

Plataforma	Tipo de cadeia	Tamanho da cadeia	Intervalo entre blocos	Linguagem
Bitcoin	Pública	96 GB	10 minutos	Scripts Bitcoin + Assinatura
Counterparty				EVM Bytecode
Ethereum	Pública	17-60 GB	12 s	EVM Bytecode
Stellar	Pública	N/A	3 s	Cadeias de transações + Ass.
Monax	Privada	N/A	Configurável	EVM Bytecode + permissões
Lisk	Privada	N/A	Configurável	JavaScript

Tabela 4.1. Comparação entre as principais plataformas para criação de contratos inteligentes. Adaptado de [Bartoletti e Pompianu, 2017].

ou enviá-lo para outros usuários ou contratos usando a função ***pay***. Todo *Ether* recebido por um contrato é guardado na variável ***balance***, a qual não pode ser modificada pelo programador, mas apenas por funções pré-definidas na plataforma.

A Figura 4.7 exemplifica um contrato escrito na linguagem *Solidity* que tem por objetivo transferir uma quantidade de *Ether* armazenado em nome do contrato para um determinado receptor. Cada contrato é composto por campos e funções. As funções são chamadas passando a quantidade de *Ether* necessária para financiar a execução daquele conjunto de instruções pelos mineradores e, opcionalmente, alguma quantidade de *Ether* a ser transferida para outro contrato ou usuário durante a execução da função. Cabe ainda observar a existência das exceções que são tratadas de forma diferente do usual. Uma exceção, nessa linguagem, não pode ser capturada, resultando no fim da execução da função, com a reversão de todas as modificações realizadas, incluindo as transferências de *Ether*. Contudo, a taxa paga pela execução da função, o gás, é perdida [Atzei et al., 2017].

```
1  contract AWallet{
2      address owner;
3      mapping (address => uint) public outflow;
4
5      function AWallet(){ owner = msg.sender; }
6
7      function pay(uint amount, address recipient) returns (bool){
8          if (msg.sender != owner ||  msg.value != 0) throw;
9          if (amount > this.balance) return false;
10         outflow[recipient] += amount;
11         if (!recipient.send(amount)) throw;
12         return true;
13     }
14 }
```

Figura 4.7. Exemplo de contrato da Ethereum [Atzei et al., 2017].

Os contratos são construídos em estruturas semelhantes a uma classe. No exemplo da Figura 4.7, o contrato *AWallet* possui dois atributos, um que representa o dono do contrato e outro que registra quais valores já foram enviados por meio desse contrato. Na linha 5, a função *AWallet* funciona como um construtor, sendo chamada apenas na criação do contrato. Essa função, no exemplo, apenas registra que o dono do contrato é

quem o enviou para a cadeia de blocos (*msg.sender*). A função *pay* executa efetivamente a funcionalidade do contrato. Ela recebe como entradas uma determinada quantidade de Ether e um endereço para enviar essa quantia e retorna verdadeiro ou falso. A primeira ação realizada pela função *pay* é verificar se quem está chamando a função (*msg.sender*) é o dono do contrato. A função também garante que nenhum *Ether* será transferido para esse contrato. Qualquer quantia sendo transferida para uma função estaria na variável *msg.value*. Caso algum usuário diferente do dono chame a função *pay* ou algum *Ether* seja enviado quando a função é chamada, o contrato é terminado por uma exceção e o valor em *msg.value* é devolvido a quem chamou o contrato. A próxima verificação é se o contrato ainda possui recursos suficientes para fazer a transferência, o que é feito observando o valor da variável *this.balance*. Essa variável não pode ser atualizada pelo código do contrato diretamente, sendo controlada por funções próprias especificadas pela Ethereum, pois armazena a quantidade de *Ether* relacionada a cada usuário ou contrato. Caso não exista recurso suficiente, a função retorna falso. Caso exista, a transferência é registrada em *outflow* (linha 10) e, em sequência, a quantia é efetivamente enviada ao receptor. Caso a quantia não possa ser enviada, acontece uma exceção, levando a devolução do Ether e revertendo a modificação em *outflow*.

A estrutura básica de um contrato da Ethereum dá liberdade ao programador e permite sua ampla utilização em diferentes cenários, como o das redes elétricas. A plataforma desenvolvida pela *Energy Web Foundation*, embora use cadeia de blocos própria, é baseada na estrutura de contratos da Ethereum [Energy Web Foundation, 2018]. Outro exemplo é a empresa *Power Ledge*, que desenvolveu uma plataforma para controle de cobrança de energia renovável sobre a cadeia da Ethereum [Power Ledger Pty Ltd, 2018].

4.5.3. Vulnerabilidades dos Contratos Inteligentes

Contratos inteligentes trazem grandes vantagens em termos de eficiência e controle de operações, mas há que se considerar o impacto que pode ocorrer em caso de falhas na codificação ou no processamento do contrato. Um exemplo famoso de contrato atacado devido a uma vulnerabilidade foi o do DAO, que era parte da Ethereum e fazia financiamento colaborativo de projetos. O ataque realizado sobre uma falha de codificação desse contrato levou a perdas da ordem de 50 milhões de dólares, o que levou a extinção do serviço [Popper, 2016, Bartoletti e Pompianu, 2017].

Diferentemente de softwares tradicionais, quando uma falha é detectada ou é tornada pública, não é possível modificar o contrato, já que a cadeia de blocos impede que ele seja removido ou alterado, a menos que a cadeia seja reconstruída [Luu et al., 2016]. Embora seja possível reconstruir uma cadeia, essa é uma tarefa muito custosa e pouco provável de ocorrer.

Algumas características de contratos e da cadeia de blocos já vêm sendo discutidas pela sua relevância na construção e execução dos contratos inteligentes. Uma dessas características é a dependência da ordem das transações. Alguns contratos permitem que o dono modifique o valor pago pela execução de um determinado desafio proposto pelo contrato. Assim, se duas transações, uma para apresentar a solução e outra para mudar o preço da solução acontecem em tempos muito próximos, a ordem em que o minerador escolhe para construir o bloco pode alterar o resultado final e os ganhos ou perdas de

cada uma das partes. Esse é um problema especialmente relevante em contratos com alta taxa de atualização de preços [Luu et al., 2016]. Por exemplo, um mercado de energia renovável que tenha alta variação de preço poderia sofrer com esse problema. Um caso hipotético é que um "prossumidor" verifique o preço que está sendo pago pela energia renovável e decida vender a sua produção, ao invés de consumir a energia em sua própria residência. Contudo, uma transação reduzindo o preço pago pela energia é lançada um pouco depois que o "prossumidor" pede o registro da sua venda para o contrato, mas essa nova transação, por ter uma remuneração de retorno maior pela mineração que a transação gerada pelo "prossumidor", deve ser minerada antes da transação de venda de energia. Nesse cenário, é provável que o "prossumidor" tenha sua transação efetivada depois da transação reduzindo o preço e reduzindo o seu lucro. Um contrato que gera essa dependência da ordem das transações é explicitado na Figura 4.8.

```
1 contract MarketPlace{
2   uint public price;
3   uint public stock;
4   /.../
5   function updatePrice(uint _price){
6     if (msg.sender == owner)
7       price = _price;
8   }
9   function buy (uint quant) returns (uint){
10    if (msg.value < quant * price || quant > stock)
11      throw;
12    stock -= quant;
13    /.../
14  }}
```

Figura 4.8. Exemplo de contrato da Ethereum que permite que os valores recebidos pelas partes variem de acordo com a ordem que os mineradores decidem processar as transações [Luu et al., 2016].

O problema da ordem de execução pode ocorrer naturalmente ou de forma maliciosa, quando o proprietário do contrato tenta alterar a ordem de execução das transações participando da mineração, aumentando o preço que ele paga pela transação da sua atualização de preço ou pelo conluio com outros mineradores [Luu et al., 2016]. Outro problema similar pode ocorrer em contratos que dependem de marcações de tempo para iniciar uma determinada operação, como, por exemplo, enviar uma quantia de dinheiro. Ao processar um bloco, um minerador precisa adicionar uma estampa de tempo ao bloco, o que para mineradores com funcionamento correto significa estampar o tempo atual no sistema local. Contudo, variações de até 900 s são toleradas dentro do processamento da cadeia. Com isso, atacantes podem manipular a estampa de tempo do bloco para manipular o resultado de um contrato dependente de estampa de tempo [Luu et al., 2016]. Uma outra vulnerabilidade relaciona-se a contratos que lidam com exceções. Por exemplo, na Ethereum, um contrato pode chamar outro contrato. Em caso do contrato chamado gerar uma exceção, essa exceção pode ou não ser propagada para o contrato original, dependendo da forma como o contrato foi chamado, que pode ser feito pela instrução *send* ou pela chamada direta da função. Quando a chamada é feita diretamente pelo nome da função, a exceção se propaga e ambos os contratos são terminados e devolvem as quantias pendentes. Contudo, quando a chamada é feita por meio de *send*, se o contrato que fez a

chamada não verificar a resposta da função, ele pode executar assumindo que a chamada do outro contrato foi realizada com sucesso. Observou-se que aproximadamente 27% dos contratos da Ethereum usam a chamada *send* para outros contratos e não verificam a resposta recebida [Luu et al., 2016]. A Figura 4.9 mostra um exemplo de contrato real da Ethereum que tem essa vulnerabilidade e, por isso, levou ao fim do serviço *King of Ether Throne*. Nesse serviço, um usuário poderia se tornar rei, desde que pagasse uma compensação ao rei atual. Com isso, a cada troca de rei, o novo rei deve compensar o rei anterior, gerando lucro. Contudo, não existia nenhuma verificação se a função *claimThrone* estava sendo chamada por um usuário ou por um contrato. Assim, o rei poderia ser tanto um usuário quanto um contrato. Na Ethereum, transferir dinheiro para um contrato custa mais gás que transferir dinheiro para uma carteira. Como o código completo desse contrato não leva em consideração que um rei pode ser um contrato, quando o endereço do rei é um contrato, o gás para fazer a transferência da compensação é insuficiente. Com isso, o *send* falhava com uma exceção no contrato do rei atual, levando o rei atual a perder o trono sem receber a compensação [Bennett, 2018].

```
1 contract KingOfTheEtherThrone {
2   struct Monarch {
3     // address of the king.
4     address ethAddr;
5     string name;
6     // how much he pays to previous king
7     uint claimPrice;
8     uint coronationTimestamp;
9     }
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,
20        valuePaid, block.timestamp);
21  }}
```

Figura 4.9. Contrato da Ethereum do serviço *King of Ether Throne*, que por não conferir a resposta da operação send, foi alvo de um ataque que finalizou o serviço [Luu et al., 2016].

Outras vulnerabilidades específicas da Ethereum incluem o estouro deliberado da profundidade máxima da pilha de chamadas, chamadas para destinos que não existem, exploração de estados em reentrâncias, entre outras [Luu et al., 2016, Atzei et al., 2017]. Vale notar que algumas ferramentas de análise de contratos na cadeia Ethereum mostram que a quantidade de contratos com vulnerabilidades é alarmante [Luu et al., 2016]. Assim, a elaboração de contratos inteligentes seguros é um desafio de pesquisa, especialmente para lidar com mercados com alto volume monetário como o mercado de energia.

4.6. Aplicações de Cadeia de Blocos em Redes Elétricas Inteligentes

A cadeia de blocos é uma tecnologia com grande poder disruptivo e, por isso, sua aplicação na indústria aumenta dia após dia [Dütsch e Steinecke, 2017]. Empresas cada vez mais desejam se comunicar através de interfaces de programação de aplicações (*Application Program Interfaces* - APIs), a chamada integração B2B (*Business to Business*). É necessário, para tanto, padronizar as interfaces para facilitar a comunicação entre as empresas. A padronização da integração B2B foca em três pilares: o formato dos dados, o processo empresarial e o protocolo de comunicação [Merz, 2016]. A tecnologia de cadeia de blocos satisfaz tais pilares. A cadeia de blocos define um formato de dados único em todos os nós da rede par-a-par e, por isso, todos os seus participantes seguem o mesmo formato de dados, processo empresarial e protocolo de comunicação. Consequentemente, o emprego da cadeia de blocos reduz custos de integração, uma vez que o número de comunicações aumenta exponencialmente com o número de participantes [Merz, 2016]. Elimina-se ainda a centralização de dados, pois os participantes trocam dados através da rede par-a-par. Identificam-se, por fim, fatores chave para o sucesso de uma aplicação em cadeia de blocos [Dütsch e Steinecke, 2017]:

- **Participantes acordam nas regras da negociação.** É essencial que os participantes cooperem para acordar padrões e regras para descrever as transações;
- **Um arcabouço legal e regulatório pode ser acordado.** O arcabouço legal e regulatório permite que os registros digitais sejam traduzidos em ativos e obrigações no mundo real;
- **Os papéis e permissões são definidos.** Os papéis e permissões dos nós participantes devem ser claramente definidos, no caso de uma cadeia privada, e acordados;
- **A identidade digital é associada ao mundo real**. A identidade dos atores deve ser adequada, vinculada ao mundo real e inalterável, no caso de uma cadeia privada.

4.6.1. Microrredes (*microgrids*)

Os sistemas de energia elétrica cresceram e evoluíram mantendo sua premissa inicial de construir centrais geradoras junto às fontes de energia primária e utilizar linhas de transmissão de longa distância para levar a energia produzida até os consumidores. Com as redes elétricas inteligentes, há modificações de natureza regulatória, estrutural e tecnológica no sistema de energia tradicional [Falcão, 2009]. Uma delas é a adoção de microrredes (*microgrids*), que são redes locais de energia elétrica compostas por cargas e elementos de geração e armazenamento, formando um sistema distribuído para provimento de energia elétrica. Os elementos geradores das microrredes podem estar localizados próximos aos pontos de consumo e podem ser conectados à rede elétrica da concessionária ou podem operar isoladamente, de forma totalmente independente da concessionária de energia. Essa autonomia facilita a operação, a automação e o gerenciamento dos recursos. Os elementos que compõem as microrredes agem como uma entidade única (subsistema), que pode ser operada de forma controlada e coordenada [Marnay et al., 2015]. Assim, é possível conectar um grande número de fontes geradoras de pequeno e médio porte ao sistema elétrico principal através de subestações de forma mais eficiente, segura

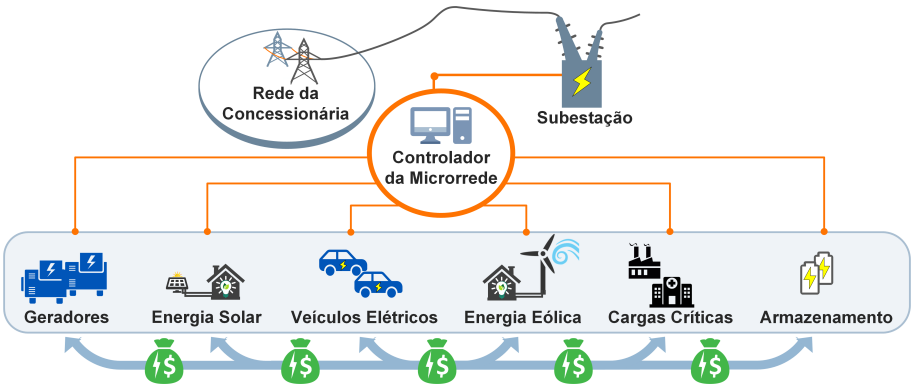


Figura 4.10. A microrrede é composta por cargas, fontes geradoras e armazenadores de energia distribuídos, que podem estar conectados a um ponto de acoplamento comum. Esses elementos são controlados e coordenados pelo controlador da microrrede, responsável por determinar se a microrrede deve operar de forma isolada ou conectada à rede da concessionária.

e gerenciável. A Figura 4.10 mostra uma microrrede e como esta se conecta à rede da concessionária. Ressalta-se que as microrredes agregam a geração de energia de fontes renováveis na rede de distribuição a média e a baixa tensão.

Os principais elementos geradores de energia nas microrredes são as fontes de energia renováveis e pequenas unidades como células a combustível, motores a diesel ou gás e pequenas turbinas a gás [Falcão, 2009]. Devido à tendência de uso de energias limpas e renováveis, diversas implementações de microrredes utilizam apenas fontes de energia renováveis como painéis fotovoltaicos e aerogeradores. Atualmente, as microrredes podem ainda se beneficiar da integração com estações de recarga e carros elétricos, capazes de exercer o papel de células distribuídas para armazenamento de energia. Em um momento crítico, os carros elétricos podem ser conectados à rede elétrica para suprir a demanda energética ou para reduzir flutuações na tensão ou frequência.

A coordenação da microrrede é feita através de equipamentos e técnicas de controle que definem a operação do sistema, determinando, por exemplo, a potência reativa e a variação da quantidade de energia adquirida da concessionária, sem perturbar o fornecimento de energia para os consumidores nem o funcionamento da rede da concessionária [Falcão, 2009]. Além disso, caso ocorram perturbações na rede da concessionária, como flutuações de frequência ou tensão, é possível desconectar totalmente a microrrede da rede principal sem cortar a alimentação das cargas [Lasseter e Paigi, 2004] alimentadas pela microrrede. Para esse fim, utiliza-se de forma intensa a tecnologia da informação e comunicação na rede elétrica, permitindo a comunicação entre os diversos componentes da rede e a otimização da sua operação.

O sistema de energia de cada cliente é controlado com o objetivo de alcançar uma solução ótima local para o consumo de energia, considerando as individualidades de cada sistema. No entanto, nem sempre a energia gerada pela microrrede nas instalações do cliente é totalmente consumida. O uso colaborativo do total de recursos disponíveis para um grupo de sistemas clientes pode resultar em uma solução global melhor. Assim, o excedente gerado pelos sistemas cliente pode ser inserido na rede da concessionária ou pode ser negociado diretamente com outros consumidores, originando o mercado de energia

das microrredes. Esse mercado tem o potencial de suprir a demanda local utilizando recursos de energia da vizinhança, reduzindo a necessidade de transporte de energia através de longas linhas de transmissão, que muitas vezes é caro e apresenta perdas substanciais no caminho [Mengelkamp et al., 2018a].

O "prossumidor" pode receber benefícios na forma de desconto nas contas de energia elétrica futuras ou pode ser pago utilizando algum tipo de moeda de troca. Para essa finalidade, é necessário existir um sistema eficiente, resiliente, igualitário e inviolável para transferência de informação, capaz de operar em um ambiente descentralizado [Cohn et al., 2017]. A estrutura descentralizada da tecnologia de cadeia de blocos vai ao encontro da mudança de paradigma em curso no mercado de energia. Essa mudança desloca o tradicional modelo de negócios centralizado para um modelo descentralizado.

A cadeia de blocos oferece um ambiente seguro e de baixo custo para transações financeiras ou operacionais serem armazenadas e validadas através de uma rede distribuída, sem a necessidade de um entidade central para controlar as transações [Power Ledger Pty Ltd, 2018]. A consistência dos dados é garantida pelo mecanismo de consenso utilizado na rede. As cadeias de bloco vêm ganhando cada vez mais espaço no setor de energia elétrica como ferramenta capaz de promover a negociação de energia. O uso das cadeias de bloco no setor de energia elétrica compõe, então, parte da solução para atualizar e melhorar os sistemas legados que operam de forma centralizada. Sistemas elétricos baseados em cadeias de bloco podem aumentar a confiabilidade e a qualidade do fornecimento de energia e promover a evolução para um sistema híbrido, no qual grandes usinas de energia coexistem com as microrredes distribuídas.

As negociações realizadas entre os "prossumidores" e consumidores são feitas de forma direta, formando uma rede par-a-par. Um dos requisitos para que negociações par-a-par locais sejam realizadas é a redução do tamanho dos lotes negociados. No comércio de energia e *commodities*, unidades padronizadas são definidas de acordo com o tamanho, a qualidade e a quantidade. A padronização dos critérios e do tamanho dos lotes é necessária para superar os custos de transação na configuração atual do mercado. Os atores não são capazes de vender nos mercados atacadistas de energia se a oferta não corresponder aos critérios padronizados. Eles são obrigados por intermediários, como corretores e bancos, a elaborar contratos. Assim, os negociantes de *commodities* são de fato grandes clientes ou especialistas [Dütsch e Steinecke, 2017]. As cadeias de blocos são capazes de reduzir os custos de transação através da padronização por contratos inteligentes e da execução automática das tarefas. A redução dos custos de transação permite lotes de energia de tamanhos menores, sendo possível eliminar os intermediários no comércio e comercializar a energia diretamente entre "prossumidores" [Dütsch e Steinecke, 2017].

Os contratos inteligentes contidos nas cadeias de blocos permitem a automatização dos pagamentos, habilitando o comércio entre "prossumidores" de forma segura e eficiente. Cada "prossumidor" deve estar equipado com um dispositivo capaz de armazenar todas as transações realizadas na rede composta pelos consumidores e produtores. Os contratos inteligentes devem ser elaborados de forma que a venda da energia excedente produzida seja automática. Ao manter um comércio local de energia, a quantidade de energia retirada da rede de transmissão na localidade é potencialmente reduzida [Cohn et al., 2017].

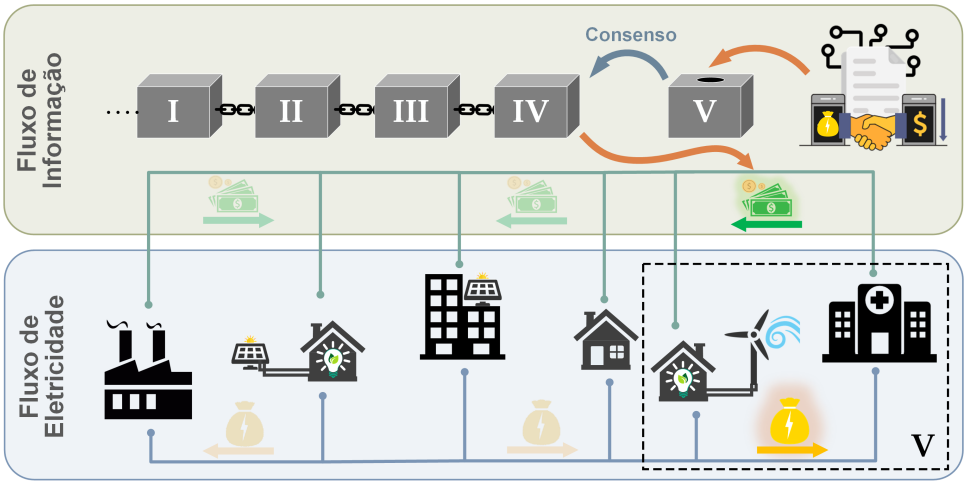


Figura 4.11. O modelo de mercado de energia baseado em cadeias de blocos é descentralizado. As negociações ocorrem de forma direta entre os pares, sem a necessidade de um agente intermediário. O fluxo de energia elétrica é transportado através da rede elétrica, enquanto o fluxo de informação passa por um sistema de tecnologia da informação e comunicação de alta capacidade.

Qualquer "prossumidor" e consumidor pode submeter um pedido de compra ou de venda ao sistema gerenciador de troca de energia da microrrede, suportado por um contrato inteligente. Esses pedidos são compostos por um lote e pelo preço do lote. O mercado pode funcionar com preços fixos, mas regulados pelo mercado, ou na forma de leilão, como na microrrede do Brooklyn, baseada em cadeia de blocos [Mengelkamp et al., 2018b]. Seja qual for o mecanismo de mercado, o lote de energia fornecido pelo "prossumidor" vencedor é inserido na microrrede, o consumidor vencedor paga ao "prossumidor" e pode, então, retirar a mesma quantidade de energia da microrrede. A Figura 4.11 exemplifica a negociação no mercado de energia de microrredes baseada em cadeia de blocos. No instante ilustrado, existe apenas um pedido de compra de energia, proveniente de uma carga crítica, por exemplo, um hospital. Esse pedido é feito através de uma chamada ao contrato inteligente que suporta o mercado de energia local. Existem diversos "prossumidores" com produção excedente que disponibilizam informações sobre o lote e o valor do lote de energia através de uma chamada para o contrato inteligente. O "prossumidor" vencedor da negociação e a carga crítica estão indicados na figura pela linha tracejada (conjunto V). A negociação realizada entre eles é incluída em um bloco candidato (bloco V), contendo as chamadas ao contrato, o valor do lote energia fornecido pelo "prossumidor" e o valor pago pela carga crítica. Após a verificação e validação do bloco através de um mecanismo de consenso, o bloco é inserido na cadeia de blocos e a compra é efetivada de fato, ocorrendo a transferência de ativos financeiros para a carteira do "prossumidor", e permitindo a retirada do lote de energia da microrrede. A principal diferença entre o modelo tradicional e o modelo baseado em cadeia de blocos é a eliminação do agente intermediário nas negociações.

Existem diversos projetos em desenvolvimento, ou que já estão sendo implementados e testados, que propõem o uso das cadeias de blocos no setor elétrico. Alguns desses projetos estão mais focados no comércio de energia entre os prossumidores das microrredes, como *PowerLedger*, *Key2Energy*, *Exergy* e *NRGCoin*. O projeto *Exergy* é uma plataforma criada pela *LO3 Energy*, utilizada na implantação da primeira microrrede

funcional do mundo, localizada no Brooklyn, em Nova York [Mengelkamp et al., 2018b]. Esses projetos são discutidos na Seção 4.7.

Existem alguns problemas que ainda precisam ser superados para que exista uma operação em larga escala das cadeias de blocos no mercado de energia elétrica. A rede da concessionária é fortemente regulamentada, assim como o comércio de energia elétrica. Negociações com cadeias de blocos também devem passar por um processo de regulamentação rígido. Um segundo problema que deve ser superado é a falta de interoperabilidade entre sistemas de diferentes empresas que estão desenvolvendo e utilizando cadeias de blocos próprias. O problema de interoperabilidade pode ser reduzido com a regulamentação do setor. Um terceiro problema é a eficiência energética das cadeias de blocos. Para que possam ser aplicadas no mercado de energia das microrredes, cadeias de blocos computacionalmente eficientes devem ser desenvolvidas [Mengelkamp et al., 2018a]. Por exemplo, em vez de utilizar mecanismos de consenso custosos em termos computacionais, é possível utilizar mecanismos de consenso baseados em identidade, nos quais cada agente possui uma única identidade que pode ser confirmada [Mengelkamp et al., 2018a]. Para que esses problemas sejam resolvidos, é necessária a existência de consórcios e organizações para cooperação entre companhias, operadores de sistema de distribuição e de transmissão e dos governos nacionais. Uma iniciativa nesse sentido é a *Energy Web Foundation* (EWF)[18], criada pela empresa GridSingularity e pelo Instituto de Rocky Mountain com o objetivo de acelerar a implementação da tecnologia de cadeia de blocos no setor elétrico. A EWF coopera com diversos parceiros do setor de energia elétrica, reguladores e órgãos de padronização e contribui com o desenvolvimento da cadeia de blocos EWF específica para aplicações no setor de energia elétrica, com diversos estudos de caso, provas de conceito e aplicações comerciais.

4.6.2. Medição Inteligente

Medidores inteligentes são dispositivos eletrônicos capazes de realizar medições do consumo de energia com mais detalhes do que os medidores comuns e com suporte a tecnologias de comunicação para reportar as medidas em tempo real [Efthymiou e Kalogridis, 2010]. A tecnologia de cadeia de blocos fortalece o papel de mercado dos consumidores e produtores individuais. Permite aos "prossumidores" comprar e vender energia diretamente, manualmente ou via automação de contratos inteligentes, com alto grau de autonomia. Os medidores inteligentes são os equipamentos responsáveis por medirem tanto o consumo como a produção de energia dos "prossumidores". No cenário de cadeia de blocos em microrredes, os medidores inteligentes passam a ser a interface entre as operações na cadeia de blocos e a realização das transações na rede elétrica. Logo, em microrredes, os medidores inteligentes são a interface entre a rede elétrica e a rede de comunicação, ou a cadeia de blocos.

No cenário de negociação de energia elétrica entre "prossumidores" com o suporte dos medidores inteligentes em uma microrrede, a compra e venda de energia elétrica se beneficiam da ausência das barreiras burocráticas, agilizando e barateando as transações por meio da tecnologia de cadeia de blocos [Bittwatt Pte.Ltd., 2018]. Com isso, os custos operacionais e tarifas de transferência são reduzidos pela automatização das transações ao

[18]Disponível em https://energyweb.org/.

aplicar os conceitos de contratos inteligentes. Com a utilização de uma carteira virtual, é possível participar de leilões de energia, em que as regras, como quantidade, tipo e preço, são levadas em conta para firmar contratos inteligentes entre ambas as partes.

Com os medidores inteligentes, os clientes podem ser alertados sobre o preço da energia quando esta ultrapassa o valor esperado, permitindo ajustes no consumo de energia nesses momentos para redução do valor da conta de energia elétrica [Bittwatt Pte. Ltd., 2018]. É possível também que os consumidores comprem energia quando esta torna-se mais barata e a armazenem para utilização posterior ou até para revender no futuro. Essa compra e venda de energia são feitas através de uma "Carteira de Energia" utilizando criptomoedas especiais [Bittwatt Pte.Ltd., 2018].

4.6.3. Gerenciamento e Negociação Automática de Energia

O crescente número de fontes de energia renováveis e intermitentes agregadas às redes elétricas levam a novas abordagens do mercado em relação à precificação e à distribuição da geração volátil e distribuída [Mengelkamp et al., 2018b]. Nesse novo cenário, um dos maiores desafios é o gerenciamento da rede elétrica inteligente [Guimarães et al., 2013]. Uma solução viável é agregar diversas fontes geradoras distribuídas de energia em uma usina de geração de energia virtual (*Virtual Power Plant* - VPP) [Pudjianto et al., 2007]. A VPP é comparável a uma usina de geração conectada diretamente à rede de transmissão, como ilustrado na Figura 4.12. Ao operarem sozinhas, muitas geradoras distribuídas de energia não têm capacidade, flexibilidade ou capacidade de controle e previsibilidade suficientes para interagir no mercado de energia elétrica de maneira rentável ou tecnicamente viável. No entanto, com a criação de uma VPP a partir de um grupo de geradoras distribuídas de energia, esses desafios são superados [Pudjianto et al., 2007]. A usina virtual é conectada diretamente à rede de transmissão e possui um perfil de características, como cronograma de geração, limites de geração, custo operacional e assim por diante, bem determinado. Com base nesse perfil, a usina virtual pode interagir diretamente com outros participantes do mercado para oferecer serviços e fazer contratos. Através da comunicação direta com o operador da rede de transmissão ou através de transações de mercado, uma unidade geradora conectada à rede de transmissão pode contribuir para o gerenciamento do sistema. A geração de energia elétrica e outros serviços associados podem ser vendidos por meio de interação no mercado atacadista ou por contato direto com concessionárias fornecedoras de energia e outras partes. O comércio ponto-a-ponto direto com agregação de fontes distribuídas em VPPs é uma solução viável e pode ser baseada na tecnologia de cadeia de blocos [Dütsch e Steinecke, 2017].

Para poder participar do mercado de compra e venda de energia, os operadores da usina virtual têm que produzir previsões para minimizar as flutuações. A complexidade envolvida na produção de previsões varia para cada tipo de instalação de geração. Derivar previsões para a produção de energia eólica e solar, por exemplo, é uma tarefa mais complexa do que para usinas controláveis, como usinas termoelétricas a gás. Caso um operador erre ao prever sua produção de energia, pode incorrer em prejuízos devido a multas e encargos pelo desequilíbrio energético provocado [Dütsch e Steinecke, 2017, Pudjianto et al., 2007]. O ator central de controle das VPPs pode ser substituído por uma implantação de uma solução baseada em cadeia de

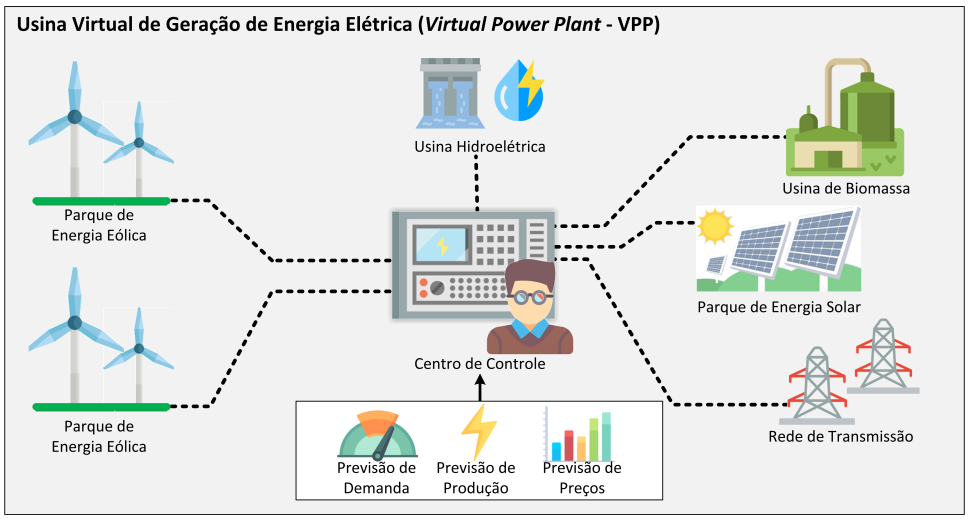


Figura 4.12. Usina virtual de geração de energia (VPP). Diversas fontes de energia renováveis são controladas por uma unidade de controle central e ligadas diretamente à rede de transmissão. A agregação de diversas fontes de energia permite gerenciar com mais previsibilidade as variações na demanda e na geração de energia. O controle centralizado pode ser substituído por contratos inteligentes em cadeia de blocos.

blocos que integra automaticamente informações locais e otimiza as redes de geração de energia. As redes locais são então agregadas à usina virtual, fornecendo capacidade de energia estável a baixo custo. Trabalhos recentes consideram o uso de cadeias de blocos para aumentar a segurança e a disponibilidade dos dados dos Sistemas de Supervisão e Aquisição de Dados (*Supervisory Control and Data Acquisition* - SCADA) [Dong et al., 2018, Liang et al., 2018]. A manutenção do sistema de supervisão distribuído e confiável é essencial para o funcionamento correto da usinas virtuais de geração.

Quanto à negociação automática de energia, vislumbra-se a tecnologia de cadeia de blocos como um canal de comunicação padronizado entre os atores do mercado de energia elétrica. O mercado de energia elétrica é composto por diversos atores, em especial geradoras, operadores de transmissão, distribuidoras, consumidores, negociante e câmara de comercialização [Merz, 2016]. Paralelamente, há ainda os órgãos de padronização e regulação do mercado. As geradoras alimentam a rede elétrica com a energia gerada, principalmente através de grandes usinas, mas atualmente também com pequenas usinas e usinas virtuais. As operadoras de transmissão são as responsáveis por manterem as redes físicas de distribuição da energia. As operadoras de transmissão são conectadas horizontalmente e trafegam a energia gerada pelas geradoras até os distribuidores e consumidores. As distribuidoras compram grandes quantidades de energia e oferecem produtos adequados aos consumidores, sejam residenciais ou industriais. Os consumidores compram os produtos correspondentes a seus perfis das distribuidoras. Vale ressaltar que os consumidores podem gerar energia e injetar diretamente na rede da distribuição, agindo assim como “prossumidores”. Os negociantes compram a energia das geradoras no mercado atacadista e revendem para outros negociantes e para distribuidoras. Muitas das vezes os negociantes compram e revendem os produtos diversas vezes, pois negociam com preços futuros sobre a energia a ser gerada. Há ainda as câmaras de negociação de energia, no Brasil, representada pela Câmara de Comercialização de Energia Elétrica

(CCEE)[19]. As câmaras de comercialização são associadas a pontos de troca de energia. A câmara de comercialização oferece o mercado no qual a energia é comercializada e, também, exerce meios para a liquidação financeira e física das transações de energia. Vale ressaltar que o mercado de energia oferece uma variedade de produtos que são negociados entre geradoras, negociantes e distribuidores. Os produtos ofertados variam de ofertas a longo prazo, para o mercado futuro, a operações *intraday*, para a liquidação no mesmo dia, com entregas escalonadas em intervalos de 15 minutos [Merz, 2016]. Nesse cenário, é necessário que as interfaces entre todos os atores sejam padronizadas e que o comportamento confiável e verificável de todos os atores seja garantido.

A abordagem de cadeia de blocos para o mercado de energia elétrica mantém todos os papéis [Dong et al., 2018]. Os negociantes continuam a realizar operações com outros negociantes e distribuidoras, corretores e a câmara de comercialização estão disponíveis como plataformas e os operadores de transmissão recebem dados de escalonamento da entrega da energia. Assim, os principais atores do mercado de energia passam a gerenciar um nó na cadeia de blocos que replica o mercado de energia elétrica. O modelo de cadeia de blocos permissionada é a que melhor se adéqua ao mercado de energia elétrica, pois assegura a comunicação entre nós, com diferentes papéis na rede, e entre participantes e nós da cadeia, no caso em que um nó representa um grupo de participantes com a mesma função na rede. O efeito mais significativo da adoção da cadeia de blocos é a padronização. Todos os participantes são forçados a ler e escrever dados exatamente no mesmo formato. Os processos de negócios são sincronizados com base nos estados e os dados estão disponíveis na cadeia de blocos. Logo, a cadeia de blocos é o veículo de transporte para distribuir dados consistentes e irrefutáveis entre todos do mercado.

Agências reguladoras passam a serem usuárias da cadeia de blocos. Ao acessarem um nó da cadeia, podem receber os dados das transações em tempo quase real, em comparação a atrasos atuais de aproximadamente 24 horas [Merz, 2016]. Vale ressaltar que ao operar sobre a cadeia de blocos, os participantes do mercado de energia não necessitam qualquer esforço extra de reportar as transações para as agências reguladoras, dado que a difusão intrínseca de informações na cadeia cumpre esse requisito legal. Dessa forma, a cadeia de blocos apresenta o potencial de diminuir o risco de fraudes, dado que um regulador pode monitorar as transações em tempo real. Portanto, se todos os que atualmente têm de relatar dados de transação às agências reguladoras registrarem esses dados na cadeia de blocos, basta que a agência reguladora também se conecte à cadeia de blocos com permissão de leitura. Cabe notar que o uso da cadeia de blocos também é vantajoso para os negociantes, pois os dados que estão armazenados na cadeia podem ser usados por várias partes como um banco de dados para as etapas de transformação posteriores. Os negociantes podem derivar o índice de certos produtos do banco de dados, que é uniforme para todos os participantes, e definir melhores estratégias de compra e venda de ativos.

4.6.4. Medição Automática em Sistemas de Transmissão

Os operadores de sistemas de transmissão precisam produzir previsões para todo o mercado de energia elétrica a cada dia. As previsões devem ser preparadas, o mais tardar, no dia anterior, com base nos chamados escalonamentos, submetidos a eles pelos

[19]Disponível em https://www.ccee.org.br/.

operadores de balanceamento do sistema energético [Hasse et al., 2016]. Cabe destacar que há as questões de quem são os responsáveis por submeterem esses escalonamentos aos operadores das redes de transmissão, a corretude dos escalonamentos e a integridade e validade irrefutáveis dos escalonamentos recebidos. Em paralelo, as medições de carga recebida e fornecida na rede de transmissão são relevantes ao operador da rede de transmissão para efeitos de compensação e liquidação. O operador da rede de transmissão coleta todos os dados para cada grupo de balanceamento e os agrega para determinar os custos de energia a serem alocados ao grupo de balanceamento.

A proposta do uso de cadeias de blocos, tanto no armazenamento das medições realizadas pelos operadores de transmissão quanto na padronização e na garantia de integridade e não repúdio nas negociações, simplifica os processos e a troca de informações entre operadores de redes de transmissão e outros atores do mercado de energia elétrica. Considerando que as operações de negociação de energia são realizadas diretamente sobre uma cadeia de blocos, mesmo que por motivos de relatórios, as transações também estarão simultaneamente disponíveis para o operador de transmissão. Assim, essa parte da tarefa da câmara de comércio, a liquidação física da entrega da energia, já é realizada pela cadeia de blocos. Quanto à liquidação financeira, o faturamento em forma de estrela, entre a câmara de comércio e os operadores, também pode ser implantado sobre a cadeia de blocos, uma vez que muitas soluções de cadeia de blocos, como a Ethereum, são equipadas com uma unidade de faturamento, os *tokens*, como o *Ether* da Ethereum. Há ainda o potencial para criação de novos *tokens* apenas para a comercialização de energia. Paralelamente à liquidação física da entrega de energia, através das entregas dos escalonamentos diárias e internos ao dia, e à liquidação financeira, é necessária a introdução de meios de verificação de que a energia negociada foi realmente entregue, tanto aos operadores de transmissão como pelos operadores de transmissão às distribuidoras e consumidores.

Atualmente, no Brasil, a medição do fluxo de energia em pontos da rede de transmissão é realizada pelo Sistema de Medição para Faturamento (SMF) composto por medidores e transformadores de potencial e de corrente elétrica. Esses equipamentos de medição são conectados ao Sistema de Coleta de Dados de Energia (SCDE) da CCEE. O sistema de coleta de dados realiza a coleta diária dos dados de todos os equipamentos de medição através da conexão com os equipamentos por meio de redes públicas, Internet, e, muitas vezes, sem o uso de proteções como Redes Privadas Virtuais (*Virtual Private Networks* - VPNs). A coleta diária impede o monitoramento preciso de variações do fluxo de energia no período de um dia, dificultando operações de planejamento e organização do sistema de transmissão. Em contrapartida, medições frequentes e com fina granularidade permitem um melhor controle do sistema. Os dados coletados são altamente sensíveis já que norteiam o quanto de energia está sendo fornecido por cada empresa geradora de energia e o quanto está sendo transmitido em uma dada linha de transmissão. No modelo atual, operadores de transmissão e geração devem realizar cópias de salvaguarda (*backup*) das medições e confiam nos valores coletados pela CCEE. Assim, nesse cenário, há a necessidade de se prover um mecanismo capaz de permitir que os dados coletados por diversas empresas sejam armazenados em uma estrutura de dados distribuída, sem controle centralizado e sem a necessidade de haver uma âncora de confiança em uma única entidade, capaz de garantir a disponibilidade, a integridade, a autenticidade, a auditabilidade e o não repúdio de todas as medições realizadas.

A cadeia de blocos satisfaz, então, as necessidades do sistema de coleta de dados. Com essa tecnologia, aumentam-se a transparência e a agilidade para comercializar energia porque as medições de fluxo de energia, assim que geradas, são disponibilizadas em um repositório distribuído e auditável [Dong et al., 2018]. Vale ressaltar que no modelo atual de coleta de dados, os pontos de medição são consultados uma vez por dia, em períodos predeterminados. Considerando a participação da CCEE na cadeia de blocos, a coleta de dados passa a ser realizada através da consulta na cadeia de blocos, recuperando as últimas transações e verificando o quanto foi registrado por cada medidor.

4.7. Discussão, Tendências e Desafios de Pesquisa

Embora se trate de uma área recente, já é possível encontrar diversas pesquisas e soluções da indústria que utilizam soluções de cadeias de blocos voltadas ao sistema elétrico. Via de regra, esses produtos visam a utilização das cadeias de blocos e dos contratos inteligentes como forma de registrar e dar suporte a operações seguras de compra e venda de energia. No entanto, o mercado alvo varia. A maioria dos sistemas envolve os consumidores finais de energia elétrica, embora haja também plataformas que dão suporte a negociações entre empresas das áreas de geração, transmissão e distribuição.

4.7.1. Soluções da Indústria

A **Bittwatt** [20], uma empresa sediada em Cingapura, desenvolveu uma plataforma de mesmo nome que funciona como um mercado digital para a troca e balanceamento energético. A plataforma baseia-se em cadeia de blocos para suporte a transações de compra e venda de energia elétrica entre vários tipos de participantes diferentes, desde os consumidores finais, até grandes empresas dos ramos de geração, transmissão e distribuição de energia. Do ponto de vista do usuário, o Bittwatt funciona como uma criptomoeda para transações no mercado de energia. Usuários, tanto fornecedores, quanto consumidores de energia, são cadastrados e a plataforma realiza o trabalho de casamento entre compradores e vendedores. As transações são consagradas através de contratos inteligentes. Segundo a empresa, o "fluxo de eletricidade é automaticamente codificado na cadeia de blocos" e os pagamentos são automaticamente disparados quando a energia é entregue.

A tecnologia da cadeia de blocos usada pelo Bittwatt é baseada na Ethereum. A solução inclui ainda a adição de medidores inteligentes na rede elétrica, tanto para consumidores, quando para produtores, de forma a permitir o monitoramento em tempo real do fluxo de energia. A empresa desenvolveu uma interface amigável que permite a cada usuário acompanhar a evolução do mercado, indicadores de preço, e gerenciar a carteira. A moeda usada na plataforma, denominada BWT, apresenta uma correspondência fixa com energia consumida/produzida: 1 BWT é equivalente a 1 kWh. A ideia é que usuários do sistema possam realizar o câmbio entre BWTs e outras moedas, o que determinaria o valor monetário da energia comercializada através da plataforma. Segundo os dados mais recentes publicados pela empresa [Bittwatt Pte.Ltd., 2018], sua solução já se encontra em fase de desenvolvimento em Londres, Bucareste e Cingapura. Segundo seu plano de negócio, a integração final de todos os componentes da plataforma será concluída em junho de 2022, data na qual a empresa antecipa operar em mais de 17 países.

[20]Disponível em http://bittwatt.com.

Diferentemente da Bittwatt, que ambiciona conectar todo tipo de participante no mercado de transação de energia, a australiana **Power Ledger**[21], criada pela Vector, tem como objetivo ser uma plataforma para negociação de energia em mercados locais de distribuição baseada em cadeia de blocos. A Power Ledger baseia-se na Ethereum e utiliza uma cadeia de blocos privada gerenciada pela própria empresa [Power Ledger Pty Ltd, 2018], embora haja planos para torná-la pública no segundo trimestre de 2019 e eventualmente gerenciada pelos próprios usuários. O mecanismo de consenso utilizado é atualmente baseado em Prova de Trabalho. No entanto, segundo a empresa, um dos objetivos futuros é transicionar a plataforma para o uso da Prova de Participação, uma das motivações para adotar a Ethereum como tecnologia. A outra é o suporte a contratos inteligentes. Um diferencial dessa plataforma é a utilização de duas criptomoedas para a realização de tipos de transações diferentes: a POWR e a Sparkz. A Sparkz é a moeda utilizada para transações efetivas de energia elétrica, enquanto a POWR é a criptomoeda usada para interconectar a plataforma com o mundo externo, como câmbio por outras criptomoedas ou por equivalentes monetários. A POWR já se encontra disponível para câmbio em diversas bolsas de criptomoedas. As transações são realizadas através de contratos inteligentes, compatíveis com o padrão ERC-20[22] de contratos inteligentes da Ethereum. Implantações da plataforma já foram testadas na Austrália e na Nova Zelândia. Segundo o plano de negócio publicado, a empresa está atualmente buscando novos parceiros comerciais e desenvolvendo novas aplicações sobre a plataforma.

Outro concorrente no setor é a **Exergy**[23], criado pela LO3 Energy. Similar às demais plataformas citadas anteriormente, a Exergy é a tecnologia base do projeto piloto *Brooklyn Microgrid*[24]. A moeda utilizada nas transações de energia nessa plataforma é chamada de XRG. Assim como as demais plataformas discutidas até aqui, a moeda é baseada no padrão ERC-20 [LO3 Energy Team, 2017].

O projeto *Brooklyn Microgrid* consiste em um mercado de energia para uma microrrede instalada no Brooklyn, em Nova York. Os participantes do piloto são residências e estabelecimentos da região que ainda estão conectados à rede elétrica de larga escala da cidade, mas fazem parte também de uma microrrede inteligente que os interconecta. Alguns desses usuários são "prossumidores", tendo, portanto, a capacidade de gerar energia renovável, através de painéis solares, por exemplo. Dessa forma, a Exergy é utilizada como uma plataforma para registrar transações de compra e venda de energia. A microrrede contém medidores inteligentes desenvolvidos pela LO3, denominados de *Transactive Meters*, capazes de interagir com a cadeia de blocos do Exergy. Essa cadeia de blocos, denominada *Transactive Grid Blockchain*, foi desenvolvida inicialmente sobre a Ethereum, mas a empresa migrou para uma solução de cadeia de blocos proprietária [Mengelkamp et al., 2018a]. A tecnologia em uso no *Brooklyn Microgrid* é baseada no *Tendermint* [Kwon, 2014]. Atualmente, a cadeia de blocos é privada.

Outra funcionalidade da plataforma é o casamento entre compradores e vendedores. Pedidos e ofertas de energia são cadastrados na cadeia de blocos na forma de

[21] Disponível em https://powerledger.io/.

[22] ERC-20 (*Ethereum Request for Comments 20*) é um padrão técnico que estabelece contratos inteligentes na cadeia de blocos da Ethereum para a implementação de *tokens*.

[23] Disponível em https://exergy.energy/.

[24] Disponível em https://www.brooklyn.energy/.

contratos inteligentes. A plataforma utiliza um sistema de leilão duplo, no qual o comprador que oferece o maior valor tem sua demanda atendida primeiro, seguido do segundo maior valor e assim por diante. Do ponto de vista do usuário, no entanto, o sistema de leilão é transparente, sendo gerenciado pela própria plataforma de acordo com preferências configuradas pelo usuário, por exemplo origem da energia desejada e valor máximo para compra. Outro aspecto interessante do projeto é a participação de instituições consideradas críticas para a sociedade, como hospitais. Essas instituições, no entanto, recebem uma parcela fixa da energia gerada pela microrrede, sem a participação no sistema de leilões.

Outra plataforma é a **PowerPeers**[25], originada na Holanda. Assim como a Power Ledger, a PowerPeers visa atuar no segmento de compra e venda de energia entre consumidores e produtores locais, explorando a popularização de tecnologias de geração de energia renovável em residências e outros tipos de consumidores finais.

A **Share & Charge**[26], por outro lado, é uma solução criada pela Motionwerk que tem como alvo o mercado de recarga de veículos elétricos. A ideia é criar uma plataforma baseada em cadeia de blocos que possa intermediar as transações entre usuários de veículos elétricos e entidades que disponibilizam postos de recarga, sejam empresas ou mesmo pessoas físicas que queiram disponibilizar uma infraestrutura de recarga própria para terceiros. A empresa conta atualmente com um projeto piloto no Reino Unido.

O número de iniciativas da indústria voltadas ao emprego das cadeias de blocos no mercado de consumo de energia elétrica é, de fato, grande. Além das plataformas já citadas nessa seção, Basden e Cottrell listam alguns outros projetos piloto pelo mundo [Basden e Cottrell, 2017]. Na Áustria, a *Wien Energie*, um conglomerado da área de energia, está atualmente testando uma solução de cadeia de blocos para gerenciar transações relacionadas à troca de energia com outras grandes empresas do setor. Também na Áustria, a *Grid Singularity* está desenvolvendo uma plataforma de troca de energia baseada em cadeia de blocos cujo diferencial é permitir, entre outras coisas, o monitoramento de equipamentos da rede elétrica. O objetivo é auxiliar no gerenciamento da rede elétrica. Na Alemanha, a empresa *Innogy* está testando uma solução piloto para a cobrança automática do consumo de carros elétricos autônomos em postos de recarga. As transações envolvidas são autenticadas por uma solução baseada em cadeia de blocos. No Reino Unido, a *startup Electron* desenvolve uma plataforma de cadeia de blocos para permitir que consumidores troquem rapidamente de empresa de fornecimento de energia.

4.7.2. Pesquisas Acadêmicas

Além das várias plataformas desenvolvidas por empresas, a aplicação de cadeias de blocos e contratos inteligentes às redes elétricas inteligentes tem gerado também bastante interesse acadêmico. Nos últimos anos, tem surgido uma massa considerável de artigos propondo plataformas ou estudando problemas pontuais nessa área.

O trabalho seminal nesse sentido é o NGRcoin [Mihaylov et al., 2014]. Trata-se de uma criptomoeda similar à Bitcoin, mas aplicada ao mercado de compra e venda de energia em mercados com geração distribuída. Embora a moeda proposta guarde muitas semelhanças com a Bitcoin, a principal diferença está na forma pela qual novas moedas

[25]Disponível em https://www.powerpeers.nl/.
[26]Disponível em https://shareandcharge.com/.

são criadas no sistema. Enquanto isso ocorre pelo processo de mineração na Bitcoin, no NRGcoin as novas moedas são criadas a partir da introdução de energia por um "prossumidor" na rede elétrica. Ao introduzir energia gerada localmente na rede elétrica, o "prossumidor" envia uma mensagem de aviso em *broadcast* para todos os demais participantes da rede NRGcoin. Por consequência, os demais participantes, em conjunto, criam uma determinada quantidade de NRGcoins e a creditam ao "prossumidor". Concomitantemente, a empresa responsável pela distribuição de energia elétrica na região transfere uma certa quantia de NRGcoins do seu próprio saldo para o "prossumidor". Por outro lado, se um "prossumidor" necessitar consumir energia da rede elétrica, ele deve pagar pela quantidade de energia consumida usando seus NRGcoins.

Ambos os preços de compra e venda de energia pelos "prossumidores" são determinados dinamicamente pela empresa distribuidora, em intervalos regulares de 15 minutos. Os autores propõem um modelo de precificação que tem como objetivo incentivar uma oferta de energia pelos "prossumidores" que case com a demanda atual. É importante destacar que os autores dissociam o processo de precificação da energia em NRGcoin do processo de câmbio entre NRGcoin e seu equivalente monetário. Tal câmbio, na proposta dos autores, seria realizado através de um mercado aberto, no qual a taxa de conversão fosse determinada pela relação entre oferta e demanda da criptomoeda.

Mais recentemente, Kang *et al.* apresentam um novo modelo de negócio para a compra e venda localizada de energia [Kang et al., 2017]. Nesse modelo, os autores assumem a popularização de veículos híbridos *plug-in* (ou PHEV, da sigla em inglês *Plug-in Hybrid Vehicule*). No modelo vislumbrado, usuários desse tipo de veículo teriam acesso a "tomadas bidirecionais", que permitiram fluxo de energia tanto da rede elétrica para o carro, como o carregamento das baterias, quanto do carro para a rede elétrica, fornecendo energia elétrica para a rede. Tais tomadas bidirecionais estariam disponíveis em estacionamentos públicos, postos de recarga ou mesmo nas casas dos usuários. Nesse cenário, veículos com baterias suficientemente carregadas eventualmente injetariam energia de volta na rede, auxiliando a suprir a demanda de energia na região onde se encontram. Segundo os autores, tal modelo ainda tem como vantagens a remoção do ponto único de falha no sistema de fornecimento de energia e reduziria a dependência do sistema de linhas de transmissão longas e redes de distribuição complexas. Os autores argumentam, no entanto, que esses benefícios só seriam alcançados caso uma parcela considerável dos usuários aderisse ao programa. Isto é, se esses usuários estivessem dispostos a permitir a descarga das baterias dos seus veículos para injeção de energia de volta à rede. Para isso, seria necessário algum tipo de incentivo que, na visão dos autores, poderia ser obtido através de um mercado de compra e venda de energia dos veículos híbridos. Mais especificamente, os autores buscam uma solução descentralizada e capaz de fornecer privacidade aos usuários participantes. Dados esses objetivos, os autores argumentam que a tecnologia de cadeia de blocos se apresenta como uma alternativa interessante. No entanto, soluções como o NRGcoin não seriam adequadas ao cenário pelas restrições energéticas dos veículos híbridos, dado o elevado custo energético dos mecanismos de consenso baseados em Prova de Trabalho. Como alternativa, os autores exploram o conceito de Consórcio de Cadeia de Blocos, no qual um grupo de nós dedicados atuam como Agregadores Locais. As transações de energia são negociadas diretamente entre os veículos, criptografadas e assinadas digitalmente pelas partes. As transações digitais resultantes

são enviadas aos agregadores locais que, por sua vez, se responsabilizam por auditá-las e registrá-las na cadeia de blocos através de um sistema de prova de trabalho. Os autores investigam também um sistema de precificação dinâmica, baseado em leilões duplos. No sistema proposto, compradores e vendedores submetem lances, os compradores submetem o valor que estão dispostos a pagar, enquanto vendedores submetem o valor pelo qual estão dispostos a vender. Com base nos lances, um preço de referência p é determinado. Vendedores cujos lances sejam menores ou iguais a p vendem, enquanto compradores cujos lances sejam maiores ou iguais a p compram.

Li *et al.* consideram outros cenários de compra e venda de energia em ambientes de geração distribuída, além do uso de veículos híbridos [Li et al., 2018]. Os autores também abordam aspectos práticos de implementação do sistema proposto, em particular, os relacionados ao estabelecimento de consenso entre os agregadores locais. O mais interessante, no entanto, é a proposta do uso de um sistema de crédito para compra de energia. Nesse sistema, cada agregador atua também como um banco com grandes reservas da criptomoeda energética. Um consumidor que precisa comprar energia, mas não possui moedas suficientes pode iniciar um pedido de empréstimo ao banco. O banco, por sua vez, avalia a capacidade de pagamento do empréstimo solicitado com base, por exemplo, o histórico de pagamentos de empréstimos anteriores do consumidor. Note que isso pressupõe que o consumidor ofereça informações de identificação para o banco, o que pode ter implicações relacionadas ao anonimato do sistema. De acordo com a capacidade de pagamento determinada, os autores apresentam um modelo derivado a partir de Teoria dos Jogos para computar os juros cobrados pelo empréstimo. Para concretizar o empréstimo, o banco cria uma carteira compartilhada com o consumidor com um saldo correspondente ao valor emprestado. As chaves criptográficas usadas para a manipulação da carteira são enviadas ao consumidor, juntamente com um certificado de autorização de uso da carteira, emitido pelo banco. O consumidor, então, pode usar a carteira para realizar compras de energia de outros participantes do sistema.

O propósito desse sistema de empréstimo é lidar com os atrasos nas verificações de operações na cadeia de blocos. Segundo experimentos conduzidos pelos autores, o atraso de verificação da cadeia de blocos proposta, baseada em consórcio, seria da ordem de dezenas de minutos. Esses valores são consideravelmente mais baixos que os atrasos de verificação encontrados em criptomoedas tradicionais, como a Bitcoin. Mesmo assim, os autores argumentam que esse atraso não é desprezível considerando-se a natureza das transações realizadas, que envolvem consumo e geração de energia elétrica. Assim, caso um consumidor precise comprar energia, mas esteja temporariamente sem moedas energéticas suficientes, um sistema tradicional baseado em câmbio a partir de outras moedas poderia incorrer em atrasos inaceitáveis.

Já Mengelkamp *et al.* consideram o cenário de um mercado local de compra e venda de energia com geração distribuída [Mengelkamp et al., 2018b]. A microrrede é ligada a uma rede tradicional de larga escala, mas "prossumidores" locais tentam suprir a demanda local com sua própria geração sempre que possível. Assim como nas demais propostas discutidas nessa seção, isso é feito através da manipulação de preços de compra e venda de energia produzida pelos "prossumidores", em um esquema de leilão duplo. Propõe-se a utilização de uma cadeia de blocos privada para registrar as operações de compra e venda. Os autores argumentam que a opção por uma cadeia de blocos privada

se deve à redução do custo do processo de consenso. O principal diferencial desse trabalho em relação aos anteriores está no uso da cadeia de blocos também para gerenciar as informações relativas ao leilão. Enquanto outros autores geralmente assumem uma infraestrutura e/ou protocolo a parte para lidar com os leilões, nesse trabalho os lances, ou seja, intenções de compra ou venda, são armazenados na própria cadeia de blocos, na forma de contratos inteligentes. Em outras palavras, o próprio lance gerado pelo consumidor, seja para compra ou para venda, constitui um compromisso de execução da transação no futuro dependente do valor de mercado a ser determinado. Uma vez que o leiloeiro determina e dissemina o valor de mercado da energia naquele momento, os contratos são executados de acordo com os lances e os saldos das carteiras são atualizados.

Em outro trabalho [Mengelkamp et al., 2018a], os mesmos autores argumentam que, em um mercado local de compra e venda de energia, os participantes do mercado são conhecidos e, por isso, o sistema de consenso baseado em prova de identidade pode ser usado. Essa seria uma alternativa mais eficiente em termos de custos de manutenção da cadeia de blocos, incluindo custos energéticos. No entanto, os próprios autores reconhecem que o sistema de prova de trabalho garante níveis mais altos de resiliência e segurança ao sistema. Dentre as outras contribuições desse trabalho, os autores propõem um modelo de sete requisitos para a viabilidade de uma microrrede baseada em cadeia de blocos. Um dos aspectos citados é a adoção da solução por um número suficiente de participantes, alguns dos quais precisam possuir a infraestrutura necessária para produção de energia. Os autores apontam ainda a escalabilidade da cadeia de blocos, em termos da taxa de verificação de transações, como um dos obstáculos técnicos ainda existentes.

Além dos aspectos técnicos relacionados a cadeias de blocos, contratos inteligentes e redes elétricas inteligentes, outros autores têm discutido os impactos regulatórios, econômicos, comerciais e políticos da combinação dessas tecnologias. Por exemplo, Green e Newman argumentam que a viabilidade dos mercados locais de troca de energia causará mudanças profundas no modelo de negócio das empresas do setor elétrico [Green e Newman, 2017]. Isso é algo que já ocorre na prática, ao menos em projetos piloto. Os autores, no entanto, alertam que a rede elétrica de larga escala ainda exercerá um papel importante no fornecimento de energia, dadas as questões de previsibilidade no consumo e das fontes de energia renováveis mais difundidas. Na prática, as grandes empresas do setor poderão acumular novos modelos de negócio, oferecendo, por exemplo, sua *expertise* na prestação de serviços de infraestrutura das microrredes. Já Bertsch *et al.* conduziram uma pesquisa de opinião na Alemanha sobre os níveis de aceitação da população em relação a diferentes tecnologias de geração de energia e aspectos relacionados [Bertsch et al., 2016]. Entre os dados levantados, aproximadamente 50% e 85% dos participantes disseram aceitar a instalação de painéis solares e turbinas eólicas se estas estiverem a ao menos 1 km de suas residências. Tais números indicam que ainda há certa resistência por parte considerável da população em relação à implantação da infraestrutura necessária a algumas das vertentes do conceito de geração distribuída.

4.8. Considerações Finais

A tecnologia de cadeia de blocos satisfaz os requisitos de segurança de aplicações em redes elétricas inteligentes, mesmo para aquelas aplicações em que não existe uma organização centralizadora para garantir a legitimidade das transações, como nas redes de

geração distribuída. A cadeia de blocos (i) assegura a confiabilidade da rede elétrica em um contexto em que não há confiança entre os pares, (ii) permite o controle e a manutenção dos dados de produção e de consumo de energia de forma distribuída, (iii) permite a auditoria do histórico de transações de compra e venda de energia elétrica de maneira irrefutável e, por fim, (iv) executa contratos de compra e venda de energia elétrica, independentemente da cooperação dos participantes.

Uma breve revisão das soluções da indústria e da literatura científica recente mostra que aplicações da cadeia de blocos e contratos inteligentes em diversos setores das redes elétricas inteligentes já se encontram próximas de estágios de produção. No entanto, também é clara a existência de obstáculos diversos a esse objetivo. Tais obstáculos são técnicos, políticos e até mesmo sociais.

Em termos técnicos, o desenvolvimento de contratos inteligentes seguros é um desafio. Há consenso de que os contratos inteligentes são necessários para a negociação segura e descentralizada entre diferentes atores do mercado de energia como, por exemplo, a compra e venda de energia entre os "prossumidores" em microrredes. Particularmente, o mercado de energia tem alto volume monetário e, nesse cenário, vulnerabilidades podem gerar enormes prejuízos financeiros. Por isso, a pesquisa nessa área é importante. Há, atualmente, vulnerabilidades tanto de codificação quanto no processamento dos contratos inteligentes que podem ser exploradas. A ordem escolhida para execução das transações e a construção do bloco pode afetar o valor a ser pago pela execução de um dado desafio causando prejuízos financeiros, por exemplo. Da mesma forma, contratos que dependem de estampas de tempo podem ter suas ordens alteradas para a ordem de execução dos blocos. Há ainda problemas identificados em contratos que geram exceções. Em particular, para a Ethereum, uma plataforma que possui cerca de 1,6 milhão de contratos inteligentes, é possível explorar o estouro deliberado da profundidade máxima da pilha de chamadas, chamadas para destinos que não existem, exploração de estados em reentrâncias, entre outras vulnerabilidades [Luu et al., 2016, Atzei et al., 2017]. O agravante é que quando uma vulnerabilidade é identificada ou é tornada pública, não é possível modificar o contrato, já que a cadeia de blocos impede que ele seja removido ou alterado, sem que a cadeia seja reconstruída. Portanto, esse é um desafio que requer um grande esforço dos pesquisadores e da indústria nos próximos anos.

Um outro desafio técnico é encontrar uma solução simultaneamente viável e totalmente descentralizada. Um dos grandes apelos da tecnologia de cadeia de blocos é fornecer um registro distribuído das transações, que não seja intermediado e dependente de uma determinada entidade. Os protótipos existentes e as propostas encontradas na literatura recente, no entanto, são soluções baseadas em cadeias privadas, gerenciadas por uma entidade central, ou um grupo pequeno responsável pela auditoria de todo o sistema. A reiterada opção por essa arquitetura é comumente justificada por problemas de escalabilidade na taxa de validação das operações e consequente aumento no atraso de verificação, em soluções de cadeias de blocos públicas. Além da questão do tempo, há também preocupação com a eficiência energética do gerenciamento da cadeia de blocos, algo particularmente importante nesse tipo de aplicação, dada a expectativa de que as redes elétricas inteligentes possam trazer menores perdas energéticas no sistema elétrico. Outra barreira técnica potencial associada à distribuição da gerência da cadeia de blocos é o custo associado à manutenção do aparato tecnológico necessário para que um usuário

participe do gerenciamento da cadeia. Esses custos podem ir de encontro à promessa de uma energia mais barata, repetidamente ecoada por empresas do setor e pesquisadores.

Do ponto de vista político, há questões regulatórias que podem afetar a viabilidade de implantação dessas soluções. No Brasil, por exemplo, atividades econômicas nos setores de geração, transmissão e distribuição de energia demandam concessões públicas atribuídas pelo governo [Ministério de Minas e Energia, 2012]. A atuação de tais plataformas no mercado brasileiro requer ajustes na legislação. Ademais, no caso de soluções que envolvam operações de compra e venda de energia diretamente entre "prossumidores", tais ajustes legislativos podem não contar com o apoio de grandes empresas já estabelecidas no setor. É importante considerar as preocupações governamentais com os processos de auditoria e recolhimento de impostos relativos às transações nesses mercados. Para que uma solução desse tipo seja legalmente introduzida no mercado brasileiro, há a preocupação de estabelecer mecanismos que permitam essas atividades.

Consideraram-se ainda os aspectos sociais dessas soluções. Embora uma solução de registro de transações através de cadeia de blocos e contratos inteligentes tenha como um de seus objetivos fornecer maior auditabilidade às operações do sistema elétrico, a adoção de cadeia de blocos só será bem sucedida se contar com a confiança dos consumidores. Isso é um desafio, especialmente considerando-se o fato de que a maior parcela de participantes do sistema será composta por usuários leigos.

Algumas das aplicações discutidas na literatura e implementadas em projetos piloto também pressupõem a popularização de veículos elétricos, o que ainda não é uma realidade em certos países, como no Brasil, onde a participação de veículos elétricos e híbridos em 2016 correspondia a cerca de 0,006% da frota total [FGV Energia, 2017]. Outro aspecto relevante é o impacto da dissociação das criptomoedas usadas para transações energéticas das suas contrapartes monetárias, algo proposto em várias das soluções encontradas na indústria e na literatura. Essa dissociação pode criar preocupação entre os consumidores acerca da volatilidade das cotações das criptomoedas e da possibilidade de manipulações nas cotações das taxas de câmbio.

Ao enumerar os desafios, conclui-se que há um vasto campo de pesquisa, ainda pouco explorado, para adoção da tecnologia de cadeia de blocos nas aplicações das redes elétricas inteligentes. Simultaneamente, há expectativa de investimento cada vez maior no desenvolvimento da tecnologia de cadeia de blocos, cujo mercado estimado é de centenas de milhões de dólares para os próximos anos. O cenário, portanto, é otimista: há demanda para pesquisa e recursos para financiá-la.

Referências

[Aitzhan e Svetinovic, 2018] Aitzhan, N. Z. e Svetinovic, D. (2018). Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams. *IEEE Transactions on Dependable and Secure Computing*, 15(5):840–852.

[Alvarenga et al., 2018] Alvarenga, I. D., Rebello, G. A. F. e Duarte, O. C. M. B. (2018). Securing configuration management and migration of virtual network functions using blockchain. Em *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, p. 1–9.

[Antonopoulos, 2014] Antonopoulos, A. M. (2014). *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 1 edição.

[Atzei et al., 2017] Atzei, N., Bartoletti, M. e Cimoli, T. (2017). A survey of attacks on Ethereum smart contracts SoK. Em *International Conference on Principles of Security and Trust*, p. 164–186.

[Banasik et al., 2016] Banasik, W., Dziembowski, S. e Malinowski, D. (2016). Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. Em Askoxylakis, I., Ioannidis, S., Katsikas, S. e Meadows, C., editors, *Computer Security – ESORICS 2016*, volume 9879 of *Lecture Notes in Computer Science*, p. 261–280. Springer International Publishing.

[Bartoletti e Pompianu, 2017] Bartoletti, M. e Pompianu, L. (2017). An empirical analysis of smart contracts: Platforms, applications, and design patterns. Em *Financial Cryptography and Data Security (FC)*, p. 494–509.

[Basden e Cottrell, 2017] Basden, J. e Cottrell, M. (2017). How utilities are using blockchain to modernize the grid. *Harvard Business Review*.

[Bennett, 2018] Bennett, E. (2018). Ethereum attacks. Disponível em `https://gist.github.com/ethanbennett/7396bf3f61dd985d3426f2ee184d8822`. Acessado em 24/08/2018.

[Bertsch et al., 2016] Bertsch, V., Hall, M., Weinhardt, C. e Fichtner, W. (2016). Public acceptance and preferences related to renewable energy and grid expansion policy: Empirical insights for Germany. *Energy*, 114:465 – 477.

[Bessani et al., 2014] Bessani, A., Sousa, J. e Alchieri, E. E. P. (2014). State machine replication for the masses with BFT-SMaRt. Em *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, p. 355–362.

[Bittwatt Pte. Ltd., 2018] Bittwatt Pte. Ltd. (2018). Bittwatt business plan. Relatório técnico, Bittwatt Pte. Ltd. Disponível em: `https://ico.bittwatt.com/static/files/Bittwatt-Business-Plan.pdf`.

[Bittwatt Pte.Ltd., 2018] Bittwatt Pte.Ltd. (2018). Bittwatt whitepaper. Relatório técnico, Bittwatt Pte. Ltd. Disponível em: `https://ico.bittwatt.com/static/files/Bittwatt-Whitepaper-EN.pdf`.

[Buterin et al., 2013] Buterin, V. et al. (2013). Ethereum white paper, 2014. Relatório técnico. Disponível em: `https://github.com/ethereum/wiki/wiki/White-Paper`.

[Cachin, 2016] Cachin, C. (2016). Architecture of the hyperledger blockchain fabric. Em *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*.

[Cachin e Vukolic, 2017] Cachin, C. e Vukolic, M. (2017). Blockchain consensus protocols in the wild. Em *International Symposium on Distributed Computing (DISC)*, p. 1–16.

[Castro e Liskov, 1999] Castro, M. e Liskov, B. (1999). Practical Byzantine fault tolerance. Em *Symposium on Operating Systems Design and Implementation (OSDI)*, p. 173–186.

[Castro e Liskov, 2002] Castro, M. e Liskov, B. (2002). Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461.

[Chen et al., 2012] Chen, P., Cheng, S. e Chen, K. (2012). Smart attacks in smart grid communication networks. *IEEE Communications Magazine*, 50(8):24–29.

[Chicarino et al., 2017] Chicarino, V. R. L., Jesus, E. F., Albuquerque, C. V. N. e Rocha, A. A. A. (2017). Uso de blockchain para privacidade e segurança em Internet das coisas. Em *Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)*, p. 100–150. Sociedade Brasileira de Computação (SBC).

[Christidis e Devetsikiotis, 2016] Christidis, K. e Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303.

[Cohn et al., 2017] Cohn, A., West, T. e Parker, C. (2017). Smart after all: Blockchain, smart contracts, parametric insurance, and smart energy grids. *Georgetown Law Technology Review*, 1(2):273–304.

[Counterparty, 2018] Counterparty (2018). Protocol specification. Disponível em `https://counterparty.io/docs/protocol\_specification/`. Acessado em 20/08/2018.

[de Oliveira et al., 2018] de Oliveira, M. T., Carrara, G. R., Fernandes, N. C., Carrano, R. C., Albuquerque, C. V. N., Medeiros, D. S. V. e Mattos, D. M. F. (2018). Uma avaliação de desempenho de cadeias de blocos privadas permissionadas através de cargas de trabalho realísticas. Em *XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'2018)*, Natal/RN, Brazil.

[Dinh et al., 2017] Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C. e Tan, K.-L. (2017). Blockbench: A framework for analyzing private blockchains. Em *Proceedings of the 2017 ACM International Conference on Management of Data*, p. 1085–1100. ACM.

[Dong et al., 2018] Dong, Z., Luo, F. e Liang, G. (2018). Blockchain: a secure, decentralized, trusted cyber infrastructure solution for future energy systems. *Journal of Modern Power Systems and Clean Energy*.

[Douceur, 2002] Douceur, J. R. (2002). The sybil attack. Em *International workshop on peer-to-peer systems*, p. 251–260. Springer.

[Dütsch e Steinecke, 2017] Dütsch, G. e Steinecke, N. (2017). Use cases for blockchain technology in energy and commodity trading. Snapshot of current developments of blockchain in the energy and commidity sector.

[Efthymiou e Kalogridis, 2010] Efthymiou, C. e Kalogridis, G. (2010). Smart Grid Privacy via Anonymization of Smart Metering Data. *2010 First IEEE International Conference on Smart Grid Communications*, p. 238–243.

[Energy Web Foundation, 2018] Energy Web Foundation (2018). Buiding the grid's digital dna. https://energyweb.org/. Acessado em 24.08.2018.

[Ethereum, 2018] Ethereum (2018). Solidity. https://solidity.readthedocs.io/en/develop/index.html. Acessado em 20.08.2018.

[Falcão, 2009] Falcão, D. M. (2009). Smart grids e microredes: o futuro já é presente. Em *Anais do VIII Simpósio de Automação e Sistemas Elétricos*, SIMPASE '09.

[FGV Energia, 2017] FGV Energia (2017). Caderno de carros elétricos. Disponível em: `https://fgvenergia.fgv.br/sites/fgvenergia.fgv.br/files/caderno_carros_eletricos-fgv-book.pdf`.

[Gauld et al., 2017] Gauld, S., von Ancoina, F. e Stadler, R. (2017). The burst dymaxion: An arbitrary scalable, energy efficient and anonymous transaction network based on colored tangles. Relatório técnico.

[Giancaspro, 2017] Giancaspro, M. (2017). Is a 'smart contract' really a smart idea? insights from a legal perspective. *Computer Law & Security Review*, 33(6):825 – 835.

[Green e Newman, 2017] Green, J. e Newman, P. (2017). Citizen utilities: The emerging power paradigm. *Energy Policy*, 105:283 – 293.

[Greenspan, 2015] Greenspan, G. (2015). Multichain private blockchain—white paper. *URl: http://www. multichain. com/download/MultiChain-White-Paper. pdf*.

[Greer et al., 2014] Greer, C., Wollman, D. A., Prochaska, D. E., Boynton, P. A., Mazer, J. A., Nguyen, C. T., FitzPatrick, G. J., Nelson, T. L., Koepke, G. H., Hefner Jr, A. R. et al. (2014). NIST framework and roadmap for smart grid interoperability standards, release 3.0. Relatório técnico.

[Greve et al., 2018] Greve, F., Sampaio, L., Abijaude, J., Coutinho, A., Ítalo Valcy e Queiroz, S. (2018). Blockchain e a revolução do consenso sob demanda. Em *Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, p. 1–52. Sociedade Brasileira de Computação (SBC).

[Guimarães et al., 2013] Guimarães, P. H. V., Murillo, A., Andreoni, M., Mattos, D. M., Ferraz, L. H. G., Pinto, F. A. V., Costa, L. H. M. e Duarte, O. C. M. (2013). Comunicação em redes elétricas inteligentes: Eficiência, confiabilidade, segurança e escalabilidade. *Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (Minicursos SBRC)*.

[Gunter et al., 2008] Gunter, C. A., Nelli, R., Gross, G. e LeMay, M. (2008). An integrated architecture for demand response communications and control. Em *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)(HICSS)*, volume 00, p. 174.

[Gupta e Sadoghi, 2018] Gupta, S. e Sadoghi, M. (2018). Blockchain transaction processing.

[Hadley et al., 2010] Hadley, M., Lu, N. e Deborah, A. (2010). Smart-grid Security Issues. *IEEE Security and Privacy*, 8(1):81–85.

[Hasse et al., 2016] Hasse, F., von Perfall, A., Hillebrand, T., Smole, E., Lay, L. e Charlet, M. (2016). Blockchain–an opportunity for energy producers and consumers. *PwC Global Power & Utilities*, p. 1–45.

[Igure et al., 2006] Igure, V. M., Laughter, S. A. e Williams, R. D. (2006). Security issues in SCADA networks. *Computers and Security*, 25(7):498–506.

[Jesus et al., 2018] Jesus, E. F., Chicarino, V. R. L., de Albuquerque, C. V. N. e Rocha, A. A. A. (2018). A survey of how to use blockchain to secure Internet of Things and the stalker attack. *Security and Communication Networks*, 2018:1–28.

[Kang et al., 2017] Kang, J., Yu, R., Huang, X., Maharjan, S., Zhang, Y. e Hossain, E. (2017). Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains. *IEEE Transactions on Industrial Informatics*, 13(6):3154–3164.

[Kiayias et al., 2017] Kiayias, A., Russell, A., David, B. e Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. Em Katz, J. e Shacham, H., editors, *Advances in Cryptology – CRYPTO 2017*, p. 357–388.

[King e Nadal, 2012] King, S. e Nadal, S. (2012). PPCoin: peer-to-peer crypto-currency with proof-of-stake. Relatório técnico.

[Kwon, 2014] Kwon, J. (2014). Tendermint: Consensus without mining. Relatório técnico. Draft versão 6. Disponível em: `https://cdn.relayto.com/media/files/LPgoWO18TCeMIggJVakt_tendermint.pdf`.

[Lamport, 2001] Lamport, L. (2001). Paxos made simple. *ACM SIGACT News*, 32(4):18–25.

[Langner, 2011] Langner, R. (2011). Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security Privacy*, 9(3):49–51.

[Lasseter e Paigi, 2004] Lasseter, R. H. e Paigi, P. (2004). Microgrid: a conceptual solution. Em *Anais do 35th Annual Power Electronics Specialists Conference (IEEE Cat. No.04CH37551)*, volume 6 of *PESC '04*, p. 4285–4290.

[Li et al., 2012] Li, X., Liang, X., Lu, R., Shen, X., Lin, X. e Zhu, H. (2012). Securing smart grid: cyber attacks, countermeasures, and challenges. *IEEE Communications Magazine*, 50(8):38–45.

[Li et al., 2018] Li, Z., Kang, J., Yu, R., Ye, D., Deng, Q. e Zhang, Y. (2018). Consortium blockchain for secure energy trading in industrial internet of things. *IEEE Transactions on Industrial Informatics*, 14(8):3690–3700.

[Liang et al., 2018] Liang, G., Weller, S. R., Luo, F., Zhao, J. e Dong, Z. Y. (2018). Distributed blockchain-based data protection framework for modern power systems against cyber attacks. *IEEE Transactions on Smart Grid*.

[Lisk, 2018] Lisk (2018). Access the power of blockchain. https://lisk.io/. Acessado em 20.08.2018.

[LO3 Energy Team, 2017] LO3 Energy Team (2017). Exergy: Electrical power whitepaper. Relatório técnico, LO3 Energy. Disponível em: `https://exergy.energy/wp-content/uploads/2017/12/Exergy-Whitepaper-v8.pdf`.

[Lopes et al., 2016] Lopes, Y., Bornia, T., Farias, V., Fernandes, N. C. e Muchaluat-Saade, D. C. (2016). Desafios de segurança e confiabilidade na comunicação para smart grids. *Miniursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (Minicursos SBSeg)*.

[Luu et al., 2016] Luu, L., Chu, D.-H., Olickel, H., Saxena, P. e Hobor, A. (2016). Making smart contracts smarter. Em *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS'16, p. 254–269, New York, NY, USA. ACM.

[Marnay et al., 2015] Marnay, C., Abbey, C., Joos, G., Ash, K., Bando, S., Braun, M., Chatzivasileiadis, S., Driesen, J., Hatziargyriou, N., Iravani, R., Jimenez, G., Katiraei, F., Lombardi, P., Lynch, K., Mancarella, P., Moneta, D., Moreira, C., Oudalov, A., Khattabi, M., Morris, G., Nakanishi, Y., Reilly, J., Ross, M., Shinji, T. e von Appen, J. (2015). Microgrids 1 engineering, economics, & experience. Relatório Técnico 635.

[Mattos et al., 2018] Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2018). A lightweight protocol for consistent policy update on software-defined networking with multiple controllers. *Journal of Network and Computer Applications*. A ser publicado.

[McDaniel e McLaughlin, 2009] McDaniel, P. e McLaughlin, S. (2009). Security and privacy challenges in the smart grid. *IEEE Security and Privacy*, 7(3):75–77.

[Mengelkamp et al., 2018a] Mengelkamp, E., Gärttner, J., Rock, K., Kessler, S., Orsini, L. e Weinhardt, C. (2018a). Designing microgrid energy markets: A case study: The brooklyn microgrid. *Applied Energy*, 210:870 – 880.

[Mengelkamp et al., 2018b] Mengelkamp, E., Notheisen, B., Beer, C., Dauer, D. e Weinhardt, C. (2018b). A blockchain-based smart grid: towards sustainable local energy markets. *Computer Science - Research and Development*, 33(1):207–214.

[Merz, 2016] Merz, M. (2016). Potential of the blockchain technology in energy trading. Em Burgwinkel, D., editor, *Blockchain Technology: An Introduction for Business and IT Managers*, chapter 2, p. 51–97. DE GRUYTER, Alemanha.

[Mihaylov et al., 2014] Mihaylov, M., Jurado, S., Avellana, N., Moffaert, K. V., de Abril, I. M. e Nowé, A. (2014). Nrgcoin: Virtual currency for trading of renewable energy in smart grids. Em *11th International Conference on the European Energy Market (EEM14)*, p. 1–6.

[Ministério de Minas e Energia, 2012] Ministério de Minas e Energia (2012). Concessões de geração, transmissão e distribuição de energia elétrica: Perguntas e respostas. Disponível em: http://www.mme.gov.br/documents/10584/1256596/Perguntas_e_respostas_-_Concessxes.pdf/57c8080d-eb1b-4052-9c3e-d3c4c010e974.

[Mo et al., 2012] Mo, Y., Kim, T. H. J., Brancik, K., Dickinson, D., Lee, H., Perrig, A. e Sinopoli, B. (2012). Cyber-physical security of a smart grid infrastructure. *Proceedings of the IEEE*, 100(1):195–209.

[Monax, 2018] Monax (2018). Monax - active agreements for growing businesses. https://monax.io/. Acessado em 20.08.2018.

[Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Relatório técnico.

[Neuman e Tan, 2011] Neuman, C. e Tan, K. (2011). Mediating cyber and physical threat propagation in secure smart grid architectures. *2011 IEEE International Conference on Smart Grid Communications, SmartGridComm 2011*, p. 238–243.

[Noce et al., 2017] Noce, J., Lopes, Y., Fernandes, N. C., Albuquerque, C. V. N. e Muchaluat-Saade, D. C. (2017). Identifying vulnerabilities in smart gric communication networks of electrical substations using geese 2.0. Em *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, p. 111–116.

[Ongaro e Ousterhout, 2014] Ongaro, D. e Ousterhout, J. (2014). In search of an understandable consensus algorithm. Em *Proceedings of USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, p. 305–320.

[Ouaddah et al., 2016] Ouaddah, A., Abou Elkalam, A. e Ait Ouahman, A. (2016). Fairaccess: a new blockchain-based access control framework for the internet of things. *Security and Communication Networks*, 9(18):5943–5964.

[Pilkington, 2016] Pilkington, M. (2016). 11 blockchain technology: principles and applications. *Research handbook on digital transformations*, p. 225.

[Popper, 2016] Popper, N. (2016). A hacking of more than $50 million dashes hopes in the world of virtual currency. https://www.nytimes.com/2016/06/18/business/dealbook/hacker-may-have-removed-more-than-50-million-from-experimental-cybercurrency-project.html. Acessado em 24.08.2018.

[Power Ledger Pty Ltd, 2018] Power Ledger Pty Ltd (2018). Powerledger whitepaper. Relatório técnico, Power Ledger Pty Ltd. Disponível em: https://powerledger.io/media/Power-Ledger-Whitepaper-v8.pdf.

[Pudjianto et al., 2007] Pudjianto, D., Ramsay, C. e Strbac, G. (2007). Virtual power plant and system integration of distributed energy resources. *IET Renewable Power Generation*, 1:10–16.

[Rahman et al., 2013] Rahman, M. A., Al-Shaer, E. e Bera, P. (2013). A noninvasive threat analyzer for advanced metering infrastructure in smart grid. *IEEE Transactions on Smart Grid*, 4(1):273–287.

[Ramachandran et al., 2011] Ramachandran, B., Srivastava, S. K., Edrington, C. S. e Cartes, D. A. (2011). An intelligent auction scheme for smart grid market using a hybrid immune algorithm. *IEEE Transactions on Industrial Electronics*, 58(10):4603–4612.

[Schwartz et al., 2014] Schwartz, D., Youngs, N. e Britto, A. (2014). The Ripple protocol consensus algorithm. Relatório técnico, Ripple Labs Inc.

[Stellar, 2018] Stellar (2018). Stellar | move money across borders quickly, reliably, and for fractions of a penny. https://www.stellar.org/. Acessado em 20.08.2018.

[Sui et al., 2009] Sui, H., Wang, H., Lu, M. e Lee, W. (2009). An ami system for the deregulated electricity markets. *IEEE Transactions on Industry Applications*, 45(6):2104–2108.

[Szabo, 1997] Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*, 2(9).

[Tschorsch e Scheuermann, 2016] Tschorsch, F. e Scheuermann, B. (2016). Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys Tutorials*, 18(3):2084–2123.

[Wang e Lu, 2013] Wang, W. e Lu, Z. (2013). Cyber security in the smart grid: Survey and challenges. *Computer Networks*, 57(5):1344 – 1371.

[Wood, 2014] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32.

[Xu et al., 2017] Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C. e Rimba, P. (2017). A taxonomy of blockchain-based systems for architecture design. Em *International Conference on Software Architecture*, ICSA'17, p. 243–252.

[Yan et al., 2011] Yan, Y., Qian, Y. e Sharif, H. (2011). A secure and reliable in-network collaborative communication scheme for advanced metering infrastructure in smart grid. *2011 IEEE Wireless Communications and Networking Conference, WCNC 2011*, (to):909–914.

[Zhu et al., 2011] Zhu, T., Xiao, S., Ping, Y., Towsley, D. e Gong, W. (2011). A secure energy routing mechanism for sharing renewable energy in smart microgrid. *2011 IEEE International Conference on Smart Grid Communications, SmartGridComm 2011*, p. 143–148.