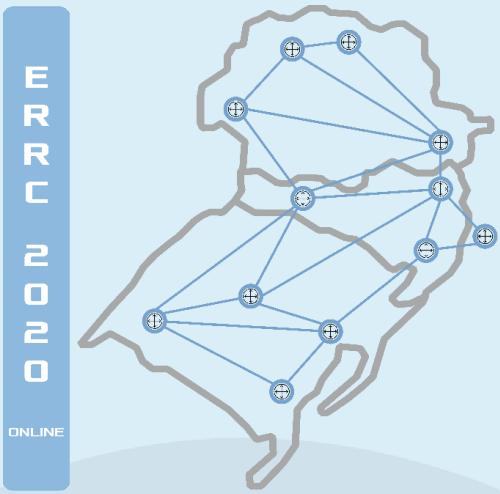


Minicursos da XVIII Escola Regional de Redes de Computadores

Organização e Edição:
Diego Kreutz
Charles C. Miers
Rodrigo B. Mansilha

Realização

SBC
Sociedade Brasileira de Computação

Organização

UDESC
UNIVERSIDADE DO ESTADO DE SANTA CATARINA
LabP2D

unipampa
Universidade Federal do Pampa
PPGES
Programa de Pós-Graduação em Eng. de Software

Patrocínio

Google

18º Escola Regional de Redes de Computadores

ERRC 2020

25 a 27 de novembro de 2020
Joinville-SC, Brasil

MINICURSOS

Editora
Sociedade Brasileira de Computação - SBC
Porto Alegre - RS

Organizadores
Diego Kreutz
Charles Christian Miers
Rodrigo Brandão Mansilha

Realização
Universidade do Estado de Santa Catarina (UDESC)
Universidade Federal do Pampa (UNIPAMPA)

Patrocínio Governamental
UDESC

Patrocínio Empresarial
Google

Promoção
Sociedade Brasileira de Computação - SBC
Comissão Especial de Redes de Computadores e Sistemas Distribuídos
Secretaria Regional do Rio Grande do Sul

Capa: Camila Martini
Supervisão Gráfica: Diego Kreutz, Charles C. Miers e Rodrigo B. Mansilha

Dados Internacionais de Catalogação na Publicação (CIP)

E74

Escola Regional de Redes de Computadores (18.: 2020: Joinville-SC, RS)

Minicursos da XVIII Escola Regional de Redes de Computador [recurso eletrônico] – Organizadores: Diego Kreutz, Charles C. Miers, Rodrigo B. Mansilha – Porto Alegre : SBC, 2020.

ISBN 978-65-87003-24-5

1. Computação - Congresso. 2. Rede de computadores. 3. Segurança da informação. I. Kreutz, Diego. II. Miers, Charles C. III. Mansilha, Rodrigo B. IV. Sociedade Brasileira de Computação. V. Universidade Federal do Pampa. VI. Universidade do Estado de Santa Catarina. VII. Título.

CDU004

Catalogação elaborada por Francine Conde Cabral
CRB-10/2606

Apresentação

A Escola Regional de Redes de Computadores (ERRC 2020), após 17 edições, ocorreu pela primeira vez fora do Rio Grande do Sul (RS), em Joinville, no estado de Santa Catarina (SC), nos dias 25 a 27 de novembro de 2020. É importante destacar também um segundo marco histórico do evento, que foi, pela primeira vez, realizado inteiramente online.

A Escola Regional de Redes de Computadores é promovida pela Sociedade Brasileira de Computação (SBC). Seu principal objetivo é promover a pesquisa e qualificar estudantes e profissionais nas áreas de Redes de Computadores, Sistemas Distribuídos e Segurança da Informação. Com relação à Segurança da Informação, é importante destacar também o Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg 2020), parte integrante da ERRC, que está em sua quinta edição.

A programação da ERRC 2020 e do WRSeg 2020 compreende três minicursos envolvendo diferentes temas na área redes de computadores, sistemas distribuídos e segurança da informação, três palestras e nove sessões técnicas, sendo duas de Iniciação Científica (IC), três Pós-Graduação (PG) e quatro do WRSeg. Neste livro de anais encontram-se os textos dos minicursos da edição de 2020.

Na edição deste ano, organizada pela Universidade do Estado de Santa Catarina (UDESC) e pela Universidade Federal do Pampa (Unipampa), contamos com o patrocínio da Google. Expressamos aqui o nosso sentimento de gratidão ao nosso patrocinador e a todos os colaboradores da organização.

Agradecemos sinceramente a todos os autores dos minicursos, trabalhos técnicos e palestrantes, que contribuíram para o sucesso da ERRC 2020! Também agradecemos aos revisores dos trabalhos da ERRC e do WRSeg, que contribuíram para qualificar os trabalhos da escola.

Diego, Charles e Rodrigo
Organizadores dos Anais e Minicursos da ERRC
Joinville-SC, Brasil, novembro de 2020.

ERRC 2020

18º Escola Regional de Redes de Computadores

Comitê Organizador

Organização Geral
Charles Christian Miers (UDESC)
Rodrigo Brandão Mansilha (UNIPAMPA)

Organização do Comitê de Programa da ERRC
Maurício Aronne Pillon (UDESC)
Guilherme Piêgas Koslovski (UDESC)

Organização Comitê de Programa WRSeg
Rafael Rodrigues Obelheiro (UDESC)

Organização de Minicursos e Oficinas
Diego Kreutz (UNIPAMPA)

Comitê de Programa da ERRC

Adenauer Yamin (UFPEL)
Alberto Schaeffer-Filho (UFRGS)
Andrea Charao (UFSM)
André Riker (UFPA)
Antonio Candia (UFSM)
Antônio Rodrigo D. De Vit (UFSM)
Arthur Lorenzon (UNIPAMPA)
Bruno Dalmazo (UFRGS)
Bruno Vizzotto (UNIPAMPA)
Carlos Raniery Paula dos Santos (UFSM)
Carlos Vinícius Rasch Alves (SENAC-RS)
Charles Christian Miers (UDESC)
Clarissa Marquezan (Huawei)
Claudio Schepke (UNIPAMPA)
Cristiano Both (UNISINOS)
Cristina Nunes (PUC-RS)
César Loureiro (IFRS-Porto Alegre)
Dalvan Griebler (SETREM)
Dartagnan Dias de Farias (SENAC-RS)
Diego Kreutz (UNIPAMPA)
Diogo Menezes Ferrazani Mattos (UFF)
Djamel Sadok (UFPE)
Douglas Macedo (UFSC)
Edison Pignaton De Freitas (UFRGS)
Eduardo Moroñas Monks (SENAC-RS)
Erico Amaral (UNIPAMPA)
Ewerton Salvador (UFPB)
Felipe Nunes (AMF)
Fábio Rossi (PUC-RS)
Gerson Batistti (UNIJUI)
Gerson Soares (UNIMI)
Giani Petri (UFSM)
Glederson Santos (IFSul)
Guilherme Piêgas Koslovski (UDESC)
Guilherme Sperb Machado (UZH)
Juliano Wickboldt (UFRGS)
Leandro Bertholdo (UFRGS)
Leonardo Pinho (UNIPAMPA)
Lisandro Zambenedetti Granville (UFRGS)
Lucas Bondan (UFRGS)
Lucas F. Müller (UFRGS)
Luciano Ignaczak (UNISINOS)
Lucio Prade (UNISINOS)
Luis Augusto Dias Knob (IFRS-Sertão)

Marcelo Luizelli (UNIPAMPA)
Marcia Pasin (UFSM)
Marco Antonio Torrez Rojas (IFSC)
Marco Aurélio Spohn (UFFS)
Matias Schimuneck (UFRGS)
Maurcio Aronne Pillon (UDESC)
Pedro Marcos (FURG)
Pedro Sá da Costa (Copelabs/PT)
Rafael Bastos (UFPEL)
Rafael Esteves (IFRS)
Rafael Kunst (UNISINOS)
Raul Ceretta Nunes (UFSM)
Regio Michelin (IFRS)
Reinaldo Gomes (UFCG)
Ricardo José Pfitscher (UFRGS)
Ricardo Luis dos Santos (UFRGS)
Ricardo Schmidt (UPF)
Roben Lunardi (IFRS)
Rodolfo Antunes (UNISINOS)
Rodrigo Calheiros (Western Sydney University)
Rodrigo Brandão Mansilha (UNIPAMPA)
Rodrigo Righi (UNISINOS)
Roger Immich (IMD/UFRN)
Rogério Turchetti (UFSM)
Simone Ceolin (UFSM)
Tiago Ferreto (PUC-RS)
Valter Roesler (UFRGS)
Vinicius Vielmo Cogo (ULisboa)
Vinícius Guimarães (IFSUL)
Walter Priesnitz Filho (UFSM/CTISM)

Comitê de Programa do WRSeg

Adenauer Yamin (UFPEL)
Bruno Dalmazo (UFRGS)
Bruno Mozzaquatro (UFSM)
Carlos Vinicius Rasch Alves (SENAC-RS)
Charles Christian Miers (UDESC)
Diego Kreutz (UNIPAMPA)
Diogo Menezes Ferrazani Mattos (UFF)
Eduardo Monks (SENAC-RS)
Erico Hoff do Amaral (UNIPAMPA)
Felipe Nunes (AMF)
Luciano Ignaczak (UNISINOS)
Luis Augusto Dias Knob (IFRS-Sertão)
Marco Antônio Sandini Trentin (UPF)
Márcia Henke (UFSM)
Rafael Rodrigues Obelheiro (UDESC)
Reinaldo Gomes (UFCG)
Ricardo Almeida (UFPEL)
Ricardo Schmidt (UPF)
Roben Castagna Lunardi (IFRS-Restinga)
Roberto Samarone Araújo (UFPA)
Rodrigo Brandão Mansilha (UNIPAMPA)
Rogério Turchetti (UFSM)
Tiago Antônio Rizzetti (UFSM)
Walter Priesnitz Filho (UFSM)

SBC

Sociedade Brasileira de Computação

Secretaria Regional do Rio Grande do Sul
Jean Felipe Patikowski Cheiran (UNIPAMPA) - Secretário

Secretaria Regional de Santa Catarina
Everaldo Artur Grahl (FURB)- Secretário

Secretaria Regional do Paraná
Thelma Elita Colanzi (UEM)- Secretária

CRRC

Comitê Assessor da Escola Regional de Redes de Computadores do Rio Grande do Sul

Instituição	Representante
UFSM	Carlos Raniery
UFRGS	Jéferson Nobre
UFRGS	Juliano Wickboldt
IFRS/Sertão	Luis Knob
SENAC/POA	Marcelo Conterato
UFSM	Márcia Pasin
UNIPAMPA	Rodrigo Brandão Mansilha
PUCRS	Tiago Ferreto

Sumário

Capítulo 1

Fake News - Conceitos, métodos e aplicações de identificação e mitigação

Pablo de Andrades Lima, Érico Marcelo Hoff do Amaral, Alex Dias Camargo, Jean Lucas Cimirro e Gérson de Munhos Concilio

Abstract

This meta-article describes a short course designado for the 18th Regional School of Computer Networks. It is intended to address the topic fake news on the perspective of information security, engineering of digital lies, and legislation. It also aims to address some projects, applications and services to combat fake news. Finally, it presents computational methods available to mitigate this very contemporary problem, as it is extremely important to be understood by the academic and scientific community.

Resumo

Este meta-artigo descreve um minicurso elaborado para a 18ª Escola Regional de Redes de Computadores que pretende abordar o tema fake news sobre a óptica da segurança da informação, da engenharia das mentiras digitais, legislações, projetos, aplicações e serviços de combate as notícias falsas. Para finalizar, são abordados alguns métodos computacionais disponíveis para mitigação desse problema tão contemporâneo, pois é de suma importância a ser compreendido pela comunidade acadêmica e científica.

1.1. Introdução

O presente trabalho, foi desenvolvido pelo Grupo de Pesquisas acadêmicas do Pampa sobre disseminação de notícias falsas na *Internet* (Pampa sem *Fake*). Atualmente, faz parte do Programa Universidade *Hacker* - *UniHacker*, sendo composto por alunos do Curso de Engenharia de Computação da Universidade Federal do Pampa, Campus de Bagé e busca estudar, pesquisar e desenvolver possíveis soluções digitais que possam mitigar a disseminação de notícias falsas na *Internet*.

Neste sentido, após pesquisas exploratórias iniciais no ano de 2018, fica exposto que a liberdade e o contexto democrático da *Internet* acabou se tornando um cenário

propício para a propagação de informações, tornando este um meio comumente utilizado para o compartilhamento de falsas notícias, as ditas *fake news*. No Brasil, a propagação desse tipo de informação já ocorre no meio digital, sendo intensificado nos últimos anos durante o período eleitoral [Ruediger et al. 2018].

Após uma análise sistemática das principais redes sociais [Resende et al. 2018], foi estimado que 48% da população brasileira usa o *WhatsApp* (serviço de troca de mensagens via *smartphone* para compartilhar e discutir notícias. O GPOPAI - Grupo de Pesquisa em Políticas Públicas para Acesso à Informação [GPOPAI 2017], em 2017, através de um levantamento, alertou que cerca de 12 milhões de pessoas difundiram notícias falsas sobre política no Brasil.

Outro aspecto importante de citar, trata da influência do "filtro bolha"na difusão de *fake news* nas mídias sociais [Sastre et al. 2018], que tem sido também um dos grandes estimuladores de "bolhas digitais". O "filtro bolha"é um conceito usado para intitular algoritmos que direcionam o acesso ao conteúdo baseado no perfil e hábitos do usuário, dando uma impressão de eficiência na busca mas restringem a maneira de como a pesquisa é realizada, sendo este método muito utilizado pelo *website* de buscas *Google* e pela rede social *Facebook* [Pariser 2011]. Com isso, é possível sugerir que a influência do filtro bolha leva os usuários a bolhas digitais de informações, as quais tornam crenças acima do fato verdadeiro, evocando o que se chama de período "pós-verdade", diminuindo a realidade atual dos fatos objetivando sustentar ideologias e opiniões próprias [Poubel 2018]. Atualmente, existe uma grande gama de serviços, aplicações, projetos e pesquisas e legislações sobre o tema *fake news*. Este documento de minicurso visa navegar por esses tópicos relacionando-os as notícias falsas, levando, assim, conteúdo teórico e prático sobre o tema.

1.2. A engenharia das mentiras digitais

Nesta seção será efetuada uma introdução sobre segurança da informação e sistemas, bem como um histórico sobre o termo *fake news*. Do mesmo modo, serão tratados conceitos relacionados as notícias falsas como filtro bolha, bolha digital e pós-verdade.

1.3. Fake News

Uma das autoras mais citadas quanto a conceituação e especificação do termo *fake news* o classifica como complicado, e considera como um ecossistema de informações com muito mais do que notícias [Wardle 2017]. O termo falso não descreve a complexidade dos diferentes tipos de desinformação, decompondo em três elementos: as motivações de quem cria este conteúdo; as formas como este conteúdo está sendo disseminado; e finalmente, os diferentes tipos de conteúdo que estão sendo criados e compartilhados, os quais são descritos na Tabela 1.1. Para [QUESSADA and PISA 2018] a frase atribuída a Goebbels, Ministro da Propaganda de Adolf Hitler, “uma mentira propagada mil vezes torna-se verdade” é um exemplo de como as *fake news* atuam no período pós-verdade e a velocidade que se espalham nas redes sociais para embasar opiniões.

O uso de robôs em redes sociais é um dos escopos de pesquisa apontado como influenciador na disseminação de notícias falsas. [Ruediger et al. 2018] alertam para preocupação em entender, filtrar e denunciar a disseminação de informações falsas na *Inter-*

Tabela 1.1: Classificação das *fake news* [Wardle 2017]

	Tipo	Descrição
1	Sátira ou paródia	Não quer necessariamente causar mal, mas pode enganar o leitor
2	Falsa Conexão	A chamada da notícia não condiz com conteúdo apresentado
3	Conteúdo Enganoso	Uso mentiroso de uma informação para difamar outro conteúdo ou pessoa
4	Falso Contexto	O conteúdo é verdadeiro, mas é compartilhado com contexto falso
5	Conteúdo Impostor	Quando usa o nome de uma pessoa ou marca, mas afirmações irreais
6	Conteúdo Manipulado	O conteúdo verdadeiro é alterado para enganar o público
7	Conteúdo Fabricado	Informações 100% falsas e construídas para causas mal e espalhar boatos

net com a utilização de ferramentas automatizadas (robôs sociais) que usufruem de perfis falsos, não reais, se passando por seres humanos e participando ativamente de debates políticos de grande repercussão; disseminando informações falsas, promovendo *hashtags* e massificando postagens automatizadas que comprometem o debate espontâneo através de *softwares* que geram este tipo de conteúdo artificialmente.

No estudo sobre "filtros bolha"de [Sastre et al. 2018], foram citadas as mudanças realizadas pelo *Facebook* com implantação do sistema de *crowdsourcing*, ou colaboração coletiva, que define as prioridades dos *feeds* de notícias nos perfis dos usuários, classificando o que irá aparecer ao usuário por uma maior familiaridade com os conteúdos mais acessados, com propósito de reduzir a difusão de *fake news* através de robôs. Porém, essa configuração gerou uma repercussão negativa com empresas que utilizam estratégias de divulgação por meio de mídias digitais. De acordo com Sérgio Dávila, editor-executivo do jornal a Folha de São Paulo, considerado o maior jornal do Brasil, as redes sociais tendem a criar "bolhas"e "condomínios de convicções"forçando as pessoas a se relacionar somente com outras que pensam como elas [Caulyt 2018].

A pesquisa de [Ferrara et al. 2016] mostra uma tendência promissora de evolução no combate às *fake news* utilizando o padrão automatizado de *Machine Learning* (ML) e inteligência humana para diferenciar robôs de pessoas. Não menos obstante, cabe citar o crescimento das *deepfakes*, uma técnica usada para substituir rostos originais em vídeos e utilizados para disseminar notícias falsas com uso de aplicativos para a troca de rosto, gerando grande quantidade de vídeos sinteticamente manipulados e distribuídos nas redes sociais, representando um grande desafio técnico para detecção e filtragem de tal conteúdo [de Moraes 2019].

O uso de perfis falsos para disseminação de notícias acaba sendo também um objeto de estudo de suma importância, pois esta técnica que é usada por empresas impulsionadoras de conteúdo ou pessoas comuns, muitas vezes tem como objetivo na propagação

de conteúdos falsos que são de seu interesse. Sabendo que a criação de perfis *fakes* é umas das formas mais utilizadas dentro das redes sociais para este fim específico, foi realizado um estudo baseado na criação de perfis falsos nas principais redes sociais, como, *Twitter e Facebook*, com a finalidade de identificar de que forma as redes tratam este tipo de prática, objetivando a mitigação de perfis que podem ser usados por robôs para disseminação de informações falsas e na criação em massa de contas em suas mídias.

Embora algumas redes sociais não possuam vínculo entre si, é notável a existência de um padrão de verificação de usuários durante a criação de perfis dentro das mesmas, onde é necessário efetuar a entrada com um *e-mail* para realizar a verificação inicial a qual já é um fator de validação. Com isso em vista, durante o procedimento de criação de contas *fake* foi utilizado o serviço chamado *Temp-Mail*[1], no qual foram gerados *emails* temporários que foram utilizados para realizar a ativação de tais perfis. Em ambas as redes sociais, em casos aleatórios, foi preciso de uma outra confirmação, por telefone, nestes casos foi necessário o uso de outra ferramenta, o aplicativo *2ndline - US Phone Number*, disponível para *Android* no *Play Store*, ao qual sua finalidade é disponibilizar um número de telefone celular norte-americano que recebe e encaminha mensagens SMS para a ativação de serviços *online*.

Nesse aspecto, foi notado que por questões publicitárias, ainda existem pequenas brechas para atuação de perfis falsos e até comércio deste tipo de contas conforme mostra (item 1) da Tabela 1.2 com *links* de acesso dos endereços. Há ainda no *Youtube* diversos vídeos e canais (item 2) que tratam da criação de perfis falsos para utilização em *marketing* digital, além de agências que prometem o impulsionamento digital através do uso de robôs (item 3).

Tabela 1.2: Lista de *links* de endereços

Item	Endereço	Descrição
1	http://www.fbpvastore.com/	Loja de Perfis
2	https://youtu.be/0eM8uFn1PAs	Canal de *marketing* digital
3	https://www.autland.com/	Agência de automação social

Desta forma, é possível perceber que o uso de perfis não verdadeiros ainda é um problema e, esta prática pode ser fortemente utilizada para quem tem por objetivo a disseminação de notícias falsas, pelo fato de que sua disseminação é muito rápida, principalmente quando esta notícia se é compartilhada em grupos de bolhas sociais, onde os assuntos são filtrados e os usuários recebem apenas determinados tipos de conteúdo. Sendo assim é importante que as pessoas tenham algum tipo educação digital para que possam confirmar em canais confiáveis de informação se determinado assunto é ou não verdadeiro e, se mesmo após essa reflexão ainda existir alguma dúvida, ir buscar informações a respeito para determinar se esta notícia é verídica.

[1] https://temp-mail.org

1.4. Legislação

As tecnologias de informação e comunicação, assim como ambientes computacionais disponibilizaram nos últimos anos um conjunto de soluções para os usuários, como computadores de alto desempenho, dispositivos móveis, redes com altas taxas de velocidade e a *Internet*. Contudo, a utilização destes recursos nem sempre está direcionada a atividades lícitas, muitas vezes sendo estes mecanismos empregados em diferentes práticas ilegais, dentre elas se destacam a difusão de informações falsas e não verídicas. Nesse contexto, a presente seção deste minicurso relaciona a disseminação de noticias falsas com um dos mais importantes princípios da área de segurança da informação, a Legalidade, a qual descreve que o uso da Tecnologia de Informação e Comunicação deve seguir as leis vigentes do local ou país.

A seguir serão apresentadas, em relação as *fake news*, a interpretação do Código Penal Brasileiro sobre o tema, o Projeto de Lei 2360, o Código Eleitoral através da Lei 13.834 de 04 de junho de 2019, o Marco Civil da Internet (Lei nº 12.965, de 23 de abril de 2014) e a LGDP, Lei nº 13.709

O termo do inglês "*fake news*", ou seja, "notícias falsas"o qual se refere a qualquer notícia ou informação não verídica (falsa ou mentirosa) compartilhadas através de meios digitais como se verdade fosse. Esse tipo de mensagem é comumente disponibilizada por meio de redes sociais, aplicativos ou qualquer outro meio de comunicação na Internet. Neste contexto, o Código Penal brasileiro não previa tal disseminação como crime, visto que, Art. 5º, Inciso XXXIX, da Constituição Federal de 1988 proclama: "não há crime sem lei anterior que o defina, nem pena sem prévia cominação legal"(Princípio da Legalidade). Desta forma, o Código Penal e sua legislação não define como crime a divulgação de notícias falsas, tanto pela ausência de previsão de seu tipo normativo, assim como pela ausência de qualquer cominação de pena.

Contudo, o Projeto de Lei n° 2630, de 2020, "estabelece normas relativas à transparência de redes sociais e de serviços de mensagens privadas, sobretudo no tocante à responsabilidade dos provedores pelo combate à desinformação e pelo aumento da transparência na *Internet*, à transparência em relação a conteúdos patrocinados e à atuação do poder público, bem como estabelece sanções para o descumprimento da lei". Este projeto de lei, também denominada como Lei das *Fake News*, é um projeto de lei proposto pelo Senador Alessandro Vieira (CIDADANIA/SE) e definido como a Lei Brasileira de Liberdade, Responsabilidade e Transparência na *Internet*. Essa lei, já em seu primeiro artigo, "estabelece normas, diretrizes e mecanismos de transparência para provedores de redes sociais e de serviços de mensagens privada a fim de garantir segurança, ampla liberdade de expressão, comunicação e manifestação do pensamento."

O artigo 3° desta lei descreve algumas boas práticas, como medidas adequadas e proporcionais no combate ao comportamento inautêntico e na transparência sobre conteúdos pagos:

a. Liberdade de expressão e de imprensa;

b. Garantia dos direitos de personalidade, dignidade, honra e privacidade;

c. Respeito à formação de preferências políticas e de uma visão de mundo pessoal do

usuário;

d. Compartilhamento da responsabilidade de preservação de uma esfera pública livre, plural, diversa e democrática;

e. Garantia da confiabilidade e da integridade de sistemas informacionais;

f. Promoção do acesso ao conhecimento de assuntos de interesse público;

g. Proteção dos consumidores; e

h. Transparência nas regras para anúncios e conteúdos patrocinados.

Além disso, o PL 2630 veda terminantemente: contas inautênticas, disseminadores artificiais não rotulados, redes de disseminação artificial que disseminem desinformação e conteúdos patrocinados não rotulados. A fim de atender Leis Eleitorais, este PL prevê a disponibilização a Justiça Eleitoral de dados relacionados a publicações politicas direcionadas a candidatos, partidos ou coligações, que tenham sido impulsionadas ou disseminadas, conforme descrito no art. 16 "Os provedores de redes sociais devem disponibilizar mecanismos para fornecer aos usuários as informações do histórico dos conteúdos impulsionados e publicitários"com os quais a conta teve contato nos últimos 6 (seis) meses."

Entre outros pontos importantes tem-se os art. 18 e 24 que tratam obre a responsabilidade de agentes políticos em mandatos eletivos e a proteção do direito de expressão e publicação de conteúdos por servidores públicos em suas contas privadas, de forma que não sejam perseguidos ou prejudicados por tais ações. Em relação as sanções, previstas neste Projeto de Lei, observa-se em seu Capítulo VI: art. 31. Sem prejuízo das demais sanções civis, criminais ou administrativas, os provedores de redes sociais e de serviços de mensagens privada ficam sujeitos a: I – advertência, com indicação de prazo para adoção de medidas corretivas; ou II – multa de até 10% (dez por cento) do faturamento do grupo econômico no Brasil no seu último exercício. §1º Na aplicação da sanção, a autoridade judicial observará a proporcionalidade, considerando a condição econômica do infrator, as consequências da infração na esfera coletiva e a reincidência. §2º Para os efeitos desta Lei, será considerado reincidente aquele que repetir no prazo de 6 (seis) meses condutas anteriormente sancionadas[2].

Em relação ao Código Eleitoral Brasileiro existe a Lei 13.834 de 04 de junho de 2019, a qual altera a Lei nº 4.737, de 15 de julho de 1965 (Código Eleitoral, para tipificar o crime de denunciação caluniosa com finalidade eleitoral). Esta lei pune com dois a oito anos de prisão quem divulgar notícias falsas com finalidade eleitoral. A lei havia sido sancionada originalmente em junho, mas um veto parcial tinha deixado de fora o dispositivo que criminaliza a disseminação de *Fake News* nas eleições.

Ao se tratar do Marco Civil da Internet torna-se claro o avanço desta Lei no trato jurídico das relações derivadas do uso da internet. Quando se discute a questão das *Fake News*, principalmente em períodos eleitorais (períodos normalmente rápidos e curtos), nota-se que a disseminação de noticias falsas pode culminar em alterações dos resultados

[2]https://www.politize.com.br/lei-das-fake-news/

de um pleito. Neste sentido é claro que o fator tempo é muito importante para se evitar problemas e ilícitos, neste contexto a utilização do *notice and take down* (baseia-se na notificação extra judicial para a remoção de noticias falsas no período de 24h, sendo o provedor solidário em responder por tal ilícito). Com a criação do Marco Civil da Internet passou-se a adotar o judicial *notice and take down*, ou seja, atualmente é necessária a notificação judicial para a remoção de determinado conteúdo do ar, o que amplia o tempo de resposta dos provedores, aumentando desta forma o fator de impacto do problema. Sendo assim, é notório que o Marco Civil da Internet é insuficiente em relação a propagação de notícias na Internet. Para ajustar estas brechas tem-se outros projetos como o Projeto de Lei n. 5.203 de 2016, que exige em sua redação, a indisponibilização por parte dos provedores de informações em um prazo de 48h após o recebimento de notificação[3].

Por fim, destaca-se a Lei Geral de Proteção de Dados (LGPD) a qual consiste na legislação brasileira que visa regulamentar a aquisição e o tratamento de dados pessoais no Brasil. Esta Lei por si só não é um instrumento específico para o combate a disseminação de *Fake News*, contudo permite a regulação do tipo e volume de dados que as empresas, detentoras de informações, poderão manipular e, com isso estar e conformidade com a lei específica para o combate a proliferação de notícias falsas, com base na segurança e salvaguarda de tais dados.

1.5. Aplicações e serviços de combate as notícias falsas

Nesta seção serão apresentados alguns projetos atuais, voltados para o combate a desinformação, os quais se baseiam na formação, educação midiática e conscientização, pontos estes considerados fundamentais para a mitigação do compartilhamento e notícias falsas e desinformação.

O primeiro serviço a ser abordado é o *Check*, uma ferramenta desenvolvida pela [Meedan 2020][4], que é uma organização sem fins lucrativos, foca na melhora da qualidade e equidade dos dados *online*, voltadas para redações, ONGs e instituições acadêmicas. É um espaço de trabalho *online* que permite aos usuários verificar fotos e texto *online* e criar conjuntos de dados. O *Check* ajuda jornalistas, organizações da sociedade civil, pesquisadores e investigadores de direitos humanos na vanguarda da coleta e verificação de informações. Durante o dia das eleições presidenciais de 2016 nos Estados Unidos, a *Check* apoiou o *Electionland*, um esforço nacional de cobertura de problemas de votação durante as eleições de 2016. A ferramenta utiliza uma estrutura de *workflow*, onde pessoas analisam e classificam dados, imagens e textos para uso na checagem de fatos. A empresa Meedan desenvolveu um conjunto de ferramenta para verificação de fatos interligada ao *WhatsApp, Facebook Messenger, WeChat* e outros aplicativos de troca de mensagens, ela permite que usuários encaminhem mensagens para um organização de verificação de fatos, atualmente em cinco países, Brasil, Africa do Sul, Índia, Quênia e Nigéria. Ao realizar uma solicitação os usuários receberão automaticamente os resultados dessa checagem de fatos, junto com algumas informações simples sobre por que a conclusão foi alcançada e um cartão visual que é projetado para ser compartilhável, a ferramenta pode

[3]https://jus.com.br/artigos/69900/a-insuficiencia-do-marco-civil-da-internet-em-relacao-as-fake-news-nas-eleicoes

[4]https://meedan.com/check

receber solicitações em qualquer idioma, porém sua interface está atualmente disponível em inglês, espanhol, francês, português, árabe, russo e romeno. O principal diferencial do *Check* em comparativo a outras agências de checagem de fatos se dá pelo motivo de que uma informação que anteriormente foi verificada pelo seus colaboradores é armazenada em um banco de dados e assim utilizando de *Machine Learning*, para fazer com que quando ocorra uma nova requisição por uma informação, primeiramente ela seja comparada com o conteúdo do banco de dados a fim de verificar se possui alguma reportagem ou informação similar ou igual a alguma que já foi verificada, fazendo com que a equipe de checagem não precise verificar novamente uma mesma informação fazendo com que os responsáveis por averiguar as informações tenham como foco em buscas ainda não classificadas.

O *Hoaxy*[5] é uma ferramenta *open source* desenvolvida pela Universidade de Indiana-IUNI, pelo CNetS e o *Observatory on Social Media*, com objetivo principal visualizar a disseminação de artigos online através do Twitter [Shao et al. 2016]. De acordo com autor a plataforma realiza coleta de dados, detecção e análise de desinformação online e direciona para verificação de fatos relacionados. Frisa que de modo geral, o compartilhamento de conteúdo de verificação de fatos geralmente fica atrás da desinformação entre 10 e 20 horas. Além disso, notícias falsas são dominadas por usuários muito ativos, enquanto a verificação de fatos é uma atividade mais básica. Com os riscos crescentes relacionados à enorme desinformação online, os observatórios de notícias sociais têm o potencial de ajudar pesquisadores, jornalistas e o público em geral a entender a dinâmica do compartilhamento de notícias reais e falsas.

Com algumas semelhanças, temos o *bot sentinel*[6], que trabalha com uso de aprendizado de máquina projetada para detectar robôs. O sistema pode classificar corretamente as contas com uma precisão de 95%, concentrando em comportamentos e atividades específicos considerados inadequados pelas regras do *Twitter*. Os pesquisadores analisaram contas que violavam repetidamente as regras do *Twitter* e treinaram o modelo para classificar contas semelhantes às que foram identificadas como "problemáticas". O *website* analisa centenas de *tweets* para classificar com precisão cada conta do *Twitter* e fornecer um relatório fácil de entender. Porém ele é incapaz de classificar se uma informação é verdadeira ou falsa pelo fato dele apenas comparar informações que são postadas por usuários com conta pública, a checagem é realizada apenas por máquina pois ele analisa apenas a disseminação das postagens e não seu conteúdo, pois o *bot sentinel* não possui um time humanizado de checagem.

Uma observação importante é que ideologia, afiliação política, crenças religiosas, localização geográfica ou frequência de *tweets* não são fatores para determinar a classificação de uma conta do *Twitter*. As contas são classificadas com base em uma pontuação de 0% a 100%; quanto maior a pontuação, maior a probabilidade da a conta se envolver em assédio direcionado, "trolagem tóxica"ou usar táticas enganosas projetadas para causar divisão e caos. Outro detalhe importante é análise de contas falsas que fazem parte de uma grande conspiração que tenta influenciar as políticas e/ou eleições ou disseminam campanhas de influência. Uma conta falsa que está ativamente tentando causar divisão e

[5]https://hoaxy.iuni.iu.edu
[6]https://botsentinel.com/

discórdia se comportará de maneira consistente com alguém que recebe uma pontuação alta do *bot sentinel*

Cabe citar ainda o NILC-USP – Detecção Automática de Notícias Falsas para o Português[7], desenvolvido por pesquisadores do Núcleo Interinstitucional de Linguística Computacional (NILC), da USP e da Universidade Federal de São Carlos (UFSCar), através de análise textual, com uso de inteligência artificial, para servir de apoio ao usuário na identificação de notícias falsas com uma precisão de 90% [Monteiro et al. 2018]. Ao receber um texto, o sistema aplica métodos para extrair atributos linguísticos desse texto e os utiliza em um modelo de aprendizado de máquina, que classifica a notícia como verdadeira ou falsa. O texto deve ter pelo menos 100 palavras. A partir destas palavras, ele irá verificar em dois modelos de detecção: Palavras do Texto e Classes Gramaticais. O primeiro modelo, é baseado em comparações sucessivas em um banco de 10395 palavras visando buscar palavras iguais as quais estão contidas no texto inserido pelo usuário, ao qual irá gerar uma pontuação para o texto informado, fazendo com que textos com uma maior ocorrência das palavras sejam mais plausíveis de veracidade. O segundo modelo irá calcular a porcentagem de classes gramaticais que está contida no texto a partir de uma biblioteca que está disponível na linguagem de programação Python(nlpnet [8]). Com as pontuações geradas pelos dois métodos, é aplicado um classificador que definirá se a notícia é verdadeira ou não. É importante ressaltar que os projetistas da ferramenta, não aconselham que checagem dos fatos seja feita apenas pela própria, pois a mesma usa apenas uma análise textual, sendo assim necessário uma humana para uma maior confiabilidade nos resultados.

Todas as ferramentas aqui descritas possuem algum método específico de atuação dentro de algum escopo, mas vale destacar [Monteiro et al. 2018] que também atua com uso de IA para classificação de notícias falsas.

1.6. Métodos computacionais disponíveis para mitigar

Nesta seção será feita uma explanação sobre Inteligência Artificial (IA) e Processamento de Linguagem Natural (do inglês, *Natural Language Processing*, NLP), assim como essas tecnologias podem ajudar computacionalmente no combate as *fake news*. Um exemplo prático implementado pode ser encontrado no repositório de códigos-fonte *GitHub* [9].

O termo Inteligência Artificial foi criado por [McCarthy et al. 1956] nos trabalhos do "*Dartmouth Summer Research Project on Artificial Intelligence*", os quais foi inserido como um novo campo de conhecimento associando linguagens e inteligência, raciocínio, aprendizagem e resolução de problemas. A comparação de humano e máquina a muito tempo é objeto de estudo, visto que [TURING 1950] já instigava reflexões como a questão: "Podem as máquinas pensar?", propondo o Jogo da Imitação. Em seus estudos, o autor esperava que máquinas e homens acabariam por competir em campos puramente intelectuais, porém encerrava com a afirmação: "Podemos avistar só um pequeno trecho do caminho à nossa frente, mas ali já vemos muito do que precisa ser feito"[TURING 1950].

Basicamente, um dos objetivos desta pesquisa é utilizar técnicas de Aprendizado

[7]https://nilc-fakenews.herokuapp.com/
[8]http://nilc.icmc.usp.br/nlpnet/
[9]https://github.com/alexcamargoweb/wrseg-2020/

de Máquina (AM), ou, em inglês, *Machine Learning* (ML). O tema trata de uma área da Inteligência Artificial que tem por objetivo a construção de sistemas computacionais capazes de adquirir conhecimento de forma automática, tomando suas decisões baseado em experiências acumuladas através de problemas anteriores com solução bem sucedida [Monard and Baranauskas 2003]. De acordo com [Murphy 2012], no AM, os algoritmos são capazes de realizar previsões de padrões em conjunto de dados pré-estabelecidos, podendo ser, essencialmente, classificados em dois tipos de aprendizado: supervisionado (preditivo) ou não-supervisionado (descritivo). Na abordagem aqui tratada foram utilizadas Redes Neurais Artificiais (RNAs), que são algoritmos (supervisionados) inspirados biologicamente no sistema nervoso de animais, úteis para processar grandes quantidades de dados de treinamento e objetivo de classificar novas instâncias [MARUMO 2018]. A arquitetura de uma RNA apresenta camadas de neurônios de processamento e conexões entre eles, onde durante o processo de treinamento são atribuídos pesos para cada conexão que servirão como determinantes de quais neurônios serão ativados na próxima camada, simulando as sinapses de um sistema nervoso animal [Bahdanau et al. 2014]. O processamento de cada neurônio é feito através de uma função de ativação que neste caso possui diversos tipos, onde a escolha desta função determina como devem ser as entradas da rede e como será o resultado de saída da mesma.

Com a evolução das RNAs, surgiu o termo *Deep Learning* (DL) ou Aprendizado Profundo, devido às muitas camadas por ela adotadas [Nallapati et al. 2016]. No contexto desta seção, esse tipo de aprendizagem, chamado Mineração de Texto, tende a identificar padrões de texto e analisá-los com uma técnica chamada *text summarization*. Posteriormente, os valores são classificados de acordo com as métricas pré-definidas e os pesos atribuídos a rede [MARUMO 2018].

Como parte de suma importância sobre esse tema, a mineração de texto pode ser compreendida pela técnica de Processamento de Linguagem Natural (conhecido amplamente por NLP). Essencialmente, trata da utilização de métodos e recursos computacionais para análise de dados linguísticos [Sakurai 2019]. Nesse contexto, uma implementação de NLP auxilia na resolução de ambiguidades estruturando numericamente o texto, separando palavras através num processo de "*tokenização*", processando-o com análises léxicas, sintáticas, semânticas e pragmáticas, para posterior compreensão do algoritmo de IA [Dale et al. 2000].

Em resumo, complementando e relacionando os conceitos aqui abordados, fica compreendido que dentro do conceito de IA, possuímos: o Aprendizado de Máquina (geralmente, supervisionado ou não supervisionado) como técnica de inferência; as Redes Neurais Artificiais como tipo de modelo preditivo e o *Deep Learning* como uma RNA com muitas camadas interligadas. Por fim, foi disponibilizado um *link* contendo uma implementação de uma NLP em *Python* via *Colab*, um ambiente em nuvem de alto desempenho mantido pelo *Google Research*.

1.7. Projetos sobre *Fake News*

Nesta seção, serão apresentados e listados na Tabela 1.3 alguns projetos que estão colaborando para o combate a desinformação atualmente. O Fato sem *Fake*-FsF, também da Universidade Federal do Pampa, é parte integrante de um projeto de pesquisa e exten-

são do Grupo *t3xto*. A proposta visa contribuir socialmente com conhecimentos sobre *fake news* e desinformação, para formar combatentes da "infodemia"que corrompe o bom senso democrático. O projeto é desenvolvido através de produção de *podcasts* apresentados e produzidos pelo Prof. Marco Bonito e seus orientandos de iniciação científica: Gabriel Pujol, Luana Kasper e Emília Sosa, além do suporte do técnico de áudio Pedro Janczur. O projeto é financiado pela Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS).

Tabela 1.3: Projetos e ações de combate a notícias falsas

Nome	*Link* de acesso
[t3xto 2020]	https://anchor.fm/fato-sem-fake
[Paganotti et al. 2019]	https://vazafalsiane.com
[Sayad 2019]	https://educamidia.org.br

O projeto Vaza, Falsiane! é um curso *online* de iniciativa de três amigos, jornalistas e professores universitários que ao longo dos últimos dois anos estudam as *fake news*, investigando as melhores formas de produzir conteúdo sobre o assunto para um público amplo. Nesse período, foi incubado pela ONG Repórter Brasil e venceu um edital de financiamento do *Facebook*.

A EducaMídia, programa do Instituto Palavra Aberta com apoio do *Google.org* é uma entidade sem fins lucrativos que advoga a causa da plena liberdade de ideias, de pensamento e de opiniões. Com suas pesquisas, seminários e campanhas, promove a liberdade de expressão, a liberdade de imprensa e a livre circulação de informação como pilares fundamentais para o desenvolvimento de uma sociedade forte e democrática. O projeto Educamídia foi criado para capacitar professores e organizações de ensino, engajando a sociedade no processo de educação midiática dos jovens, desenvolvendo seus potenciais de comunicação nos diversos meios. Atualmente, atua no desenvolvimento de três competências: interpretação crítica das informações, produção ativa de conteúdos e participação responsável na sociedade. Possui atuação também na formação de professores e educadores, no apoio a formuladores de políticas públicas e na sensibilização para o tema. A plataforma centraliza conteúdos para formação e pesquisa, além de materiais e recursos para a sala de aula alinhados com a Base Nacional Comum Curricular (BNCC).

Outra ação de mitigação das notícias falsas, são as agências checagem de fato, *fact checking*, que dão credibilidade as notícias *online*. De acordo com [Fatos 2018], a checagem de fatos é um método jornalístico por meio do qual é possível certificar se a informação apurada foi obtida por meio de fontes confiáveis e, então, avaliar se é verdadeira ou falsa, se é sustentável ou não. Temos no Brasil hoje algumas agência de checagem de fatos como Aos Fatos[10] e Publica[11]. Alguns portais de notícias também tem oferecido esse tipo de serviço como o G1 que possui o portal Fato ou Fake[12] que atua de forma parecida as agencias de *fact checking*.

Encerrando esta seção e não menos importante, salientamos os esforços realizados

[10]https://www.aosfatos.org/
[11]https://apublica.org/
[12]https://g1.globo.com/fato-ou-fake/

pelo TSE-Tribunal Superior Eleitoral em combater as notícias falsas, que no meio político são comumente usadas. O portal da Justiça Eleitoral Brasileira, neste ano de 2020 com as eleições municipais, possui o projeto Fato ou Boato, criado em 2016 para ampliar o esclarecimento de informações relacionadas ao processo eleitoral, a página Fato ou Boato fomenta a circulação de conteúdos verídicos e estimula a verificação por meio da divulgação de notícias checadas, recomendações e conteúdos educativos.

Essa iniciativa integra o Programa de Enfrentamento a Desinformação nas Eleições 2020, que atualmente mobiliza mais de 50 instituições, entre partidos políticos e entidades públicas e privadas, para enfrentar os efeitos negativos provocados pela desinformação relacionada à democracia.

Tendo como maior objetivo o enfrentamento das *fake news*, nove das principais agências de checagem do Brasil integram essa força-tarefa em favor da circulação de conteúdos verificados, que efetivamente promovam debates e esclarecimentos fundamentais à tomada de decisão do eleitor.

Na centralidade da desinformação propagada em anos eleitorais, estão as notícias falsas sobre a urna eletrônica, que no portal dispões de diversas informações desmistificam dúvidas, fatos e boatos sobre o processo eleitoral eletrônico no Brasil. Outra ação importante desenvolvida pelo Tribunal, é a série de vídeos que apresenta de maneira simples e didática os passos que qualquer cidadão pode adotar para verificar conteúdos e se tornar um agente de combate à desinformação.

1.8. Considerações Finais

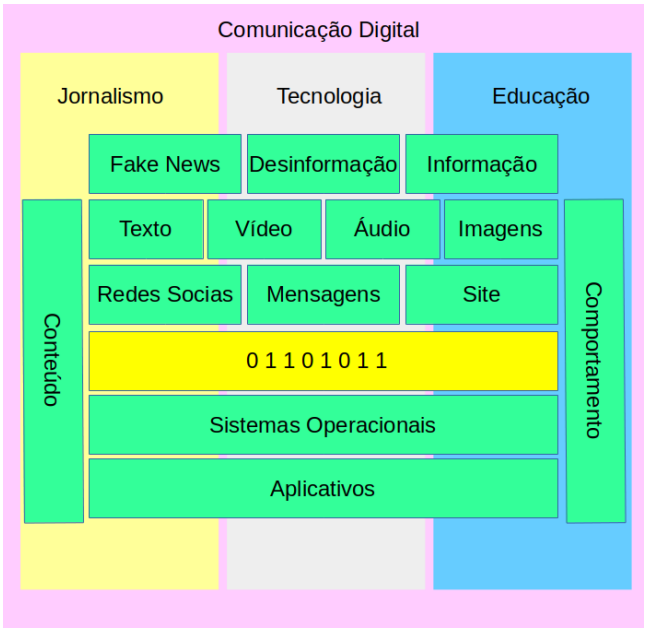
Uma das maneiras de poder visualizar o problema e suas possíveis soluções é a separação dele por "caixas", ilustrado na Figura 1.1 onde os possíveis métodos de solução atuariam especificadamente dentro de uma ou mais "caixas".

O diagrama mostra ao fundo um problema de comunicação digital que no centro na verdade temos os dados em formato de *bits*, ou seja, a sua atuação e modo de operação é digital, tendo como áreas de atuação na sua possível solução, o jornalismo, educação e tecnologia. Podemos compreender também que se trata de um problema segmentado em *fake news*, desinformação e informação, onde esses dados transitariam em formatos de vídeos, texto, imagem ou áudio, por redes sociais, sites ou aplicativos de mensagens, sendo que estes poderiam ser tratados ou auditados através de uma análise de comportamento ou conteúdo com uso de aplicativos ou diretamente do sistema operacional.

Mesmo assim é possível verificar que boa parte das ferramentas tecnológicas que contribuem para redução da propagação de *fake news* através de robôs digitais, aproveitando do impulsionamento dos algoritmos de “filtro bolha”, utiliza técnicas de *Deep Learning*.

O uso de perfis não verdadeiros ainda é um problema e esta prática vem sido fortemente usada para quem tem por objetivo a disseminação do notícias falsas. De acordo com [Ferrara et al. 2016], pesquisas mostram uma tendência promissora de evolução no combate às *fake news* utilizando o padrão automatizado de *Machine Learning* e inteligência humana para diferenciar robôs de pessoas.

Figura 1.1: Diagrama sinóptico



Como métodos viáveis para a solução das *fake news*, partindo do pressuposto que a notícia falsa se propaga com maior proporção do que uma notícia verdadeira, é possível trabalhar também com a disseminação de fatos verídicos e, não somente com o combate à informações falsas. Mesmo assim, é possível verificar a dificuldade no tratamento de notícias falsas nos meios digitais atuais como redes sociais. Devido a complexidade do tema, legislações e algoritmo devem tomar todo cuidado no que tange a liberdade de expressão, sendo ainda a conscientização dos usuários a melhor forma de prevenção.

Referências

[Bahdanau et al. 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Caulyt 2018] Caulyt, F. (2018). Facebook perdeu importância para a folha. *diz editor. Deutsche Welle Brasil, versão online, Boon (Alemanha)*, 9.

[Dale et al. 2000] Dale, R., Moisl, H., and Somers, H. (2000). *Handbook of natural language processing*. CRC Press.

[de Moraes 2019] de Moraes, C. P. (2019). "deepfake" como ferramenta manipulação e disseminação de "fakenews" em formato de vídeo nas redes sociais. *OSF Preprints*.

[Fatos 2018] Fatos, A. (2018). O que é checagem de fatos—ou fact-checking. *Acesso em 5 de agosto de 2020*, 12.

[Ferrara et al. 2016] Ferrara, E., Varol, O., Davis, C., Menczer, F., and Flammini, A. (2016). The rise of social bots. *Communications of the ACM*, 59(7):96–104.

[GPOPAI 2017] GPOPAI (2017). Públicas para o acesso à informação. Technical report, da EACH/USP–GPOPAI. Subsídio público e acesso ao conhecimento. 2017 ....

[MARUMO 2018] MARUMO, F. S. (2018). Deep learning para classificação de fake news por sumarização de texto.

[McCarthy et al. 1956] McCarthy, J., Minsky, M., and Rochester, N. (1956). The dartmouth summer research project on artificial intelligence. *Artificial intelligence: past, present, and future*.

[Meedan 2020] Meedan (2020). Check.

[Monard and Baranauskas 2003] Monard, M. C. and Baranauskas, J. A. (2003). Conceitos sobre aprendizado de máquina. *Sistemas inteligentes-Fundamentos e aplicações*, 1(1):32.

[Monteiro et al. 2018] Monteiro, R. A., Santos, R. L. S., Pardo, T. A. S., de Almeida, T. A., Ruiz, E. E. S., and Vale, O. A. (2018). Contributions to the study of fake news in portuguese: New corpus and automatic detection results. In Villavicencio, A., Moreira, V., Abad, A., Caseli, H., Gamallo, P., Ramisch, C., Gonçalo Oliveira, H., and Paetzold, G. H., editors, *Computational Processing of the Portuguese Language*, pages 324–334, Cham. Springer International Publishing.

[Murphy 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

[Nallapati et al. 2016] Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. (2016). Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

[Paganotti et al. 2019] Paganotti, I., Sakamoto, L. M., and Ratier, R. P. (2019). "mais fake e menos news": resposta educativa às notícias falsas nas eleições de 2018. *Liberdade de Expressão Questões da atualidade*, page 52.

[Pariser 2011] Pariser, E. (2011). *The filter bubble: What the Internet is hiding from you*. Penguin UK.

[Poubel 2018] Poubel, M. (2018). Fake news e pós-verdade. *Infoescola. Sociedade. s/d. Disponível em< https://www. infoescola. com/sociedade/fake-news*, 15.

[QUESSADA and PISA 2018] QUESSADA, M. and PISA, L. F. (2018). Fake news versus mil: a difícil tarefa de desmentir goebbels.

[Resende et al. 2018] Resende, G., Messias, J., Silva, M., Almeida, J., Vasconcelos, M., and Benevenuto, F. (2018). A system for monitoring public political groups in whatsapp. In *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*, pages 387–390.

[Ruediger et al. 2018] Ruediger, M. A., Grassi, A., and Guedes, A. L. (2018). Robôs, redes sociais e política no brasil: análise de interferências de perfis automatizados de 2014.

[Sakurai 2019] Sakurai, G. Y. (2019). Processamento de linguagem natural-detecção de fake news.

[Sastre et al. 2018] Sastre, A., de Oliveira, C. S. P., and Belda, F. R. (2018). A influência do "filtro bolha" na difusão de fake news nas mídias sociais: reflexões sobre as mudanças nos algoritmos do facebook. *Revista GEMInIS*, 9(1):4–17.

[Sayad 2019] Sayad, A. L. V. (2019). Educação midiática e pensamento crítico: antídotos contra a "desinformação". *Liberdade de Expressão Questões da atualidade*, page 9.

[Shao et al. 2016] Shao, C., Ciampaglia, G. L., Flammini, A., and Menczer, F. (2016). Hoaxy: A platform for tracking online misinformation. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, page 745–750, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

[t3xto 2020] t3xto, G. (2020). Pampa sem fake. Access date: 04 nov. 202020.

[TURING 1950] TURING, I. B. A. (1950). Computing machinery and intelligence-am turing. *Mind*, 59(236):433.

[Wardle 2017] Wardle, C. (2017). Fake news. it's complicated. *First Draft*, 16.

Capítulo

2

Blockchains com Hyperledger: conceitos, instalação, configuração e uso

Charles Christian Miers, Guilherme Piêgas Koslovski, Maurício Aronne Pillon, Marco Antonio Marques

PPGCA - LabP2D - UDESC

Resumo

A introdução dos contratos inteligentes possibilitou o desenvolvimento de uma série de inovações, tais como aplicações e organizações autônomas descentralizadas, propriedade inteligente, tokens inteligentes, dentre outros, que permitiram que a blockchain fosse além das soluções financeiras, fato conhecido como Blockchain 3.0. Dentre os diversos modelos de Blockchain 3.0, o Hyperledger é uma plataforma Distributed Ledger Technologies (DLT) permissionada, modular, configurável e de código aberto, desenvolvida pela Linux Foundation. O Hyperledger implementa o conceito de contratos inteligentes, que são regras de negócio implementadas como transações na blockchain e chamadas por transações subsequentes, permitindo o desenvolvimento de contratos de negócio e aplicações descentralizadas. Tais características tornaram o Hyperledeger Fabric uma solução de destaque na implementação de soluções de blockchain privadas ou consorciadas, estando presente nos maiores provedores de computação em nuvem. Neste contexto, este minicurso tem como objetivo a apresentação teórica e prática do procedimento de criação, configuração e uso de uma rede blockchain baseada no modelo Hyperledger Fabric consorciado.

2.1. Conceitos básicos

A blockchain é um livro-razão seguro, compartilhado e distribuído que facilita o processo de gravação e rastreamento de recursos sem a necessidade de confiança em uma autoridade central, permitindo que duas partes comuniquem-se e troquem recursos em uma rede, na qual decisões são tomadas pela maioria, e não por uma única entidade [Salman et al. 2019]. Este livro-razão é composto de transações assinadas criptograficamente e agrupadas em blocos. Cada bloco está ligado criptograficamente com o bloco anterior através de uma validação utilizando um mecanismo de consenso. Conforme novos blocos

são adicionados à cadeia, torna-se mais difícil a alteração dos blocos antigos [Yaga et al. 2019].

Apesar do interesse crescente sobre o tema, a aplicação, implementação e operação das tecnologias relacionadas com blockchain ainda são consideradas tarefas complexas. Esta tecnologia foi apresentada inicialmente em 2008 com o Bitcoin, com o objetivo de permitir a realização de transações financeiras sem a necessidade de um intermediário confiável. Desde então, com o desenvolvimento de aplicações distribuídas e contratos inteligentes (Subseção 2.1.4), evoluiu rapidamente para soluções em diversas áreas. Neste sentido, torna-se necessário revisitar alguns conceitos básicos sobre blockchains, de modo a facilitar sua compreensão e aplicabilidade. De posse destes conhecimentos é possível a abordagem de aspectos práticos e operacionais relacionados à instalação, configuração e usabilidade de uma blockchain consorciada modelo Hyperledger Fabric.

2.1.1. Tipos de blockchain

A evolução da tecnologia blockchain pode ser dividida em três etapas: blockchain 1.0, blockchain 2.0 e blockchain 3.0. A blockchain 1.0 está relacionada com o Bitcoin e criptomoedas de modo geral. O Bitcoin é o primeiro e maior exemplo deste modelo de blockchain e representa, aproximadamente, 60 % do valor total de mercado de criptomoedas, composto por mais de 7 mil criptomoedas e avaliado em mais de 440 bilhões de dólares. Segundo [Swan 2015], enquanto o objetivo da blockchain 1.0 é a descentralização do dinheiro e meios de pagamento, a blockchain 2.0 busca a descentralização de mercados de modo geral, contemplando transações envolvendo diversos tipos de ativos utilizando a blockchain. Os contratos inteligentes são o principal destaque da blockchain 2.0 e consistem em contratos implementados via código na blockchain, que são executados automaticamente quando as condições pré-definidas são atendidas, sem a necessidade de um intermediário confiável. Já a blockchain 3.0 busca a aplicação da tecnologia em diferentes setores, de modo que seu potencial disruptivo implique em evoluções importantes na sociedade.

2.1.2. Permissionamento

Uma blockchain pode ser classificada como permissionada ou não permissionada. No modelo permissionado uma organização ou um consórcio de organizações são responsáveis pela validação e armazenamento das transações realizadas. Neste modelo, os participantes são previamente conhecidos e dependem de autorização para integrar a rede, gerar ou validar transações. De modo geral, as blockchains permissionadas são úteis para empresas, bancos e instituições que necessitam cumprir uma série de regulamentações e demandam controle completo de seus dados [Sharma 2020]. No modelo não permissionado, a validação e armazenamento das transações dependem do trabalho de um grupo de agentes anônimos, conhecidos como mineradores. A atuação dos mineradores é incentivada através da distribuição de *tokens*, como recompensa pelo trabalho realizado. Outra característica deste modelo é a transparência; os participantes devem ter acesso às informações referentes às transações processadas pela rede, incluindo os endereços e a composição dos blocos. Com base no modelo de permissionamento adotado, as blockchains são categorizadas como públicas, consorciadas ou privadas [Miers et al. 2019]:

- Em uma blockchain pública, todos podem ler, enviar ou validar transações, além de participar do processo de consenso distribuído, sendo considerada um modelo totalmente descentralizado.
- Uma blockchain consorciada é composta por duas ou mais instituições parceiras, que podem alterar as regras conforme suas necessidades. O consenso é obtido através de um processo realizado por um grupo específico de participantes sendo, desta forma, parcialmente descentralizado.
- Já uma blockchain privada é utilizada em modelo de organização única, que pode alterar o funcionamento conforme seus interesses e necessidades. É um modelo centralizado com processo de consenso simples de ser obtido.

Cada uma destas categorias possui uma aplicação distinta, baseada em um conjunto de características da blockchain. A Tabela 2.1 apresenta alguns critérios de comparação entre as principais características dos três tipos de blockchain [Miers et al. 2019].

Tabela 2.1: Comparação entre modelos de acesso da blockchain [Miers et al. 2019].

Características	blockchain Pública	blockchain Consórcio	blockchain Privada
Consenso distribuído	Todos os nós	Nós selecionados	Nós selecionados
Permissão de verificação	Pública	Restrita	Restrita
Imutabilidade	Sim	Adulterável	Adulterável
Centralização	Descentralizado	Parcial	Centralizado
Processo de consenso	Todos os nós	Nós selecionados	Nós selecionados

O consenso distribuído define se todos os nós podem participar do processo de consenso ou apenas nós pré-determinados. A permissão de verificação pode variar entre pública ou restrita, com a identidade dos participantes podendo ser anônima (em blockchains públicas) ou conhecida, nos modelos consórcio e privado [Tinu 2018]. Apesar de, por característica, os dados armazenados na blockchain serem imutáveis após a obtenção de consenso, algumas implementações dos modelos consórcio e privada podem permitir alterações. Já quanto ao grau de centralização, os modelos podem variar de centralizado no modelo de blockchain privada a descentralizado, no modelo público. Por fim, o processo de consenso define se qualquer entidade pode participar do processo ou apenas entidades pré-selecionadas [Miers et al. 2019].

2.1.3. Mecanismos de consenso

Devido à sua natureza assíncrona e descentralizada, um aspecto chave da tecnologia blockchain é determinar quem publicará o próximo bloco. Quando um nó ingressa em uma blockchain, este concorda com o estado inicial do sistema, definido no bloco Gênesis. Todo bloco seguinte deve ser válido e também possível de ser validado, de maneira independente, por qualquer nó da rede. Combinando o estado inicial com a capacidade de verificar cada bloco desde então, os usuários podem concordar sobre o estado atual da blockchain [Yaga et al. 2018]. Porém, cada nó pode ter uma visão diferente do estado da blockchain em um dado instante. Para garantir a convergência dessas visões, a blockchain

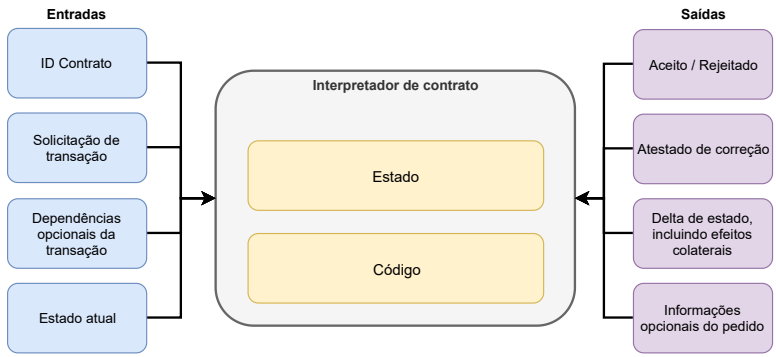
utiliza um mecanismo de consenso que permite que redes distribuídas ou descentralizadas tomem decisões unânimes quando necessário [Sankar et al. 2017]. Este mecanismo cria um sistema consistente, no qual todos os nós concordam com a ordem dos blocos e seus conteúdos.

De modo geral, o consenso pode ser obtido de diferentes maneiras, seja com o uso de algoritmos de loteria, como *Proof of Elapsed Time* (PoET) e *Proof of Work* (PoW), ou através do uso de métodos baseados em votação, tais como *Practical Byzantine Fault Tolerance* (PBFT) e RAFT [Hyperledger.Architecture 2017]. Dentre os mecanismos baseados em votação destacam-se os modelos *Crash Fault Tolerant* (CFT) e *Bizantine Fault Tolerant* (BFT). Enquanto o mecanismo CFT assume que alguns nós responsáveis pelo consenso podem falhar, o mecanismo BFT admite que além de falhar, alguns nós podem agir de maneira maliciosa. Cada um destes mecanismos possui benefícios e limitações. Desta forma, a escolha adequada à solução deve estar ligada ao cenário ao qual esta solução é implementada.

2.1.4. Contratos Inteligentes

Um contrato inteligente é uma regra de negócio contendo a lógica a ser executada em uma blockchain, podendo ser uma simples atualização de dados ou uma complexa execução de um contrato com condições anexas. Existem dois tipos diferentes de contratos inteligentes: os contratos inteligentes instalados e os contratos inteligentes na cadeia. Os contratos inteligentes instalados tem as regras de negócio instaladas nos validadores da rede antes que esta seja lançada. Já os contratos inteligentes na cadeia tem as regras de negócio implementadas como transações na blockchain, e são chamadas por transações subsequentes. Neste modelo, o código que define as regras de negócio torna-se parte da blockchain [Hyperledger.Smartcontracts 2017]. O modelo proposto na Figura 2.1 representa como uma solicitação enviada à um contrato inteligente é processada.

Figura 2.1: Processamento de contrato inteligente - Fonte: Adaptado de [Hyperledger.Smartcontracts 2017].



As saídas adequadas são geradas se a solicitação é válida e aceita. Estas saídas incluem o novo estado e possíveis efeitos colaterais desta alteração. Quando o processamento está concluído, o interpretador empacota o novo estado, um atestado de correção e qualquer outra informação adicional, e o envia para o serviço de consenso. A camada

de contrato inteligente valida cada solicitação, garantindo que está de acordo com as políticas da transação. Solicitações inválidas são rejeitadas e descartadas, sem inclusão na blockchain.

2.2. Introdução ao Hyperledger

O Projeto Hyperledger é uma iniciativa da Linux Foundation para desenvolver um ecossistema de código aberto de desenvolvimento de blockchain. A Linux Foundation visa criar um ambiente no qual comunidades de desenvolvedores de *software* e empresas se reúnam e se coordenem para construir estruturas de blockchain. O modelo Hyperledger é um projeto multiplataforma, modular e de código aberto. Esta abordagem apresenta, como diferenciais, características como extensibilidade, flexibilidade e a capacidade de modificar qualquer componente sem afetar o sistema [Hyperledger.Architecture 2017]. O Hyperledger é um projeto guarda-chuvas, sob o qual estão em desenvolvimento cinco *frameworks* distintos [Dhillon et al. 2017]:

- **Sawtooth**: Desenvolvido pela Intel com o objetivo de criar uma blockchain de código aberto, o Hyperledger Sawtooth é uma plataforma modular para desenvolver, implementar e executar *Distributed Ledger Technologies* (DLT). Adota um mecanismo de consenso conhecido como PoET, que tem como objetivo a obtenção de consenso com mínimo esforço computacional. É um *framework* que permite a implementação de soluções permissionadas e não permissionadas.

- **Fabric**: Este *framework* tem como objetivo permitir o desenvolvimento de aplicações empresariais com arquitetura modular, possibilitando a adoção de diferentes mecanismos de consenso e serviços únicos de filiação.

- **Iroha**: Conjunto de bibliotecas e componentes que permite a implementação de DLT em infraestruturas já existentes.

- **Burrow**: Blockchain permissionada que executa contratos inteligentes de maneira similar à *Ethereum Virtual Machine* (EVM). Permite a execução de contratos inteligentes em múltiplas blockchains compatíveis entre si porém em execução em domínios distintos.

- **Indy**: Trata-se de um *Software Development Kit* (SDK) para o Hyperledger que oferece componentes que permitem adicionar novos recursos e funcionalidades para gestão de identidades de maneira descentralizada.

Desta forma, existem projetos distintos de blockchain Hyperledger, cada qual possuindo características para um modelo específico de solução. Contudo, todos os projetos Hyperledger seguem um modelo de desenvolvimento que inclui uma abordagem modular e extensiva, interoperabilidade e ênfase em soluções seguras, com uma abordagem independente de *token* e sem criptomoeda nativa, além do uso de *Application Programming Interface* (API).

2.2.1. Hyperledger Fabric

O Hyperledger Fabric é uma arquitetura modular de blockchain que permite a conexão de componentes como mecanismo de consenso, serviços de filiação e funções de transação. Esta característica permite uma customização efetiva da plataforma, conforme as necessidades do ambiente. Assim, quando implementada em um ambiente composto por apenas uma empresa, por exemplo, o uso de um mecanismo de consenso CFT pode ser mais adequado que um mecanismo BFT. Por outro lado, em ambientes descentralizados multiorganizacionais, um mecanismo BFT pode ser necessário [Hyperledger 2020a]. Além do mecanismo de consenso, outros componentes do Hyperledger Fabric são modulares e configuráveis:

- *Ordering service*: Responsável por estabelecer o consenso quanto à ordem das transações e enviá-las aos *peers*.
- *Membership service provider*: Responsável pela associação de entidades que compõe a rede à identidades criptográficas.
- *Peer-to-peer gossip service*: Responsável por disseminar os blocos para todos os nós.
- *Chaincode*: No Hyperledger Fabric os contratos inteligentes são conhecidos também como *chaincode*, e podem ser desenvolvidos em Go, Javascript e, eventualmente, outra linguagem como Java. Neste modelo, existem dois tipos de *chaincode*: *system chaincode* e *application chaincode*. O *system chaincode* normalmente lida com transações relativas ao sistema, tais como gerenciamento do ciclo de vida e configurações de políticas. Já o *application chaincode* gerencia o estado do *ledger*, incluindo registros de dados.

Adicionalmente, a plataforma também dá suporte a uma variedade de sistemas de gerenciamento de banco de dados, e permite a configuração das políticas de endosso e validação exigidas. No Hyperledger Fabric, a blockchain é composta por nós pertencentes às organizações que compõe o consórcio, além de um ou mais nós do tipo *orderer*, responsáveis por ordenar as transações. Para poderem se comunicar, estes nós devem participar de um mesmo canal.

2.2.2. Canais e seus componentes

O modelo proposto pelo Hyperledger Fabric permite que as organizações participem de múltiplas redes blockchain independentes, através dos canais. Toda transação deve ser executada em um canal, no qual todos os participantes devem ser autenticados e autorizados a realizar transações. O canal oferece o compartilhamento de uma infraestrutura, mantendo a privacidade dos dados e da comunicação, e é composto pelos nós membros (*peer*) pertencentes às organizações participantes, nós âncora (*anchor peer)*, nós de ordenamento das transações (*order peer)*, *ledger* e *chaincode*.

Todas as organizações que compõe o canal devem possuir ao menos um *anchor peer*. Os *anchor peer* são responsáveis por comunicar-se com os *orderer peer*, obter os

blocos contendo as transações, e disseminá-los para os demais nós de suas organizações. Apesar de poderem pertencer a múltiplos canais, e possuir diversos *ledger*, os dados dos canais são privados, e não podem trafegar entre eles. Já os *orderer peer* são responsáveis por ordenar as transações através de um mecanismo de consenso, montar os blocos e entregá-los aos *anchor peer* [Hyperledger 2020a].

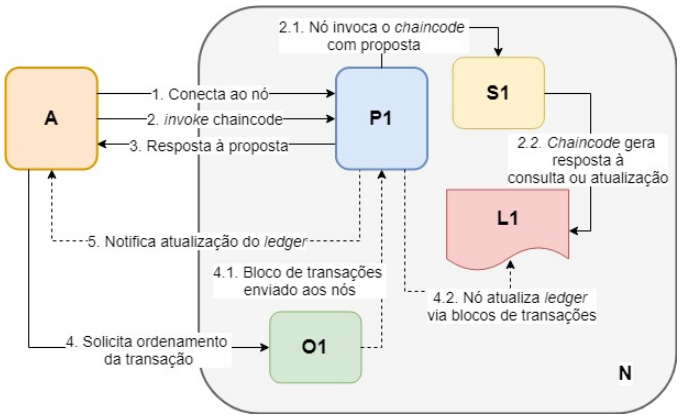
2.2.3. Arquitetura de transações

O Hyperledger Fabric implementa um novo modelo de arquitetura de transações composto de três passos e conhecido como *execute-order-validate*. O primeiro passo executa a transação, verifica sua exatidão e a endossa. O segundo passo ordena as transações utilizando um mecanismo de consenso. O terceiro passo valida as transações por meio de uma política de validação específica da aplicação, antes de adicioná-la ao *ledger*. A política de validação especifica quais e quantos nós devem validar a execução do *chaincode*.

Esta arquitetura agrega características para lidar com desafios referentes à escalabilidade, desempenho e confidencialidade existentes no modelo tradicional *order-validate*. Pode-se imaginar, como exemplo, um *chaincode* cuja execução seja um *loop* infinito. Em uma arquitetura *order-execute*, este *chaincode* teria uma consequência crítica. Já o modelo *execute-order-validate* é capaz de identificar este problema na primeira etapa de execução, quando os nós devem endossar as transações que serão submetidas à validação. Neste modelo, a transação não seria endossada e seria descartada antes de ser repassada para os nós *orderer* e de endosso.

Para realizar a execução e interação com os *chaincodes*, o Hyperledger Fabric faz uso das transações. Conforme apresenta [Dhillon et al. 2017], existem dois tipos de transações: *Invoke* e *Query*. A transação do tipo *invoke* executa um *chaincode* na blockchain, enquanto a transação *query* executa uma consulta. Através de uma transação *invoke* é possível a um cliente executar uma função específica contida em um *chaincode*. A Figura 2.2 ilustra como aplicações interagem com os nós para acessar o *ledger*.

Figura 2.2: Fluxo de transações no Hyperledger Fabric. Adaptado de [Hyperledger 2020a].



Neste exemplo, a aplicação A conecta ao nó P1 e realiza um *invoke* do *chaincode* S1, para realizar uma atualização do *ledger* L1. O nó P1 invoca S1, solicitando uma

resposta contendo o resultado da solicitação de atualização e retorna a resposta para A. A aplicação A recebe a resposta, monta uma transação contendo todas as respostas e envia ao ordenador O1. Caso a solicitação fosse apenas uma consulta ao *ledger*, o processo estaria completo. Em seguida, o ordenador coleta as transações e as agrupa em blocos, distribuindo-os a todos os nós da rede. P1 valida a transação antes de adicioná-la ao *ledger*. Por fim, P1 gera um evento e envia à aplicação.

2.2.4. Binários do Hyperledger Fabric

O Hyperledger Fabric dispõe de um conjunto de binários responsáveis pelas principais funcionalidades da plataforma:

- *Configtxgen*: Responsável pela geração dos artefatos de rede, tais como o genesis.block e channel.tx;
- *Configtxlator*: Responsável por gerar e configurar o canal de comunicação;
- *Cryptogen*: Responsável pela geração do material criptográfico;
- *Discovery*: Cliente de linha de comando para o serviço de descoberta;
- *Idemixgen*: Utilitário para geração de chaves a serem usadas com o *Membership Service Provider* (MSP);
- *Oderer*: Responsável pela interação com o nó de ordenamento;
- *Peer*: Responsável pela interação com o nó de validação; e
- *Fabric-ca-client*: Cliente para criação e registro de usuários.

2.3. Instalação e configuração do Hyperledger Fabric

Conforme apresentado na documentação oficial da plataforma, para a correta instalação e utilização do Hyperledger Fabric é necessária a instalação de um conjunto de pré-requisitos. Estes pré-requisitos, listados na Subseção 2.3.1, permitem a implementação dos nós que compõe a blockchain, geração das chaves criptográficas, instalação do *chaincode*, dentre outras ações necessárias.

Para apresentar as funcionalidades disponíveis no Hyperledger Fabric e como operá-las, este trabalho utiliza um repositório contendo um modelo de rede, os *scripts* e arquivos de configurações necessários para a implementação proposta. Este repositório é baseado no modelo desenvolvido por [Padvan 2020] que, por sua vez, tem como referência os exemplos disponibilizados pelo Hyperledger, na coletânea *fabric-samples*.

A instalação do Hyperledger Fabric pode ser realizada em ambientes GNU/Linux, MS-Windows e Apple MacOS, sendo que cada uma das instalações tem suas particularidades. Este trabalho adota como plataforma base de desenvolvimento o sistema operacional GNU/Linux, especificamente a distribuição Ubuntu Desktop 16.04.

2.3.1. Instalação dos pré-requisitos

A correta execução do Hyperledger Fabric depende da instalação de uma série de pré-requisitos. Com o objetivo de facilitar a instalação destas dependências, foi elaborado o *script preinstall.sh*, disponibilizado na raíz do repositório clonado. Para que seja possível clonar o repositório é necessário, primeiramente, realizar a instalação do Git, através do comando *sudo apt install git*. Em seguida, deve-se clonar o repositório, de preferência no diretório raiz, através do comando *git clone https://github.com/marco-developer/errc2020-hyperledgerfabric.git*. Por fim, o *script* deve ser executado como root, e instalará os seguintes componentes:

- cUrl: Ferramenta que permite a transferência de dados via linha de comando, utilizada para realizar o *download* de aplicações necessárias;
- Docker: Plataforma de código aberto que permite o desenvolvimento, implementação e execução de aplicações em contêineres. Utilizado para a criação e execução da rede e dos nós que compõe o Hyperledger Fabric. Versão utilizada: Última *stable*;
- Docker Compose: Ferramenta que auxilia a definição e execução de aplicações multi contêineres através de um arquivo YAML. Utilizada para facilitar a configuração e implementação dos nós que compõe a rede Hyperledger. Versão utilizada: 1.27.3;
- Binários do Hyperledger Fabric: Necessários para criação, configuração e interação com a rede Hyperledger;
- Imagens Docker: Imagens utilizadas para criação dos contêineres que compõe a plataforma;
- Coletânea de exemplos do Hyperledger Fabric: Exemplos disponibilizados pelos desenvolvedores que auxiliam na compreensão do funcionamento da plataforma;
- Node: Permite a execução de código Javascript utilizado na implementação e execução da plataforma. Versão utilizada: V12.19;
- Go: Linguagem de programação utilizada no desenvolvimento do *chaincode*. Versão utilizada: 1.15.3;
- NPM: Gerenciador de pacotes Javascript, utilizado na disponibilização da API para comunicação com a blockchain; e
- SSH: Protocolo de rede seguro, utilizado para acesso ao *host* de desenvolvimento.

Após a instalação das dependências acima listadas, o *script* adicionará à variável PATH do *bash* os caminhos referentes aos componentes, de modo que possam ser executados em qualquer diretório. Finalizada a execução do *script* sem erros, será possível iniciar a configuração do modelo de implementação proposto por este trabalho.

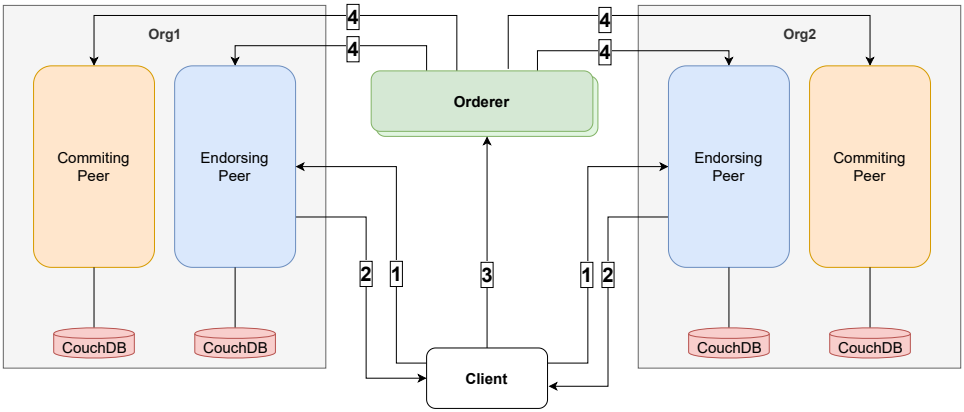
2.3.2. Proposta de implementação

Conforme citado na Seção 2.3, para explorar e apresentar as diversas opções de configuração da rede e seus respectivos arquivos, será utilizado neste minicurso um repositório contendo um modelo básico de rede.

Este repositório agrupa os arquivos de configuração necessários para a implementação e interação com a blockchain, detalhados na Subseção 2.3.3. A implementação proposta neste modelo consiste em uma blockchain composta por duas organizações, que possuem dois nós (*peer*), sendo um deles de endosso (*endorsing peer*) e uma *Certified Authority* (CA) cada. O serviço de ordenação das transações, por sua vez, é composto por três nós, utilizando o mecanismo de consenso RAFT.

O RAFT é um mecanismo que busca o consenso através da eleição de um líder, responsável pela gestão do consenso envolvendo as transações. Este líder recebe as transações e tem como tarefas replicá-las para os outros membros, que devem validá-las, além de informá-los quando devem aplicar as alterações propostas [Ongaro and Ousterhout 2014]. Os nós CouchDB são responsáveis pelo armazenamento do *ledger* contendo as transações validadas. É importante notar que todo nó *peer* demanda um nó CouchDB, que será responsável pelo armazenamento do *ledger*. A Figura 2.3 representa um exemplo de transação no modelo proposto.

Figura 2.3: Modelo de implementação.



Neste modelo (Figura 2.3) está representada uma transação do tipo *invoke*, na qual o cliente envia a proposta aos *endorser peer*(1), que executam e endossam a transação, retornando-a ao cliente(2). Em seguida, o cliente envia a proposta endossada ao *orderer*(3), que a ordena e reencaminha para todos os nós de validação(4). Estes nós validarão a transação e armazenarão o novo estado do *ledger* no CouchDB. Para a implementação do modelo proposto, é necessária a execução de uma série de etapas:

- Edição do arquivo *criptoconfig.yaml*, contendo as configurações referentes aos nós *orderer* e *peer*;
- Geração do material criptográfico relativo aos nós definidos em *criptoconfig.yaml*;

- Edição do arquivo *configtx.yaml*, que contém configurações referentes às organizações, nós *orderer*, canal e políticas de interação;
- Geração do bloco gênesis, da transação de configuração do canal e definição dos *anchor peer*;
- Edição do arquivo *docker-compose.yaml*, contendo as informações necessárias para a criação dos contêineres e operacionalização da rede Hyperledger Fabric;
- Criação do canal e conexão dos nós que o compõe;
- Instalação e inicialização do *chaincode*.

A Subseção 2.3.3 apresenta as principais funções dos arquivos de configuração *criptoconfig.yaml* e *configtx.yaml*, bem como a geração do material criptográfico, geração do bloco gênesis, criação e execução dos nós que compõe a rede. Já a Seção 2.4 apresenta o processo de criação do canal, conexão dos nós membros, instalação e inicialização do *chaincode*.

2.3.3. Configuração

O repositório utilizado para armazenar o exemplo em questão contempla os arquivos de configuração que definem a composição das organizações, as informações necessárias para a criação dos contêineres contendo os nós da rede, bem como a API utilizada para interação com a blockchain. Desta forma, é necessário compreender o conteúdo de cada um dos arquivos de configuração para que seja possível a implementação do modelo proposto.

Cryptoconfig.yaml

O arquivo de configuração *crytoconfig.yaml* define características dos nós *orderer* e *peer*, e é utilizado como entrada pelo binário *cryptogen* para a criação do material criptográfico. O Código 1 mostra a configuração do modelo de implementação proposto.

Código 1: criptoconfig.yaml - Definição dos nós

```
1  OrdererOrgs:
2      - Name: Orderer
3      Domain: example.com
4      EnableNodeOUs: true
5      - Specs:
6          - Hostname: orderer
7          SANS:
8              - "localhost"
9              - "127.0.0.1"
10         - Hostname: orderer2
11         SANS:
12             - "localhost"
13             - "127.0.0.1"
14         - Hostname: orderer3
15         SANS:
16             - "localhost"
17             - "127.0.0.1"
18 PeerOrgs:
19     - Name: Org1
20     Domain: org1.example.com
21     EnableNodeOUs: true
22     - Template:
23         Count: 2
24         SANS:
25             - "localhost"
26     - Users
27         Count: 1
28     - Name: Org2
29     Domain: org2.example.com
30     EnableNodeOUs: true
31     - Template:
32         Count: 2
33         SANS:
34             - "localhost"
35     - Users
36         Count: 1
```

Inicialmente, estão definidas as características genéricas dos nós *orderer*, como nome e o domínio, bem como as características específicas de cada nó, como *hostname* e endereço *Internet Protocol* (IP) (linhas 1 a 17). Também estão definidas as configurações das organizações, com seus respectivos nomes e domínio, dois nós *peer* (linhas 23 e 32) e um usuário (linhas 27 e 36). Após as definições destas configurações é possível prosseguir para a geração do material criptográfico referente aos nós, por meio da execução do comando:

cryptogen generate –config=./crypto-config.yaml –output=./crypto-config/.

O material criptográfico gerado com a execução deste comando será armazenado no caminho definido em "*–output*", no caso em *./crypto-config/*.

Configtx.yaml

O arquivo *configtx.yaml* inclui as configurações do canal de comunicação. Esta configuração é armazenada no *ledger*, e especifica quais organizações são membros do canal, quem são os nós responsáveis pelo ordenamento das transações e geração dos blocos, bem como a política de atualização do canal [Hyperledger 2020a]. As configurações definidas neste arquivo são utilizadas pelo binário *configtxgen* para geração do bloco gênesis. Este arquivo é dividido nos seguintes subgrupos de configuração:

- *Organizations*: Define quais organizações são membros do canal. Cada organização é identificada através de um MSP ID e um MSP de canal, que define direitos administrativos e de participação. O MSP de canal é armazenado na configuração do canal e contém os certificados utilizados para identificar os nós, aplicações e administradores da organização. O modelo de implementação proposto é composto de três organizações: Org1, Org2 e OrdererOrg, responsável pela administração do serviço de ordenamento. O serviço de ordenamento é implementado como organização pois as melhores práticas recomendam que os certificados de nós *peer* sejam emitidos por CA diferente dos certificados de nós *orderer*.

- *Capabilities*: Esta seção permite que organizações que executam versões diferentes dos binários do Hyperledger Fabric participem do mesmo canal. Existem três subgrupos distintos: *Application capabilities*, que define quais os recursos utilizados e a versão mínima do binário *peer*, *orderer capabilities*, que define os recursos utilizados pelos nós de ordenamento, tais como mecanismo de consenso, e a versão mínima do binário *orderer* e, por fim,*channel capabilities*, que define a versão mínima do Hyperledger Fabric permitida.

- *Application*: Define as políticas que controlam a interação entre organizações através do canal. Essas políticas controlam o número de nós de organizações necessários para aprovar uma definição de *chaincode* ou atualização da configuração do canal, além de definir as permissões para escrita ou consulta ao *ledger*.

- *Orderer*: Os nós de ordenamento definidos na configuração devem ser incluídos no *consenter set*, que é o grupo de nós habilitados a criar novos blocos e distribuí-los aos nós *peer* que compõe o canal. O Código 2 lista características de um nó *orderer*.

O campo *OrdererType* define o mecanismo de consenso adotado, no caso, o RAFT. O campo *Consenters* define a lista de nós *orderer* que podem participar do processo de consenso. Neste exemplo há apenas um nó *orderer*, definido como orderer.example.com, e utilizando a porta 7050. Já os campos *ClientTLSCert* e *ServerTLSCert* definem o caminho dos certificados TLS utilizados pelo nó. Por fim, o campo *Addresses* define o endereço de acesso ao nó.

Além destas configurações, esta seção traz também duas configurações importantes, referentes à configuração dos blocos: *BatchTimeout* e *BatchSize*. O campo *BatchTimeout* define com que frequência um bloco é criado. Já o campo *BatchSize* define o tamanho máximo do bloco, em quantidade de transações e em *bytes*.

Código 2: configtx.yaml - Definição de nó *orderer*

```
1  OrdererType: etcdraft
2
3  EtcdRaft:
4          Consenters:
5          - Host: orderer.example.com
6          Port: 7050
7          ClientTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt
8          ServerTLSCert: crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt
9  Addresses:
10         - orderer.example.com:7050
11
12 BatchTimeout: 2s
13 BatchSize:
14         MaxMessageCount: 10
15         AbsoluteMaxBytes: 99 MB
16         PreferredMaxBytes: 512 KB
```

- *Channel*: Define as políticas de mais alto nível da configuração, tais como algoritmo de *hash* e estrutura dos dados utilizada na criação de novos blocos. Para a maioria das implementações, as configurações padrão definidas nesta seção não requerem modificações.
- *Profiles*: Auxilia a ferramenta *configtxgen* a construir a configuração do canal. Esta reúne informações das seções anteriores utilizadas para criar a transação de criação do canal e para escrever o bloco gênesis.

Com o arquivo *configtx.yaml* adequadamente configurado é possível realizar a criação do bloco gênesis, da transação contendo as configurações do canal e definição dos nós âncora das organizações participantes. Para isso, são executados os comandos listados no Código 3, extraídos do *script create-artifacts.sh*.

Docker-compose.yaml

O arquivo *docker-compose.yaml* contém as configurações aplicadas a todos os contêineres que compõe a rede Hyperledger Fabric, e é utilizado como entrada para o docker-compose, durante o processo de criação e execução da rede. Para o modelo proposto, é necessário acessar o subdiretório */artifacts/* e executar o comando *docker-compose up -d*, que irá criar os contêineres referentes aos seguintes nós:

Código 3: create-artifacts.sh - Bloco gênesis e configuração do canal

```
1  # Define os nomes dos canais de sistema e comunicacao
2  SYS_CHANNEL="sys-channel"
3  CHANNEL_NAME=<nome do canal>
4
5  # Gera bloco genesis
6  configtxgen -profile OrdererGenesis -configPath . -channelID $SYS_CHANNEL -outputBlock ./genesis.block
7
8  # Gera transacao de configuracao do canal
9  configtxgen -profile BasicChannel -configPath . -outputCreateChannelTx ./<nome do canal>.tx -channelID
     $CHANNEL_NAME
10
11 # Define os nos ancora de cada organizacao
12 configtxgen -profile BasicChannel -configPath . -outputAnchorPeersUpdate ./Org1MSPanchors.tx -channelID
     $CHANNEL_NAME -asOrg Org1MSP
13
14 configtxgen -profile BasicChannel -configPath . -outputAnchorPeersUpdate ./Org2MSPanchors.tx -channelID
     $CHANNEL_NAME -asOrg Org2MSP
```

- *Orderer*:
  - orderer.example.com
  - orderer2.example.com
  - orderer3.example.com
- Org1:
  - ca.org1.example.com
  - peer0.org1.example.com
  - peer1.org1.example.com
- Org2:
  - ca.org2.example.com
  - peer0.org2.example.com
  - peer1.org2.example.com
- Armazenamento de estado:
  - couchdb0.example.com
  - couchdb1.example.com
  - couchdb2.example.com
  - couchdb3.example.com

As configurações dos nós estabelecidas no *docker-compose.yaml* são divididas em quatro subgrupos, conforme o tipo de nó: CA, *peer*, *orderer* e CouchDB. Em cada subgrupo é definida a imagem a ser utilizada para montar o contêiner, o nome do *host*, as variáveis de ambiente do contêiner, os volumes mapeados, as portas a serem expostas, o comando a ser executado após a inicialização, as dependências e outras configurações necessárias. Neste arquivo, deve ser dada especial atenção às configurações das variáveis de ambiente e ao mapeamento dos volumes, pois envolvem caminhos de acesso à certificados e chaves, necessários para o correto funcionamento dos nós e, consequentemente, do ambiente.

2.4. Criação do canal e instalação do chaincode

Nas seções anteriores foram instalados os pré-requisitos, criados os materiais criptográficos referentes aos nós que compõe as organizações envolvidas, gerada a transação contendo as configurações do canal de comunicação, o bloco gênesis e criados e executados os contêineres que compõe a rede blockchain. Desta forma, com a rede blockchain operacional, é possível prosseguir com a criação do canal e posterior implementação do *chaincode*. Os comandos executados nesta seção dependem da definição prévia de um conjunto de variáveis globais. O Código 4 faz parte de *script createChannel.sh*, e apresenta as funções que definem as variáveis globais usadas durante a criação do canal e implementação do *chaincode*.

Código 4: createChannel.sh - Variáveis globais

```
# Define o caminho dos certificados das CAs das organizacoes:
export PEER0_ORG1_CA=<Caminho do cert. CA Org1 >
export PEER0_ORG2_CA=<Caminho do cert. CA Org2 >
export ORDERER_CA=<Caminho do cert. CA Orderer >

# Define as config. do MSP, caminho cert. TLS,
caminho config. do MSP e endereco de cada no:
setGlobalsPeerFor0Org1(){
        export CORE_PEER_LOCALMSPID="Org1MSP"
        export CORE_PEER_TLS_ROOTCERT_FILE=
        $PEER0_ORG1_CA
        export CORE_PEER_MSPCONFIGPATH=
        <Caminho do MSP>
        export CORE_PEER_ADDRESS=localhost:7051
}
setGlobalsPeer1ForOrg1(){
        export CORE_PEER_LOCALMSPID="Org1MSP"
        export CORE_PEER_TLS_ROOTCERT_FILE=
        $PEER0_ORG1_CA
        export CORE_PEER_MSPCONFIGPATH=
        <Caminho do MSP>
        export CORE_PEER_ADDRESS=localhost:8051
}
setGlobalsForPeer0Org2(){
        export CORE_PEER_LOCALMSPID="Org2MSP"
        export CORE_PEER_TLS_ROOTCERT_FILE=
        $PEER0_ORG2_CA
        export CORE_PEER_MSPCONFIGPATH=
        <Caminho do MSP>
        export CORE_PEER_ADDRESS=localhost:9051
}
setGlobalsForPeer1Org2(){
        export CORE_PEER_LOCALMSPID="Org2MSP"
        export CORE_PEER_TLS_ROOTCERT_FILE=
        $PEER0_ORG2_CA
        export CORE_PEER_MSPCONFIGPATH=
        <Caminho do MSP>
        export CORE_PEER_ADDRESS=localhost:10051
}
```

2.4.1. Criação do canal

Após a criação dos blocos de configuração, através do comando *configtxgen*, é possível proceder com a criação do canal de comunicação. Este processo envolve três etapas: criação do canal, conexão dos nós ao canal e definição dos *anchor peer* das organizações. O Código 5 apresenta a criação do canal, utilizando como entrada o bloco gerado anteriormente:

Código 5: createChannel.sh - Criação do canal

```
# Define variaveis globais
setGlobalsForPeer0Org1

# Cria canal
peer channel create
        -o localhost:7050
        -c "$CHANNEL_NAME"
        –ordererTLSHostnameOverride orderer.example.com
        -f ./artifacts/channel/$CHANNEL_NAME.tx
        –outputBlock ./channel-artifacts/$CHANNEL_NAME.block
        –tls true
        –cafile $ORDERER_CA
```

Este comando realiza a criação do canal por meio do nó *orderer* (*-o localhost:7050*), utilizando o arquivo de transação contendo as configurações do canal (*-f ./artifacts/channel/$CHANNEL_NAME.tx*) gerado pelo binário *configtxgen*. A saída deste comando será armazenada no bloco gênesis, por meio do argumento *–outputBlock <caminho>\ $CHANNEL_NAME.block*. A execução deste comando com sucesso apresentará no terminal uma saída indicando que o bloco zero (bloco gênesis) foi adicionado à blockchain.

2.4.2. Conectando os nós ao canal

Com o bloco gênesis adicionado, é possível realizar a conexão dos nós das organizações que irão integrar o canal. Para isso, é necessário executar o comando *peer channel join* em todos os nós das organizações que compõe o consórcio. O Código 6 apresenta o trecho do *script* que realiza a conexão dos nós ao canal.

Código 6: createChannel.sh - *join channel*

```
1  joinChannel(){
2  setGlobalsForPeer0Org1
3  peer channel join -b
     ./channel-artifacts/$CHANNEL_NAME.block
4
5  setGlobalsForPeer1Org1
6  peer channel join -b
     ./channel-artifacts/$CHANNEL_NAME.block
12 }
```

```
7  setGlobalsForPeer0Org2
8  peer channel join -b
     ./channel-artifacts/$CHANNEL_NAME.block
9
10 setGlobalsForPeer1Org2
11 peer channel join -b
     ./channel-artifacts/$CHANNEL_NAME.block
```

A execução deste comando com sucesso apresentará no terminal uma saída informando que a proposta de conexão do nó ao canal foi enviada com sucesso. Para realizar a conexão dos demais nós, basta definir as variáveis globais dos demais nós e repetir o procedimento.

2.4.3. Definição dos anchor peer

Este tipo de nó é um *peer* que comunica-se e pode ser descoberto por todos os outros nós da rede. Todas as organizações que compõe o consórcio devem possuir um ou mais *anchor peers*. Para o modelo proposto, vamos definir o nó *peer0* das organizações como *anchor peer*. O Código 7 traz a definição dos *anchor peers* de Org1 e Org2.

Código 7: createChannel.sh - Define *anchor peer*

```
1  # Define variaveis globais para Org1
2  setGlobalsForPeer0Org1
3
4  # Define anchor peer para Org1
5  peer channel update
6        -o localhost:7050
7        –ordererTLSHostnameOverride orderer.example.com
8        -c "$CHANNEL_NAME"
9        -f ./artifacts/chan-
           nel/${CORE_PEER_LOCALMSPID}anchors.tx
10       –tls true
11       –cafile $ORDERER_CA
12
```

```
13 # Define variaveis globais para Org2
14 setGlobalsPeer0Org2
15
16 # Define anchor peer para Org2
17 peer channel update
18       -o localhost:7050
19       –ordererTLSHostnameOverride orderer.example.com
20       -c "$CHANNEL_NAME"
21       -f ./artifacts/chan-
           nel/${CORE_PEER_LOCALMSPID}anchors.tx
22       –tls true
23       –cafile $ORDERER_CA
```

Com a execução destes comandos os nós *peer0.org1.example.com* e

peer0.org2.example.com serão definidos como os *anchor peers* das organizações, estando acessíveis a todos os participantes da rede.

2.4.4. Instalação do *chaincode*

Após a criação do canal e conexão dos nós participantes, é necessário realizar a instalação do *chaincode* naqueles nós que terão permissão para alterar valores no *ledger*. Seguindo o modelo proposto, cada organização terá um nó com o *chaincode* instalado, que será o *peer0*. O processo de instalação do *chaincode* é realizado pelo *script deploychaincode.sh*, e é composto pelas seguintes etapas:

- *Geração do pacote contendo chaincode*: O primeiro passo antes da instalação do *chaincode* é gerar seu pacote. Nesta etapa devem ser informados o caminho do *chaincode*, a linguagem utilizada e um *label* para identificação. O Código 8 apresenta o processo de geração do pacote contendo o *chaincode*.

Código 8: deployChaincode.sh - Empacota *chaincode*

```
1  # Define variaveis globais
2  setGlobalsForPeer0Org1
3
4  CHANNEL_NAME="$CHANNEL_NAME"
5  CC_RUNTIME_LANGUAGE="golang"
6  VERSION="1"
7  CC_SR_PATH="./artifacts/src/github.com/fabcar/go"
8  CC_NAME="fabcar"
9
10 # Gera pacote contendo chaincode
11 peer lifecycle chaincode package $CC_NAME.tar.gz –path $CC_SRC_PATH –lang $CC_RUNTIME_LANGUAGE
     –label $CC_NAME_$VERSION
```

  Para geração do pacote, são definidas as variáveis globais do *peer0.org1*, o nome do canal de comunicação, nome e versão do *chaincode* e linguagem utilizada em seu desenvolvimento.

- *Instalação do Chaincode*: Após a geração do pacote contendo o *chaincode*, ele está apto a ser instalado nos devidos nós. O processo de instalação do *chaincode* é simples, conforme Código 9, extraído do *script* de instalação:

Código 9: deployChaincode.sh - Instalação do *chaincode*

```
1  # Define variaveis globais para Peer0 Org1
2  setGlobalsPeer0Org1
3
4  # Instala chaincode em Peer0 Org1
5  peer lifecycle chaincode install $CC_NAME.tar.gz
6
7  # Define variaveis globais para Peer0 Org2
8  setGlobalsPeer0Org2
9
10 # Instala chaincode em Peer0 Org1
11 peer lifecycle chaincode install $CC_NAME.tar.gz
```

- *Aprovação do chaincode*: Após a instalação do *chaincode*, o próximo passo deve ser a aprovação por parte das organizações que compõe o canal. O comando para

aprovação do *chaincode* demanda diversas informações, dentre elas o ID do *chaincode*. Desta forma, antes de emitir o comando para aprovação do *chaincode*, será executado o comando *queryinstalled*, para obter o ID, direcionando sua saída para a variável *PACKAGE_ID*, que será utilizada no comando de aprovação. Após a aprovação, é executado o comando *checkcommitreadiness*, que verifica o resultado da aprovação. O Código 10 exemplifica estes passos.

Código 10: deployChaincode.sh - Aprovação do *chaincode*

```
# Obtem ID do chaincode instalado
queryInstalled() {
setGlobalsPeer0Org1
peer lifecycle chaincode queryinstalled >&log.txt
cat log.txt
PACKAGE_ID=$(sed -n "/${CC_NAME}_${VERSION}/{s/Package ID: //; s/, Label:.*$//; p;}"log.txt)
}

# Aprova chaincode na Org1
approveForMyOrg1() {
peer lifecycle chaincode approveformyorg -o localhost:7050
        –ordererTLSHostnameOverride orderer.example.com –tls
        –collections-config $PRIVATE_DATA_CONFIG
        –cafile $ORDERER_CA –channelID $CHANNEL_NAME –name
        ${CC_NAME} –version ${VERSION}
        –init-required –package-id ${PACKAGE_ID}
        –sequence ${VERSION}
}

# Verifica aprovacao do chaincode
checkCommitReadyness() {
    peer lifecycle chaincode checkcommitreadiness –collections-config $PRIVATE_DATA_CONFIG
        –channelID $CHANNEL_NAME –name ${CC_NAME} –version ${VERSION}
        –sequence ${VERSION} –output json –init-required
}
```

A primeira função obterá o ID do pacote, a segunda função utilizará este ID para aprovar o *chaincode* instalado no nó *peer0.org1*, e a terceira função irá verificar se o *chaincode* foi instalado corretamente, retornando o resultado desta consulta em formato *JavaScript Object Notation* (JSON). Após a aprovação para a organização 1, o mesmo procedimento deve ser realizado para a organização 2, devendo alterar apenas a definição das variáveis globais de *SetGlobalsForPeer0Org1* para *SetGlobalsForPeer0Org2*.

- *Registro do chaincode*: Uma vez que o *chaincode* foi aprovado por um número suficiente de organizações (por padrão a maioria delas), ele pode ser registrado através do comando *commit*, conforme apresentado no Código 11.

  A execução da função *commitChaincodeDefinition* irá realizar o *commit* do *chaincode* nos nós *peer0* das organizações que compõe o canal. Após sua execução, a função *queryCommited* irá verificar quais *chaincodes* estão instalados no canal especificado. Desta forma, a correta instalação e registro do *chaincode* deve apresentar o nome do *chaincode* como retorno da função *queryCommited*.

- *Inicialização do chaincode*: Por fim, o *chaincode* aprovado deve ser inicializado nos *peers*. Para isso, deve ser executado o comando *invoke*, utilizando o argumento *–isInit*, conforme Código 12.

Código 11: deployChaincode.sh - Registro do *chaincode*

```
# Registra o chaincode
commitChaincodeDefinition() {
setGlobalsPeer0Org1

peer lifecycle chaincode commit -o localhost:7050
        –ordererTLSHostnameOverride orderer.example.com
        –tls $CORE_PEER_TLS_ENABLED –cafile $ORDERER_CA
        –channelID $CHANNEL_NAME –name ${CC_NAME}
        –collections-config $PRIVATE_DATA_CONFIG
        –peerAddresses localhost:7051 –tlsRootCertFiles $PEER0_ORG1_CA
        –peerAddresses localhost:9051 –tlsRootCertFiles $PEER0_ORG2_CA
        –version ${VERSION} –sequence ${VERSION} –init-required
}
# Verifica chaincode instalado
queryCommitted() {
setGlobalsPeer0Org1

peer lifecycle chaincode querycommitted –channelID $CHANNEL_NAME –name ${CC_NAME}
}
```

Código 12: deployChaincode.sh - Inicia *chaincode*

```
# Funcao de inicializacao do chaincode
chaincodeInvokeInit() {
setGlobalsPeer0Org1

peer chaincode invoke -o localhost:7050
        –ordererTLSHostnameOverride orderer.example.com
        –tls $CORE_PEER_TLS_ENABLED
        –cafile $ORDERER_CA
        -C $CHANNEL_NAME -n ${CC_NAME}
        –peerAddresses localhost:7051 –tlsRootCertFiles $PEER0_ORG1_CA
        –peerAddresses localhost:9051 –tlsRootCertFiles $PEER0_ORG2_CA
        –isInit -c '{"Args":[]}'
}
```

A execução da função *chaincodeInvokeInit* irá inicializar o *chaincode* por meio de uma transação do tipo *invoke*, que não requer argumentos adicionais. Com isso, o *chaincode* está instalado e funcional, sendo possível enviar transações de atualização ou consulta ao *ledger*.

O *script deployChaincode.sh* possui, ainda, a função *chaincodeInvoke*, executada após instalação do *chaincode*. Esta função, presente no *chaincode*, não demanda argumentos, e tem como objetivo inicializar o *chaincode* adicionando um conjunto pré-definido de dados ao *ledger*, conforme apresentado no Código 13.

2.5. Interagindo com a blockchain

Após a inicialização com sucesso do *chaincode*, a blockchain está operacional, sendo possível interagir por meio dos diferentes tipos de transação disponíveis. Esta interação com a blockchain pode ser realizada via linha de comando ou através de API.

2.5.1. Interação via linha de comando

A interação por meio de linha de comando dá-se através da emissão de comandos utilizando o binário *peer*, de modo semelhante aos comandos emitidos nos procedimentos de

Código 13: fabcar.go - Função *initLedger*

```
1  func (s *SmartContract) initLedger(APIstub shim.ChaincodeStubInterface) sc.Response {
2
3  cars := []Car{
4  Car{Make: "Toyota", Model: "Prius", Colour: "blue", Owner: "Tomoko"},
5  Car{Make: "Ford", Model: "Mustang", Colour: "red", Owner: "Brad"},
6  Car{Make: "Hyundai", Model: "Tucson", Colour: "green", Owner: "Jin Soo"},
7  Car{Make: "Volkswagen", Model: "Passat", Colour: "yellow", Owner: "Max"},
8  Car{Make: "Tesla", Model: "S", Colour: "black", Owner: "Adriana"},
9  }
10
11 i := 0
12 for i < len(cars) {
13         carAsBytes, _ := json.Marshal(cars[i])
14         APIstub.PutState("CAR"+strconv.Itoa(i), carAsBytes)
15         i = i + 1
16 }
17 return shim.Success(nil)
18 }
```

criação do canal e instalação do *chaincode*. O repositório compartilhado inclui o *script* interact.sh, desenvolvido para demonstrar exemplos de interação com a blockchain por meio da linha de comando. Neste *script* estão contidas algumas funções de exemplo, dentre as quais três funções básicas: *addCar*, *queryCar* e *queryAllCars*. A função *addCar* recebe como entrada os valores referentes aos campos definidos na função homônima, presente no *chaincode*, e está representada no código 14.

Código 14: interact.sh - Função addCar

```
1  addCar() {
2  setGlobalsForPeer0Org1
3
4  # Solicita dados para transacao
5  echo -e '\nDigite os dados da transacao: \n'
6
7  echo -e '\n Chave: '
8  read chave
9  echo -e '\n Make: '
10 read make
11 echo -e '\n Model: '
12 read model
13 echo -e '\n Color: '
14 read color
15 echo -e '\n Owner: '
16 read owner
17 echo -e '\n'
18
19 # Envia transacao
20 peer chaincode invoke -o localhost:7050
21         –ordererTLSHostnameOverride orderer.example.com
22         –tls $CORE_PEER_TLS_ENABLED
23         –cafile $ORDERER_CA
24         -C $CHANNEL_NAME -n $CC_NAME
25         –peerAddresses localhost:7051 –tlsRootCertFiles
             $PEER0_ORG1_CA
26         –peerAddresses localhost:9051 –tlsRootCertFiles
             $PEER0_ORG2_CA
27         -c '"Args":["createCar", "'$chave'", "'$make'",
             "'$model'", "'$color'", "'$owner'"]'
28 }
```

Após receber e armazenar os dados necessários, a função *addCar* utiliza uma transação *invoke*, informando os *peers* de endosso e *orderer*, passando como argumentos a função do *chaincode* a ser executada, no caso *createCar*, e os argumentos necessários.

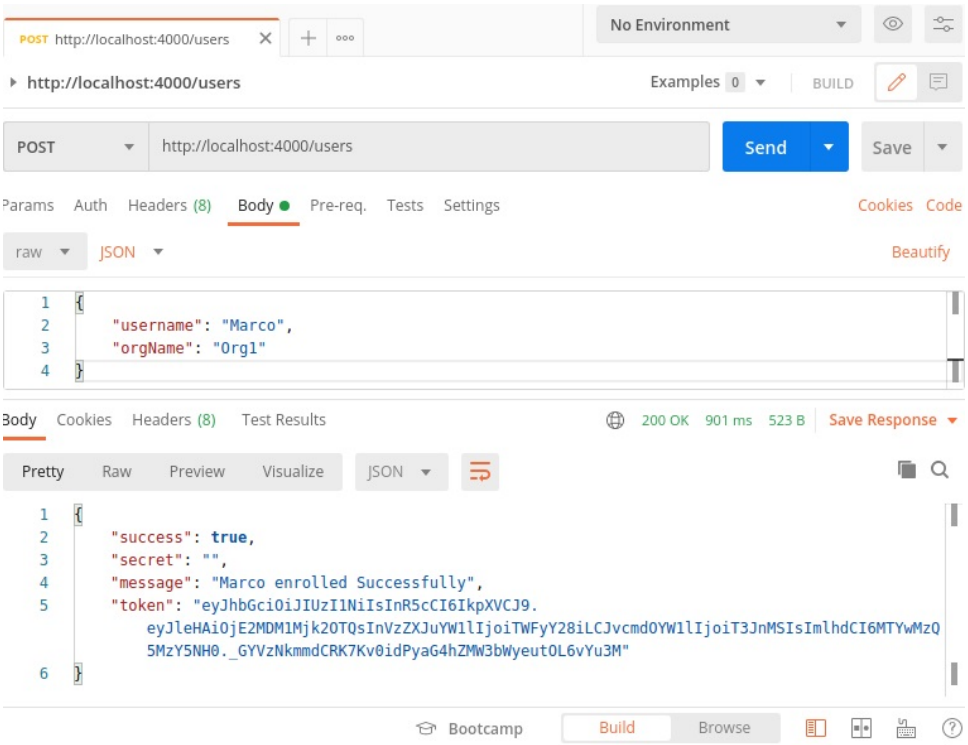
A função *queryCar*, por sua vez, recebe como entrada uma chave e retorna o registro equivalente, ou nada, caso o registro não seja encontrado. Importante notar que esta busca é *case sensitive*. Para realizar esta busca a função utiliza o comando *peer chaincode query* passando como argumentos o canal, nome do *chaincode*, e nos argumentos a função a ser executada, no caso *queryCars* e a chave de busca. Por fim, a função *queryAllCars* não demanda argumentos. Esta função, também contida no *chaincode*, retorna todos os registros contidos na blockchain, por meio do comando *peer chaincode query*, pas-

sando como argumentos o canal, nome do *chaincode* e a função a ser executada, no caso *queryAllCars*.

2.5.2. Interação via API

O repositório possui, no subdiretório ./api-1.4/, uma estrutura de arquivos contendo um conjunto de funções javascript que permitem a interação com a blockchain via API. Para que a interação seja possível é necessária a instalação de um conjunto de módulos do node.js. O *script pre_API.sh* realiza a instalação dos módulos necessários para o correto funcionamento da API. Após a execução do *script*, deve ser executado o comando *node-mon app*, que irá iniciar o servidor e deixar a API acessível via porta 4000. Para interação com a API, será utilizado o aplicativo Postman, um cliente API gratuito que permite a criação e envio de solicitações HTTP e HTTPS, bem como o recebimento das respectivas respostas. A interação segura com a API demanda a geração de um conjunto de usuário e *token*, através do endpoint */users*. A Figura 2.4 apresenta uma solicitação do tipo POST, para geração de usuário e *token*.

Figura 2.4: Solicitação de geração de *token* para usuário



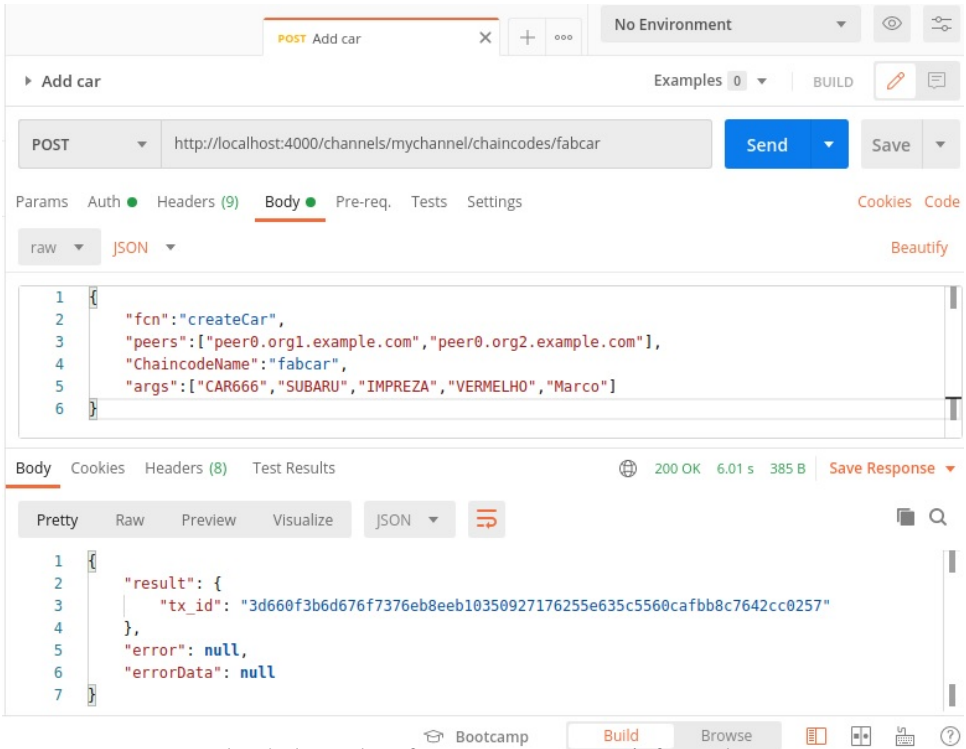
Neste exemplo, como o Postman está instalado no mesmo *host* que a blockchain, a solicitação é direcionada para *http://localhost:4000/users*, contendo no corpo da solicitação o nome de usuário e a organização ao qual pertence. A solicitação é executada e sua resposta apresentada na parte inferior. Na imagem temos uma solicitação executada com sucesso para geração do *token* referente ao usuário Marco, pertencente à organiza-

ção Org1. O conteúdo do campo *token* será, então, utilizado para envio de transações por parte deste usuário.

Transações do tipo *invoke* via API

Para transações do tipo *invoke*, que irão realizar alterações no *ledger*, o *token*, gerado conforme ilustrado na Figura 2.4, deve ser inserido no cabeçalho da solicitação POST. Esta solicitação deve ser encaminhada para */channels/mychannel/chaincodes/fabcar*, informando em seu corpo os parâmetros necessários, como a função a ser executada, os *peers* que irão validar a transação, o nome do *chaincode* e os argumentos da função chamada. Este *endpoint* especifica os respectivos canais e *chaincode* para os quais será enviada a transação, no caso o canal "*mychannel*"e *chaincode fabcar*. A Figura 2.5 apresenta uma solicitação contendo uma transação do tipo *invoke*.

Figura 2.5: Transação do tipo *invoke*



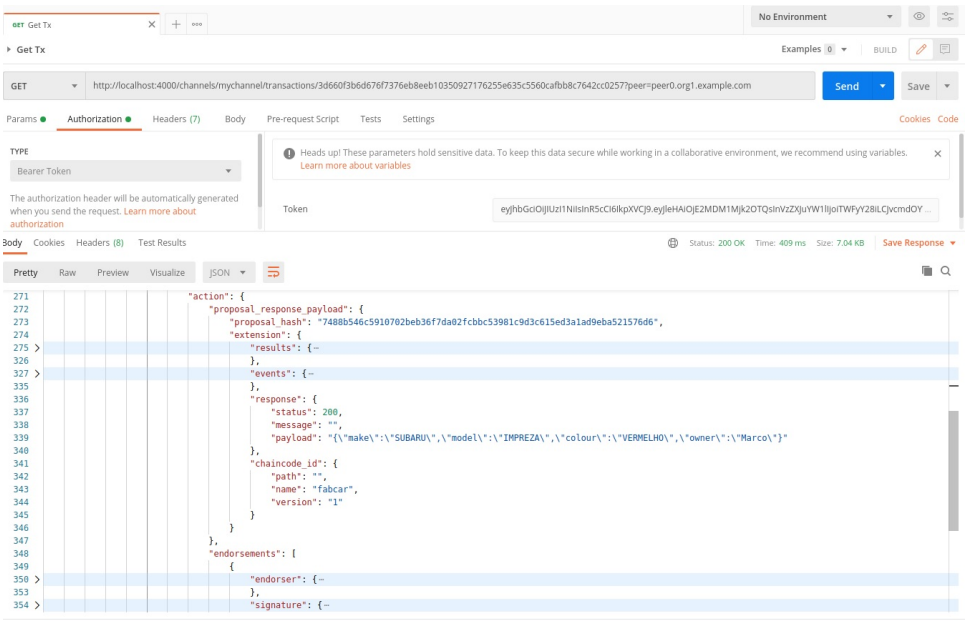
Neste exemplo, é chamada a função *createCar*, e informados os argumentos necessários para sua execução. A execução com sucesso retorna o campo *tx_id*, contendo o identificador da transação realizada.

Transações do tipo *query* via API

As transações do tipo *query* também podem ser executadas via API, com o uso do *token*. O procedimento é similar ao envio de transações do tipo *invoke*, porém mais simples, pois demanda menos argumentos. Para realizar uma consulta por meio de uma transação *query*, é necessário realizar uma solicitação do tipo GET, direcionada para o

endpoint /channels/mychannel/transactions/, informando o *token*, o identificador da transação e o *peer* que será utilizado para consulta. Desta forma, para realizar uma consulta à blockchain quanto à transação realizada na Figura 2.5, deve ser enviada uma solicitação GET, conforme apresentada na Figura 2.6.

Figura 2.6: Transação do tipo *query*



Esta solicitação irá realizar a consulta por meio do *peer0.org1.example.com*, utilizando o *token* informado. Como resposta, obtém-se um conjunto completo de informações sobre a transação, contendo seu conteúdo, os nós que endossaram a transação, seu *hash*, dentre outros.

2.5.3. Hyperledger Explorer

O Hyperledger Explorer é uma ferramenta que oferece uma interface para visualização das transações em uma blockchain Hyperledger Fabric. Dentre suas principais funcionalidades destacam-se sua interface web amigável, métricas referentes aos blocos e transações, ferramentas de busca e filtros para blocos e transações, descoberta dinâmica de novos canais e notificação em tempo real de novos blocos. Esta ferramenta está disponibilizada no repositório clonado, especificamente no subdiretório *./blockchainexplorer*. Sua execução depende, primeiramente, que a blockchain Hyperledger Fabric esteja configurada e em execução. Em seguida, é necessária a configuração correta dos arquivos de configuração *docker-compose.yaml*, *config.json* e *first-network.json*, estando os dois primeiros localizados no diretório *./blockchainexplorer* e o último no subdiretório *./blockchainexplorer/connection-profile*.

Esta aplicação pode ser executada de maneira local ou em contêineres Docker. Neste trabalho o Hyperledger Explorer é executado em ambiente Docker, sendo composto por dois contêineres: explorer e explorer-db. O contêiner explorer abrange o *front-end* da

aplicação, enquanto o explorer-db abriga o banco de dados PostgreSQL, responsável por persistir os dados do Hyperledger Fabric.

O primeiro passo para execução do Hyperledger Explorer envolve a configuração do arquivo *docker-compose.yaml*. Neste arquivo serão definidas as configurações dos contêineres que compõe a aplicação, tais como variáveis de ambiente, nome do *host*, nome do contêiner, rede, portas, volumes e dependências. O mapeamento dos volumes é essencial para integração com a blockchain, e já estão pré-configurados conforme a estrutura de diretórios definida no repositório, sendo necessário verificar apenas o volume referente à pasta contendo o material criptográfico (*/cripto-config/*). O arquivo *docker-compose.yaml* presente no repositório já considera o Hyperledger Explorer como uma subpasta do projeto. Desta forma, o volume referente à pasta cripto-config aponta para *../artifacts/channel/crypto-config/*.

O arquivo *config.json* informa qual o perfil a ser utilizado para as configurações de rede. No caso, o perfil definido aponta para o subdiretório *./connetion-profile/*, onde está localizado o arquivo *first-network.json*, que contém as configurações referentes à blockchain e ao nó com o qual o Hyperledger Explorer se comunicará para obter as informações. Este arquivo é composto por subgrupos contendo configurações específicas, tais como *client, channels, organizations e peers*. Em *client* estão as configurações quanto ao uso de TLS, usuário e senha de administrador, organizações participantes da rede, e *timeout* de conexão de nós *peer* e *orderer*. Em *channels* estão listadas as configurações dos canais, tais como os nós, e *timeout de conexão*. Em *organizations* constam o caminho referente à chave privada e ao certificado do administrador do MSP das organizações. Por fim, em *peers* estão listadas configurações e caminho do certificado CA dos nós *peer* e respectiva URL.

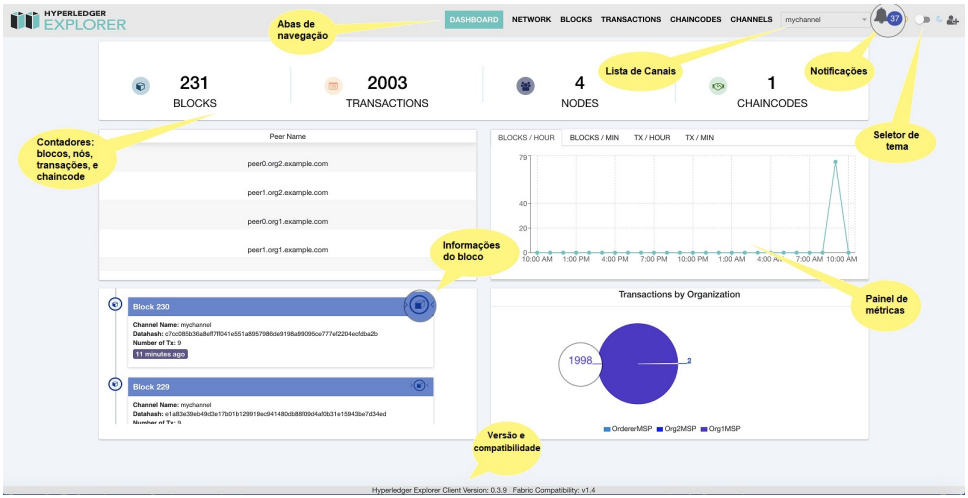
Com a configuração adequada destes arquivos, é possível iniciar a aplicação Hyperledger Explorer por meio do comando *docker-compose up -d*, a ser executado dentro do subdiretório *./blockchainexplorer*. A execução deste comando irá baixar as imagens necessárias, definidas em *docker-compose.yaml* e montar os contêineres. A interface da aplicação pode ser acessada via navegador, do próprio *host*, apontando para *http://localhost:8080* ou de outra máquina da rede, apontando para *http://<IP do host>:8080*. A Figura 2.7 ilustra a interface principal da aplicação.

A interface da aplicação apresenta um conjunto de painéis com as principais informações da blockchain, incluindo os últimos blocos e métricas referentes à blocos e transações. O menu principal é composto de um conjunto de abas de navegação, com acesso à informações sobre a rede, blocos, transações, *chaincodes* e canais. Desta forma, é possível navegar e explorar o conteúdo dos blocos, verificar seu *hash* e as transações que contém. É possível também verificar detalhes das transações, incluindo data e hora, os nós que a endossaram, seu *hash* e conteúdo.

2.6. Considerações

O modelo de implementação proposto e executado no presente trabalho busca detalhar as etapas de configuração e instalação de uma blockchain Hyperledger Fabric permissionada. Esta proposta tem enfoque educacional, para implementação em ambiente de testes. Para implementação em ambiente de produção, os códigos e configurações apre-

Figura 2.7: Hyperledger Explorer *dashboard* - Fonte: Adaptado de [Hyperledger 2020b]



sentados devem ser revistos, levando em conta as melhores práticas de desenvolvimento e segurança da informação. Neste sentido, o objetivo deste trabalho é apresentar os principais conceitos, configurações e arquitetura da solução de modo que, de posse destes conhecimentos, seja possível modelar e implementar soluções baseadas em blockchain Hyperledger Fabric em diversos cenários.

A tecnologia blockchain vem sendo amplamente estudada e implementada nos mais variados setores. Porém, apesar de seu potencial disruptivo, esta nem sempre é a melhor alternativa, sendo necessária uma análise criteriosa, considerando questões como integração com sistemas legado, capacidade e velocidade de processamento, escalabilidade e maturidade da tecnologia, afim de comprovar sua viabilidade e benefícios perante as soluções já existentes.

Agradecimentos

Os autores agradecem o apoio do Laboratório de Processamento Paralelo Distribuído (LabP2D) no Centro de Ciências Tecnológicas (CCT) da Universidade do Estado de Santa Catarina (UDESC).

Os autores agradecem o apoio da Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC).

This work was supported by Ripple's University Blockchain Research Initiative.

Referências

[Dhillon et al. 2017] Dhillon, V., Metcalf, D., and Hooper, M. (2017). The hyperledger project. In *Blockchain enabled applications*, pages 139–149. Springer.

[Hyperledger 2020a] Hyperledger (2020a). A blockchain platform for the enterprise. https://hyperledger-fabric.readthedocs.io/en/release-2.2/index.html.

[Hyperledger 2020b] Hyperledger (2020b). Hyperledger explorer docs. https://blockchain-explorer.readthedocs.io.

[Hyperledger.Architecture 2017] Hyperledger.Architecture (2017). Hyperledger architecture paper 1 consensus. https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf.

[Hyperledger.Smartcontracts 2017] Hyperledger.Smartcontracts (2017). Hyperledger architecture volume 2 - smart contracts. https://www.hyperledger.org/wp-content/uploads/2018/04/Hyperledger_Arch_WG_Paper_2_SmartContracts.pdf.

[Miers et al. 2019] Miers, C., Koslovski, G., Pillon, M., Simplicio, M., Carvalho, T., Rodrigues, B., and Battisti, J. (2019). *Análise de Mecanismos para Consenso Distribuído Aplicados a Blockchain*.

[Ongaro and Ousterhout 2014] Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA. USENIX Association.

[Padvan 2020] Padvan, A. (2020). Basic network 2.0. https://github.com/adhavpavan/BasicNetwork-2.0.

[Salman et al. 2019] Salman, T., Zolanvari, M., Erbad, A., Jain, R., and Samaka, M. (2019). Security services using blockchains: A state of the art survey. *IEEE Communications Surveys Tutorials*, 21(1):858–880.

[Sankar et al. 2017] Sankar, L. S., Sindhu, M., and Sethumadhavan, M. (2017). Survey of consensus protocols on blockchain applications. In *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 1–5. IEEE.

[Sharma 2020] Sharma, T. K. (2020). Permissioned and permissionless blockchains: A comprehensive guide. https://www.blockchain-council.org/blockchain/permissioned-and-permissionless-blockchains-a-comprehensive-guide/.

[Swan 2015] Swan, M. (2015). *Blockchain: Blueprint for a new economy*. "O'Reilly Media, Inc.".

[Tinu 2018] Tinu, N. (2018). A survey on blockchain technology- taxonomy, consensus algorithms and applications. *International Journal of Computer Sciences and Engineering O*, 6.

[Yaga et al. 2018] Yaga, D., Mell, P., Roby, N., and Scarfone, K. (2018). Nistir 8202 - blockchain technology overview. Technical report.

[Yaga et al. 2019] Yaga, D., Mell, P., Roby, N., and Scarfone, K. (2019). Blockchain technology overview. *CoRR*, abs/1906.11078.

Capítulo 3

Introdução à Verificação Automática de Protocolos de Segurança com Scyther

Diego Kreutz (UNIPAMPA), Rodrigo Mansilha (UNIPAMPA), Silvio E. Quincozes (UFF), Tadeu Jenuário (UNIPAMPA), João Otávio Chervinski (Monash University)

Resumo

Os protocolos de segurança representam o alicerce das comunicações realizadas na Internet. Um dos principais desafios no projeto desses protocolos é garantir a sua própria segurança. Para superar esses desafios, foram desenvolvidas ferramentas de verificação formal e automática de protocolos de segurança, como a Scyther, CryptoVerif, ProVerif, AVISPA e Tamarim Prover. Entretanto, essas ferramentas são ainda pouco conhecidas e utilizadas na prática por projetistas de protocolos e estudantes de computação. Este minicurso, cujo conteúdo está disponível no GitHub[1], visa diminuir essa lacuna através da apresentação da ferramenta Scyther e sua aplicação na verificação de quatro protocolos de segurança.

3.1. Introdução

Com o surgimento de novos instrumentos legais, como a Lei Geral de Proteção de Dados (LGPD)[2], a segurança na Internet, mais especificamente a proteção dos dados em trânsito, é um dos assuntos que entrou na pauta de diversas nações. As entidades comunicantes e os dados em trânsito são autenticados e protegidos através de protocolos de segurança como o *Needham-Schroeder* (NS) [Needham and Schroeder 1978], *Diffie-Hellman* (DH) [Steiner et al. 1996], *Internet Key Exchange* (IKE) [Carrel and Harkins 1998], *Internet Protocol Security* (IPsec) [Atkinson 1995], *Secure Shell* (SSH) [Lonvick and Ylonen 2006], *Transport Layer Security* (TLS) [Rescorla 2018] e *Signal Protocol*[3].

Um dos principais desafios de projeto e implementação de sistemas e protocolos de segurança da informação é garantir a corretude e a resistência a ataques com

[1] https://github.com/scyther-lea/errc2020
[2] http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm
[3] https://signal.org/

o estabelecimento de propriedades como confidencialidade, integridade, disponibilidade, autenticidade e legalidade [Rainer et al. 2020]. A verificação automática e formal de sistemas e protocolos, utilizando métodos formais e matemáticos, tem sido cada vez mais utilizada para analisar de forma sistemática e automática a robustez de protocolos de segurança [Chen et al. 2016, Baelde et al. 2017, Chudnov et al. 2018, Li et al. 2018, Bai et al. 2018, Liu and Liu 2019, Kreutz et al. 2019, Jenuario et al. 2020, Cohn-Gordon et al. 2020]. Recorrentemente, pesquisas relatam, por exemplo, que o processo de verificação formal já contribuiu de maneira significativa para a correção de protocolos e serviços que, embora vulneráveis, estavam em utilização na Internet [Dalal et al. 2010, Arapinis et al. 2012, Affeldt and Marti 2013, Cremers et al. 2016, Delia Jurcut et al. 2018, Cook 2018].

Existem diversas ferramentas desenvolvidas especificamente para auxiliar o projeto e a verificação automática de protocolos, como Scyther[4] [Cremers 2006, Cremers 2008], CryptoVerif[5] [Blanchet 2006], AVISPA [Armando, A., et al. 2005], ProVerif [Blanchet et al. 2018] e Tamarim Prover[6] [Meier et al. 2013]. No entanto, essas ferramentas são pouco conhecidas e utilizadas pela comunidade. Por exemplo, encontramos pouquíssimos trabalhos (*e.g.*, alguns trabalhos dos autores deste minicurso) sobre verificação formal de protocolos de segurança utilizando essas ferramentas nos principais cursos de computação do Rio Grande do Sul e nas últimas três edições do Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg) e do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg), dois dos principais eventos de segurança do Brasil.

Ao final deste minicurso, o estudante (ou profissional) reconhecerá o potencial do emprego de ferramentas de verificação automática e formal de protocolos de segurança. Especificamente, o estudante (ou profissional) será capaz de:

(o_1) conhecer as semânticas operacionais da ferramenta Scyther (Seção 3.2);

(o_2) aplicar a Scyther na análise do protocolo Atualização de Chave Simétrica (ACS) (Seção 3.3);

(o_3) aplicar a Scyther na verificação automática e correção do protocolo Wide Mouth Frog (WMF) [Kelsey et al. 1997, Velibor et al. 2016] (Seção 3.4);

(o_4) aplicar a Scyther na verificação automática e correção do protocolo Needham-Schroeder (NS) [Needham and Schroeder 1978] (Seção 3.5);

(o_5) aplicar a Scyther na análise do protocolo Four-party Generalized Needham-Schroeder-Lowe (β) [Cremers and Mauw 2006] (Seção 3.6).

Considerações finais sobre verificação automática utilizando a ferramenta Scyther são apresentadas na Seção 3.7. Além disso, vale enfatizar que o conteúdo do mini-

[4] https://people.cispa.io/cas.cremers/scyther/
[5] https://prosecco.gforge.inria.fr/personal/bblanche/cryptoverif/
[6] https://tamarin-prover.github.io/

curso, incluindo os códigos dos protocolos na semântica da Scyther e um exemplo de implementação em Python, estão disponíveis em um repositório do GitHub[7].

3.2. A Ferramenta Scyther

Nesta seção, iremos introduzir as semânticas operacionais e práticas com a ferramenta Scyther [Cremers 2008]. As semânticas operacionais (Seção 3.2.1) são necessárias para traduzir um protocolo de segurança convencional, apresentado numa notação específica, para a semântica da Scyther. Na Seção 3.2.2, apresentamos um primeiro exemplo prático de utilização da ferramenta em ambiente Linux, bem como os pré-requisitos de sistema.

3.2.1. Semânticas Operacionais

Cada ferramenta de verificação automática e formal de protocolos de segurança utiliza linguagens e semânticas próprias. As principais semânticas operacionais da ferramenta Scyther [Cremers 2006, Cremers 2008] são apresentadas a seguir.

Termos atômicos: a ferramenta Scyther manipula termos de modo que o protocolo apresente o comportamento desejado. Os termos atômicos, ou tipos específicos de dados, servem para definir os dados que serão utilizados no protocolo. Entre eles estão: o **var**, que define variáveis para armazenar os dados recebidos de uma comunicação; o **const**, que define constantes que são geradas para cada instanciação de uma função e possuem visibilidade local; e o **fresh**, que define elementos inicializados com valores pseudo-aleatórios.

Chaves simétricas: qualquer termo atômico pode atuar como chave para encriptação simétrica através da notação `{dado}termo`, que corresponde a encriptação de `dado` usando `termo` como chave simétrica (na Seção 3.3 é apresentado um exemplo de protocolo baseado em chaves simétricas). Por padrão, `termo` é interpretado como parte de um infraestrutura de chave simétrica, a menos que seja explicitamente definido como elemento de um par de chaves assimétricas, como segue.

Chaves assimétricas: uma infraestrutura de chave públicas ou assimétricas é predefinida pela ferramenta, sendo `sk(X)` correspondente a uma chave privada de `X` e `pk(X)` a chave pública correspondente. Um dado cifrado utilizando uma chave pública (*e.g.*, `{dado}pk(X)`) só poderá ser decifrado com a chave privada correspondente (`sk(X)`). É importante observar que a Scyther permite modelar infraestruturas de chaves assimétricas adicionais (*e.g.,* usando `termo` como uma das chaves). Para fins didáticos, esse processo é omitido deste minicurso. No entanto, recomendamos que o leitor interessado consulte os tópicos avançados do manual da Scyther[8].

Função Hash: permite embaralhamento através de uma função de resumo criptográfico cuja a função inversa é desconhecida por todos — assim, torando-se impraticável a obtenção dos dados originais a partir e tal resumo. A função Hash pode ser definida através de uma declaração como **hashfunction** `H`, que corresponde a definição de uma função Hash denominada `H`. Essa função pode ser usada da seguinte forma: `H(dado)`, que corresponde a geração de um resumo criptográfico de um `dado` através da função hash `H`. É importante notar que as funções hash criptográficas normalmente são definidas

[7] https://github.com/scyther-lea/errc2020
[8] https://github.com/cascremers/scyther/blob/master/gui/scyther-manual.pdf

com visibilidade global, pois devem ser conhecidas por todos participantes.

Tipos predefinidos: determinam o comportamento de uma função ou termo. Por exemplo, o tipo **Agent** é utilizado para criar um agente que irá interagir nas comunicações. **Function** é um tipo especial que define um termo como sendo uma função que pode receber uma lista de termos como parâmetro. O tipo **Nonce** é considerado padrão para termos e é destinado ao armazenamento de valores utilizados durante a troca de mensagens.

Tipos básicos de eventos: incluem **send** (enviar) e **recv** (receber). Eles são utilizados para a comunicação entre os agentes de um protocolo (*e.g.*, Alice e Bob). É comum que cada evento de envio (*e.g.*, **send_1**) possua um evento de recebimento correspondente (*e.g.*, **recv_1**). Assim, para representar o envio de uma mensagem de Alice para Bob, contendo o dado cifrado com a chave pública de Bob, pode-se utilizar a sintaxe **send_1**(`Alice`, `Bob`,{`dado`}`pk`(`Bob`)), onde o `dado` é declarado por Alice como, por exemplo, uma variável **fresh** do tipo **Nonce**. Similarmente, para representar o recebimento da mensagem enviada de Alice para Bob, basta utilizar a sintaxe **recv_1**(`Alice`, `Bob`,{`dado`}`pk`(`Bob`)). Com isso, Bob irá receber a mensagem de Alice e utilizar a sua chave privada correspondente para decifrar o dado. O dado será armazenado em uma variável da semântica operacional da Scyther, como uma variável **var** do tipo **Nonce**.

Eventos de afirmação (**claim**): são utilizados em especificações de funções para modelar propriedades de segurança. Na prática, após afirmar que uma variável é secreta, a ferramenta irá verificar se a afirmação procede durante a execução do protocolo (*i.e.,* verificar se somente as partes comunicantes possuem acesso ao dado secreto). Por exemplo, para afirmar que uma variável `nonceB` é secreta, utiliza-se o termo **Secret** dentro de um evento de afirmação (*i.e.,* **claim**(`Bob`,`Secret`,`nonceB`)). Também pode-se utilizar o termo **Nisynch** dentro de um evento de afirmação (*i.e.,* **claim**(`Bob`,`Nisynch`)) para verificar se todas as mensagens recebidas pelo receptor foram de fato enviadas pelo parceiro de comunicação e não por um agente desconhecido.

3.2.2. Práticas com Scyther

Durante o minicurso, nós utilizaremos a versão v1.1.3 (via linha de comando) da ferramenta Scyther, para os sistemas operacionais Linux[9], Windows[10] e Mac OS X[11]. Nesta e nas próximas seções, iremos demonstrar a utilização prática com a versão para Linux, testes realizados na distribuição Debian 10, utilizando o Python na versão 2.7.16.

Ao descompactar o arquivo `scyther-linux-v1.1.3.tgz`, é criado o diretório `scyther-linux-v1.1.3`. Dentro dele há o código Python da Scyther, incluindo o arquivo `scyther.py`, que permite a execução da ferramenta via linha de comando. Ao ser executada com a opção `--help`, a ferramenta exibe suas opções e parâmetros, entre as quais destacamos as seguintes.

(p_1) `--filter=<protocol>[,<label>]` para checar apenas algumas claims específicas do protocolo;

[9] https://people.cispa.io/cas.cremers/downloads/scyther/scyther-linux-v1.1.3.tgz
[10] https://people.cispa.io/cas.cremers/downloads/scyther/scyther-w32-v1.1.3.zip
[11] https://people.cispa.io/cas.cremers/downloads/scyther/scyther-mac-v1.1.3.tgz

(p_2) `-a, --auto-claims` para ignorar toda e qualquer *claim* existente (definida pelo projetista do protocolo) e gerar automaticamente as *claims* para o protocolo;

(p_3) `-r, --max-runs=<int>` para definir o número máximo `<int>` de execuções (o valor padrão da ferramenta é 5);

(p_4) `-A, --all-attacks` para a ferramenta gerar todos os ataques dentro do espaço de estado com protocolo (sem a definição desta opção, a ferramenta gera apenas 1 ataque).

Vamos utilizar como exemplo o Protocolo 3.1, ilustrado também no diagrama da Figura 3.1, que ilustra uma comunicação entre Alice e Bob utilizando criptografia assimétrica (chaves públicas). No protocolo hipotético, Alice envia (linha 1) para Bob um `nonceA` e Bob responde com um `nonceB` (linha 2). Como pode ser observado, é assumido que tanto Alice quanto Bob possuem um par de chaves pública e privada.

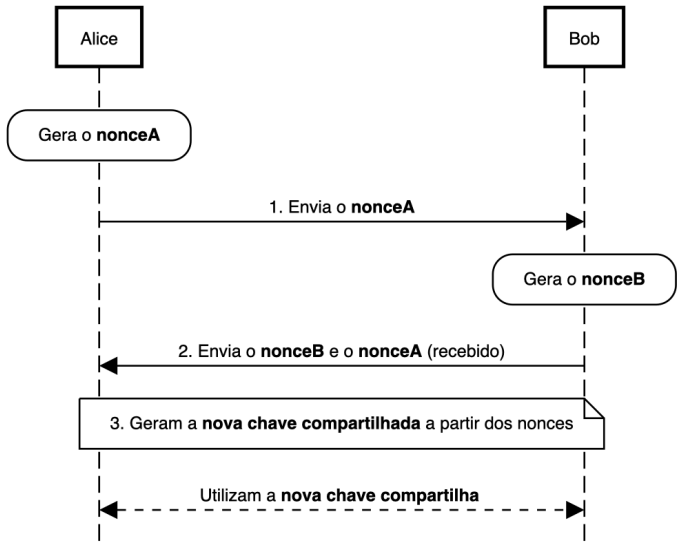


Figura 3.1: Ilustração do exemplo de comunicação com chaves públicas

Protocolo 3.1: Exemplo de comunicação com chaves públicas

1. Alice $\rightarrow$ Bob	[Alice, $E_{pk_{Bob}}$(nonceA)]
2. Bob $\rightarrow$ Alice	[Bob, $E_{pk_{Alice}}$(nonceB,nonceA)]
3. Bob, Alice	K $\leftarrow$ H(nonceA $\|\|$ nonceB)

Vale ressaltar que a semântica de especificação do protocolo é tipicamente diferente da semântica operacional da ferramenta de verificação automática. A especificação

dos protocolos segue uma semântica própria, mais geral e em nível mais alto de abstração, como é o caso do Protocolo 3.1. Posteriormente, a semântica do protocolo deve ser traduzida para a semântica específica de cada uma das ferramentas de verificação automática que serão utilizadas para validar o protocolo. No caso deste minicurso, iremos converter a semântica geral para a semântica operacional da ferramenta Scyther (vide Protocolo 3.1 traduzido para a semântica da Scyther no Algoritmo 3.1).

Protocolo 3.1, Alice gera e envia para Bob o `nonceA` (linha 1). A mensagem é cifrada (função E, do inglês E*ncrypt*) com a chave pública do Bob ($E_{pk_{Bob}}$). Ao receber a mensagem, Bob decodifica-a e recupera o `nonceA`, gera e envia um novo `nonceB`, junto ao `nonceA`, para Alice (linha 2) – ambos cifrados com a chave pública da receptora ($E_{pk_{Alice}}$). Alice verifica que o *nonce* `nonceA` recebido é o mesmo que foi enviado para Bob anteriormente. Finalmente, Alice e Bob geram a chave simétrica compartilhada `K` (linha 3) a partir dos *nonces* concatenados ($||$), utilizados como entrada para uma função hash criptográfica `H` (*e.g.*, SHA256). Com isso, nas comunicações subsequentes, Alice e Bob passam a utilizar criptografia simétrica ao invés de assimétrica.

O Algoritmo 3.1 corresponde a uma representação do Protocolo 3.1 na semântica operacional da Scyther. Vale ressaltar que a linha 3 do protocolo não é representada no algoritmo da Scyther por questão de simplicidade de ilustração. Além disso, se não ocorrer nenhum ataque e comprometimento às linhas 1 e 2 do protocolo, a geração e a utilização da chave secreta compartilhada (linha 3) pode ser considerada segura.

Algoritmo 3.1: Protocolo 3.1 na semântica da Scyther

```
1  const pk: Function;
2  secret sk: Function;
3  inversekeys (pk,sk);
4  const Eve: Agent;
5  untrusted Eve;
6  protocol exemplo(Alice,Bob,Eve){
7     role Alice{
8        fresh nonceA: Nonce;
9        var nonceB: Nonce;
10       send_1(Alice,Bob,{nonceA}pk(Bob));
11       recv_2(Bob,Alice,{nonceA,nonceB}pk(Alice));
12       claim(Alice,Secret,nonceA);
13       claim(Alice,Secret,nonceB);
14       claim(Alice,Nisynch);
15    }
16    role Bob{
17       var nonceA: Nonce;
18       fresh nonceB: Nonce;
19       recv_1(Alice,Bob,{nonceA}pk(Bob));
20       send_2(Bob,Alice,{nonceA,nonceB}pk(Alice));
21       claim(Bob,Secret,nonceA);
22       claim(Bob,Secret,nonceB);
23       claim(Bob,Nisynch);
24    }
25 }
```

Nas linhas 1, 2 e 3, são definidas e declaradas como inversas (uma cifra utilizando a chave pública pode ser decifrada com a respectiva chave privada) as chaves pública `pk`

e privada `sk`. Na linha 4, é declarado um agente malicioso Eve, que não é confiável (**untrusted**, na linha 5).

A declaração do protocolo é realizada na linha 6, definindo os três agentes, Alice, Bob e Eve. Para Alice e Bob há um papel (**role**) explicito no protocolo, nas linhas 7 e 16, respectivamente. Os `nonceA` e `nonceB` são declarados como **fresh** (novo valor gerado) e **var** (valor recebido e armazenado) para a Alice e o inverso para Bob. Há dois pares de **send** e **recv** entre Alice e Bob. No **send_1** (linha 10) e **recv_1** (linha 19) é enviado o *nonce* `nonceA` da Alice para o Bob. Já no **send_2** (linha 20) e **recv_2** (linha 11) são enviados os *nonces* `nonceB` e `nonceA` do Bob para a Alice. Finalmente, os papéis são encerrados com três *claims* (linhas 12 a 14 e 21 a 23). Duas *claims* são utilizadas para afirmar que os *nonces* `nonceA` e `nonceB` são secretos. A última é utilizada para verificar se as mensagens recebidas foram, de fato, enviadas pelo par legítimo e não por um impostor intermediário, como o agente malicioso Eve.

Para realizar a verificação automática do Algoritmo 3.1, basta copiar o código para um arquivo de texto (*e.g.*, `protocolo_exemplo.spdl`) e passá-lo como parâmetro para a ferramenta Scyther, como ilustrado na Verificação 3.1. Como pode ser observado, há pelo menos um ataque que pode comprometer o protocolo. Bob (linha 19 do Algoritmo 3.1 e linha 1 do Protocolo 3.1) não dispõem de nenhuma forma de confirmação acerca da procedência de `nonceA` (*i.e.,* ele não é capaz de verificar se veio da Alice ou não).

```
./scyther.py --all-attacks --max-runs=5 protocolo_exemplo.spdl

Verification results:
claim id [exemplo,Alice1], Secret(nA) : No attacks.
claim id [exemplo,Alice2], Secret(nB) : No attacks.
claim id [exemplo,Alice3], Nisynch    : No attacks.
claim id [exemplo,Bob1], Secret(nA)   : Exactly 1 attack.
claim id [exemplo,Bob2], Secret(nB)   : No attacks.
claim id [exemplo,Bob3], Nisynch      : Exactly 1 attack.
```

Verificação 3.1: Execução e saída da Scyther para o Algoritmo 3.1

Eve, o agente malicioso, pode interceptar o envio de Alice para Bob, mudar o `nonceA` e usar a chave pública do Bob para entregar a mensagem a ele. Bob irá reponder para Eve, que irá responder para Alice, que também não consegue verificar se a mensagem veio do Bob ou não. É importante observar que, apesar de ser um protocolo muito simples, há uma falha grave (um ataque), o que reforça a importância de utilizarmos ferramentas de verificação automática de protocolos. Para resolver o problema, é necessário incluir a identificação do remetente e do destinatário nas mensagens cifradas. Na Seção 3.5 discutimos um ataque similar, incluindo o fluxo do ataque e os detalhes da solução do problema, para o protocolo Needham-Schroeder.

3.3. O protocolo de Atualização de Chave Simétrica (ACS)

3.3.1. Protocolo didático de utilização de chaves simétricas

O protocolo didático de Atualização de Chave Simétrica (ACS), criado pelos autores, realiza a atualização da chave simétrica, compartilhada entre Alice e Bob, através de uma chave privada previamente compartilhada entre tais entidades, como ilustrado no diagrama da Figura 3.2. A atualização da chave de sessão, utilizada por ambas as partes, é realizada pelo agente que inicia a comunicação (*e.g.*, Alice, no cenário ilustrado a seguir).

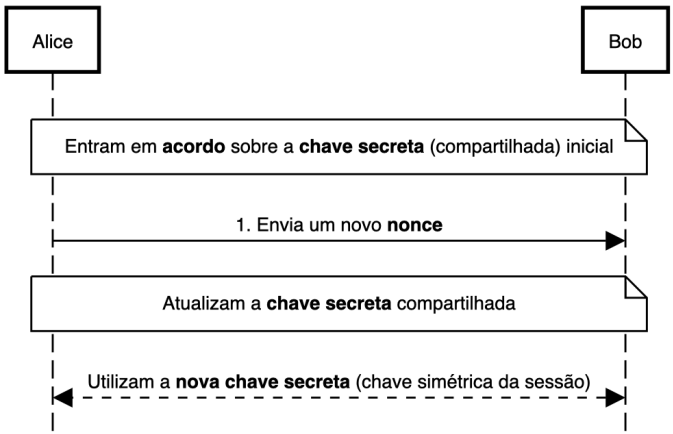


Figura 3.2: Ilustração de Alice e Bob utilizando o ACS

O Protocolo 3.2 detalha o funcionamento do ACS. Para atualizar a chave de sessão, Alice envia uma mensagem ao Bob contendo um `nonce` (linha 1). A chave `K` e o `nonce` são concatenados como parâmetro de uma função hash criptográfica `H` e o resumo criptográfico de saída é atribuído para a chave `K`, atualizando a chave secreta de sessão. Tal prática também é comumente utilizada na geração de códigos de autenticação de mensagem HMACs (*Hash Message Authentication Codes*) [Krawczyk et al. 1997].

Protocolo 3.2: Especificação do ACS entre Alice e Bob

1. Alice $\rightarrow$ Bob	$[E_K(\text{nonce})]$
2. Bob, Alice	K $\leftarrow$ H(K $\|\|$ nonce)

3.3.2. Verificação automática do protocolo

O Algoritmo 3.2 descreve o Protocolo 3.2 (ACS) na semântica da Scyther. Na linha 1, é criada a chave de sessão (secreta) `K`, compartilhada entre Alice e Bob. É importante observar que a própria ferramenta Scyther oferece nativamente a definição de chave compartilhada `k` (minúsculo), que possui a mesma finalidade. Entretanto, neste exemplo de verificação automática, utilizaremos a nossa declaração `K` para fins didáticos (*e.g.*, ficar

mais próximo da sintaxe do Protocolo 3.2). Alice envia um novo `nonce`, cifrado utilizando a chave secreta `K`, para Bob (linha 7). Bob recebe (linha 13) e decifra o `nonce`, que será utilizado para atualizar a chave `K`, conforme descrito no Protocolo 3.2. Tanto Alice quanto Bob fazem dois *claims* idênticos – ambos afirmam que o `nonce` e a chave de sessão `K` são secretos.

Algoritmo 3.2: ACS na semântica da Scyther

```
1  secret K: SessionKey;
2  const Eve: Agent;
3  untrusted Eve;
4  protocol ACS(Alice,Bob,Eve){
5     role Alice{
6        fresh nonce: Nonce;
7        send_1(Alice,Bob,{nonce}K(Alice,Bob));
8        claim_ Alice1(Alice,Secret,nonce);
9        claim_ Alice2(Alice,Secret,K);
10    }
11    role Bob{
12       var nonce: Nonce;
13       recv_1(Alice,Bob,{nonce}K(Alice,Bob));
14       claim_Bob1(Bob,Secret,nonce);
15       claim_Bob2(Bob,Secret,K);
16    }
17 }
```

Como pode ser visto na saída da Verificação 3.2, o ACS não possui nenhuma falha. O ACS é apenas um protocolo didático, muito simples, para ilustrar a utilização de chaves simétricas para realizar uma operação (*i.e.,* atualização da própria chave de sessão, nesta demonstração) entre Alice e Bob.

```
./scyther.py  --all-attacks  --max-runs=5 protocolo_acs.spdl

Verification results:
claim id [ACS,Alice1], Secret(nonce) : No attacks.
claim id [ACS,Alice2], Secret(K)     : No attacks.
claim id [ACS,Bob1], Secret(nonce)   : No attacks.
claim id [ACS,Bob2], Secret(K)       : No attacks.
```

Verificação 3.2: Execução e saída da Scyther para o Algoritmo 3.2

3.4. O protocolo Wide Mouth Frog (WMF)

3.4.1. Protocolo para Troca de Chave Simétrica

O protocolo WMF realiza a troca da chave simétrica, compartilhada entre Alice e Bob, através de um agente confiável (Charles) utilizando uma chave privada (*private key*) compartilhada [Burrows 1989, Burrows et al. 1990, Kelsey et al. 1997, Velibor et al. 2016]. Essa chave é utilizada exclusivamente para distribuição das chaves. Resumidamente, no WMF, quem gera a chave de sessão, utilizada por Alice e Bob, é o agente que inicia a comunicação (*e.g.*, Alice).

O diagrama da Figura 3.3 e o Protocolo 3.3 ilustram e detalham o funcionamento do WMF, respectivamente. Para trocar a chave de sessão, Alice envia uma mensagem

ao agente confiável Charles contendo sua identificação, um *timestamp* $\mathtt{T}_a$, a identificação do destinatário (Bob) e a nova chave de sessão $\mathtt{K}_{ab}$, que será utilizada por ambos para assegurar propriedades de segurança (*e.g.*, confidencialidade, integridade e autenticidade) das mensagens trocadas. A mensagem de Alice (linha 1 do Protocolo 3.3) é cifrada (E) utilizando a chave $\mathtt{K}_{ac}$, compartilhada entre ela e Charles.

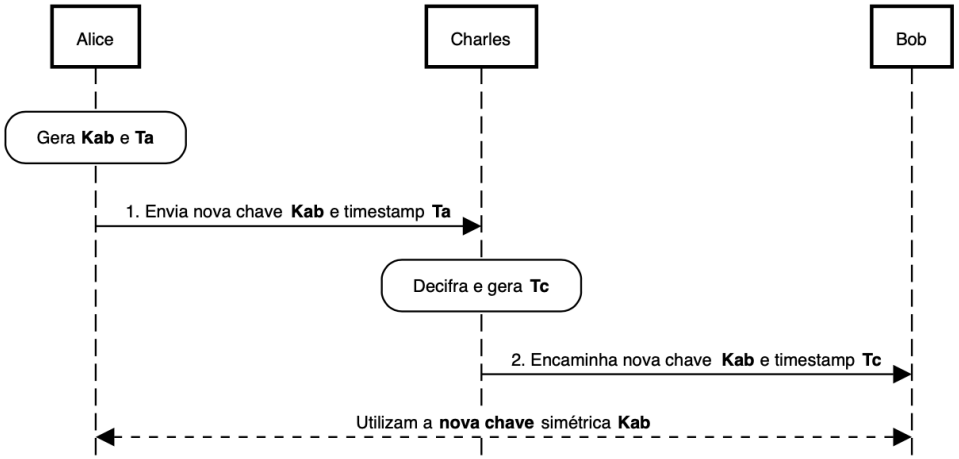


Figura 3.3: Ilustração de Alice e Bob utilizando o WMF

Charles, ao receber a mensagem de Alice, decifra ela utilizando a chave K_{ac} e cria uma nova mensagem contendo sua identificação (`Charles`), um *timestamp* $\mathtt{T}_c$, o identificador do remetente (`Alice`) e a chave de sessão compartilhada $\mathtt{K}_{ab}$, criada e enviada por Alice (linha 2). A mensagem é cifrada (E) utilizando a chave $\mathtt{K}_{bc}$, compartilhada entre Charles e Bob.

Protocolo 3.3: Especificação do WMF entre Alice e Bob

1. Alice $\rightarrow$ Charles	[Alice, $\mathrm{E}_{K_{ac}}(\mathrm{T}_a$,Bob,$\mathrm{K}_{ab})$]
2. Charles $\rightarrow$ Bob	[$\mathrm{E}_{K_{bc}}(\mathrm{T}_c$,Alice,$\mathrm{K}_{ab})$]

Vale ressaltar que o protocolo original possui diversos problemas, como:

(p_1) requer um relógio global, pois utiliza *timestamps* (carimbos temporais);

(p_2) Charles (ou servidor intermediário) possui acesso a todas as chaves compartilhadas entre Alice e Bob;

(p_3) o valor da chave de sessão $\mathtt{K}_{ab}$ é inteiramente definido pela Alice, que precisa ser competente o suficiente para gerar chaves de alta qualidade (*e.g.*, evitar problemas de entropia ou baixa qualidade na geração das chaves através de construções e implementações robustas como a solução proposta em [Kreutz et al. 2019]);

(p_4) um agente malicioso pode re-encaminhar as mensagens que estiverem dentro do período de tempo de validade do *timestamp*, resultando em ataques de *replay*;

(p_5) Alice não assume que Bob existe (e não há mensagens de verificação entre ambos);

(p_6) o protocolo é com estado (*stateful*), o que é tipicamente indesejado por que requer mais funcionalidades e capacidades do servidor (ou Charles, no caso). Por exemplo, análogo a um servidor de e-mails, Charles precisa lidar com situações de indisponibilidade de Bob.

3.4.2. Verificação automática do protocolo

O Algoritmo 3.3 descreve o Protocolo 3.3 (WMF) na semântica da Scyther. Na linha 1 é declarado um novo tipo de termo, denominado `TimeStamp`. O protocolo WMF é iniciado na linha 4, tendo como agentes Alice, Bob e Charles.

Algoritmo 3.3: WMF na semântica da Scyther

```
1  usertype SessionKey;
2  usertype TimeStamp;
3  const Fresh: Function;
4  protocol WMF(Alice,Bob,Charles){
5     role Alice{
6        fresh Kab: SessionKey;
7        fresh Ta: TimeStamp;
8        send_1(Alice,Charles,Alice,{Ta,Bob,Kab}k(Alice,Charles));
9        claim_ Alice1(Alice,Secret,Kab);
10       claim_ Alice2(Alice,Empty,(Fresh,Kab));
11    }
12    role Bob{
13       var Tc: TimeStamp;
14       var Kab: SessionKey;
15       recv_2(Charles,Bob,{Tc,Alice,Kab}k(Bob,Charles));
16       claim_Bob1(Bob,Secret,Kab);
17       claim_Bob2(Bob,Nisynch);
18       claim_Bob3(Bob,Empty,(Fresh,Kab));
19    }
20    role Charles{
21       var Kab: SessionKey;
22       var Ta: TimeStamp;
23       fresh Tc:TimeStamp;
24       recv_1( Alice,Charles,Alice,{Ta,Bob,Kab}k(Alice,Charles));
25       send_2(Charles,Bob,{Tc,Alice,Kab}k(Bob,Charles));
26    }
27 }
```

No papel (**role**) Alice (linha 5) são criadas e inicializadas com valores pseudo-aleatórios as variáveis `Kab`, que representa a nova chave de sessão que será compartilhada entre Alice e Bob (linha 6), e `Ta`, que representa o *timestamp* (do inglês, carimbo de tempo) (linha 7). Então, Alice envia para Charles (linha 8) sua identificação (`Alice`), o *timestamp* `Ta`, a identificação do recebedor (`Bob`) e `Kab`. É importante observar que a identificação da Alice é enviada fora e dentro (*i.e.*, ...,`Alice`,{`Alice`,...) do bloco cifrado ({...}`k(Alice,Bob)`) da mensagem. A verificação de `Kab` é realizada nas afirmações **claim** das linhas 9 e 10. Nas linhas 10 e 18, é verificado se a nova chave de

sessão é "fresca" (`Fresh`). O termo `Empty` indica vazio, uma vez que é a entidade (*i.e.*, Alice ou Bob) verificando a cópia local da chave K_{ab}.

No papel (**role**) Bob (linha 12) são criadas as variáveis `Tc` (linha 13), que armazenará o *timetamp* de Charles, e `Kab` (linha 14), que armazenará a chave de sessão recebida de Charles. Bob recebe a mensagem de Charles (linha 15) contendo a sua identificação (`Charles`), o *timestamp* `Tc`, a identificação `Alice` e a chave de sessão `Kab`. A verificação da chave de sessão `Kab` e da segurança da comunicação são realizadas nas afirmações **claim** das linhas 16, 17 e 18.

No papel do agente confiável Charles (**role** definido na linha 20) são criadas as variáveis `Kab` (linha 21), que irá receber a chave de sessão de Alice, o *timestamp* `Ta` (linha 22), que irá armazenar o *timestamp* recebido de Alice, e um *timestamp* `Tc` (linha 23), que irá armazenar o *timestamp* gerado por Charles. Charles recebe a mensagem de Alice (linha 24) contendo o identificador dela (`Alice`), o identificador do destinatário (`Bob`) e a chave de sessão `Kab`. A chave de sessão recebida é encaminhada por Charles para Bob, incluindo um novo *timestamp* `Tc` (linha 25).

Ao verificar automaticamente o WMF com a ferramenta Scyther, podemos visualizar algumas falhas na protocolo, como pode ser observado na Verificação 3.3. Um dos principais problema do protocolo é o fato de Alice não saber se Bob recebeu a nova chave secreta K_{ab}. Uma solução para o problema é apresentada no Protocolo 3.4.

A principal correção do Protocolo 3.3, denominada WMF-Lowe [Lowe 1997a, Lowe 1997b], pode ser vista no Protocolo 3.4. Resumidamente, são acrescentadas duas novas mensagens entre Alice e Bob. Bob envia um *nonce* `nonceB` para Alice (linha 3) e Alice responde (linha 4) para Bob com o sucessor do *nonce* (*e.g.*, `nonceB + 1`). Esse mecanismo evita que Bob pense que Alice tenha estabelecido duas sessões de comunicação – um ataque possível na versão original do WMF. Esta solução evita ataques como os registrados pelos eventos `[WMF,Alice5]`, `[WMF,Alice6]`, `[WMF,Bob6]`, `[WMF,Bob7]`, `[WMF,Bob8]`, `[WMF,Bob9]`, entre Alice e Bob, na Verificação 3.3.

Protocolo 3.4: Especificação do WMF-Lowe entre Alice e Bob

1. Alice → Charles	[Alice, $E_{K_{ac}}$(T$_a$,Bob,K$_{ab}$)]
2. Charles → Bob	[$E_{K_{bc}}$(T$_c$,Alice,K$_{ab}$)]
3. Bob → Alice	[$E_{K_{ab}}$(nonceB)]
4. Alice → Bob	[$E_{K_{ab}}$(nonceB+1)]

3.5. O protocolo Needham-Schroeder (NS)

3.5.1. Método de Autenticação para dois Participantes

O protocolo NS fornece um método de autenticação para dois participantes em uma rede insegura utilizando criptografia assimétrica [Needham and Schroeder 1978]. Entretanto, o protocolo original apresenta falhas de segurança, as quais são exploradas e corrigidas em [Lowe 1995a], culminando no protocolo *Needham-Schroeder-Lowe* (NSL). Nesta

```
./scyther.py --auto-claims --all-attacks protocolo_wmf.spdl

Verification results:
claim id [WMF,Alice3], Secret(Ta)     : No attacks.
claim id [WMF,Alice4], Secret(Kab)    : No attacks within bounds.
claim id [WMF,Alice5], Alive          : Exactly 1 attack.
claim id [WMF,Alice6], Weakagree      : Exactly 1 attack.
claim id [WMF,Alice7], Niagree        : No attacks.
claim id [WMF,Alice8], Nisynch        : No attacks.
claim id [WMF,Bob4], Secret(Kab)      : No attacks within bounds.
claim id [WMF,Bob5], Secret(Tc)       : No attacks within bounds.
claim id [WMF,Bob6], Alive            : Exactly 1 attack.
claim id [WMF,Bob7], Weakagree        : Exactly 1 attack.
claim id [WMF,Bob8], Niagree          : At least 3 attacks.
claim id [WMF,Bob9], Nisynch          : At least 3 attacks.
claim id [WMF,Charles1], Secret(Tc)   : No attacks within bounds.
claim id [WMF,Charles2], Secret(Ta)   : No attacks within bounds.
claim id [WMF,Charles3], Secret(Kab)  : No attacks within bounds.
claim id [WMF,Charles4], Alive        : Exactly 1 attack.
claim id [WMF,Charles5], Weakagree    : Exactly 1 attack.
claim id [WMF,Charles6], Niagree      : At least 3 attacks.
claim id [WMF,Charles7], Nisynch      : At least 3 attacks.
```

Verificação 3.3: Execução e saída da Scyther para o Algoritmo 3.3

seção, demonstraremos o processo de detecção e avaliação da correção dessas falhas por meio da ferramenta Scyther.

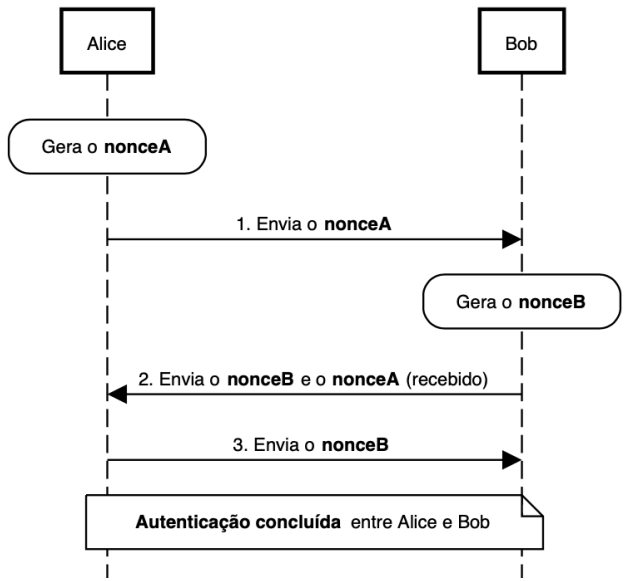


Figura 3.4: Alice e Bob utilizando o protocolo NS

A Figura 3.4 e o Protocolo 3.5 apresentam, respectivamente, o diagrama e a

especificação da comunicação entre Alice e Bob utilizando o NS. É assumido que cada usuário possui um par de chaves privada e pública. Alice deseja comunicar-se com Bob e, para tanto, requisita ao servidor de chaves a chave pública do Bob. Na sequência, Alice gera e envia para Bob um `nonceA` juntamente com sua identificação. A mensagem é cifrada com a chave pública do Bob. Ao receber a mensagem, Bob decodifica-a e recupera o `nonceA` e a identificação da remetente. Bob lê a identificação de Alice e requisita a respectiva chave pública para o servidor de chaves. No próximo passo, Bob gera e envia para Alice um novo `nonceB`, cifrado com a chave pública da receptora. Ao receber a mensagem, Alice verifica que o `nonceA` recebido é o mesmo que foi enviado para Bob anteriormente. Finalmente, Alice envia para Bob o `nonceB`. Ao receber e verificar os respectivos *nonces*, utilizando as respectivas chaves privadas, Alice e Bob são capazes de confirmar a autenticidade das mensagens.

Protocolo 3.5: Especificação do NS entre Alice e Bob

1. Alice $\rightarrow$ Bob	$[E_{pk_{Bob}}(\text{Alice, nonceA})]$
2. Bob $\rightarrow$ Alice	$[E_{pk_{Alice}}(\text{nonceA, nonceB})]$
3. Alice $\rightarrow$ Bob	$[E_{pk_{Bob}}(\text{nonceB})]$

3.5.2. Verificação automática do protocolo

A Figura 3.5 e o Algoritmo 3.4 apresentam o protocolo NS na semântica da Scyther. As chaves pública `pk` e privada `sk` são declaradas globalmente (linhas 1 e 2), embora sejam atribuídas autonomamente a Alice e Bob pela ferramenta. O comando `inversekeys` (linha 3) determina que as chaves `pk` e `sk` são inversas, ou seja, ao cifrar com a chave pública só é possível decifrar com a chave secreta e vice-versa. Para a execução de ataques ao protocolo, é definido um agente não-confiável Eve (linhas 4 e 5).

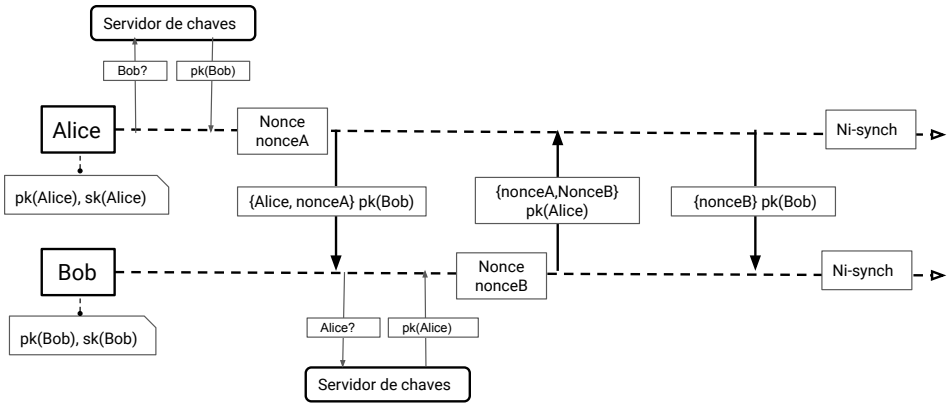


Figura 3.5: Alice e Bob utilizando o protocolo Needham-Schroeder na semântica Scyther

Algoritmo 3.4: NS na semântica da Scyther

```
1  const pk: Function;
2  secret sk: Function;
3  inversekeys (pk,sk);
4  const Eve: Agent;
5  untrusted Eve;
6  protocol NS(Alice,Bob,Eve){
7     role Alice{
8        fresh nonceA: Nonce;
9        var nonceB: Nonce;
10       send_1(Alice,Bob,{Alice,nonceA}pk(Bob));
11       recv_2(Bob,Alice,{nonceA,nonceB}pk(Alice));
12       send_3(Alice,Bob,{nonceB}pk(Bob));
13       claim(Alice,Secret,nonceA);
14       claim(Alice,Secret,nonceB);
15       claim(Alice,Nisynch);
16    }
17    role Bob{
18       var nonceA: Nonce;
19       fresh nonceB: Nonce;
20       recv_1(Alice,Bob,{Alice,nonceA}pk(Bob));
21       send_2(Bob,Alice,{nonceA,nonceB}pk(Alice));
22       recv_3(Alice,Bob,{nonceB}pk(Bob));
23       claim(Bob,Secret,nonceA);
24       claim(Bob,Secret,nonceB);
25       claim(Bob,Nisynch);
26    }
27 }
```

A chamada à função **protocol** (linha 6) marca o início da especificação do protocolo NS, com três agentes (Alice, Bob e Eve), sendo que Alice e Bob possuem papéis (**role**) explícitos. A variável `nonceA` (linha 8), do tipo **Nonce** e antecedida por **fresh**, irá conter um valor pseudo-aleatório. Já a variável `nonceB` (linha 9), do mesmo tipo, mas **var** ao invés de **fresh**, é utilizada para armazenar valores recebidos (*e.g.*, linha 11).

Cada evento de envio (*e.g.*, **send_1**, linha 10) possui um evento de recebimento (*e.g.,* **recv_1**, linha 20) correspondente. A sintaxe do evento **send_1** indica que a transmissão é de Alice para Bob, e que os dados enviados, cifrados com a chave pública de Bob `pk(Bob)`, são `Alice` e `nonceA`. No agente Bob (linha 20), há um evento com sintaxe quase idêntica, cuja única diferença é o tipo (*i.e.*, **recv** ao invés de **send**). Na sequência, Bob envia os valores `nonceA` e `nonceB` para Alice (linha 21). Alice recebe a resposta (linha 11), verifica o valor do `nonceA` e envia `nonceB` para Bob (linha 12), que recebe e verifica o valor (linha 22).

Finalmente, as afirmações **claim** (linhas 13, 14, 15 23, 24, 25) definem os requisitos de segurança do protocolo. No caso, as afirmações criadas verificam se os *nonces* gerados por Alice (`nonceA`) e Bob (`nonceB`) permanecem secretos durante as comunicações (**claim Secret**) e se as mensagens são de fato trocadas entre eles apenas (**claim Nisynch**).

A ferramenta Scyther aponta a existência de falhas no Algoritmo 3.4, como pode ser visto na Verificação 3.4. O relatório da ferramenta indica pelo menos um ataque que afeta o protocolo NS, relacionado aos eventos `[NS,Bob1]`, `[NS,Bob2]` e `[NS,Bob3]`. O ataque está relacionado aos *nonces* `nonceA` e `nonceB` e ao **Nisynch**.

Em casos como esses, a ferramenta gera também um grafo com os passos de execução do(s) ataque(s), como ilustrado na Figura 3.5.

Para comprometer a comunicação entre Alice e Bob, Eve (agente malicioso) precisa apenas convencer Alice de que é Bob. Alice envia a Eve uma mensagem {Alice,nonceA} cifrada com a chave pública pk{Eve}. Como Eve possui a chave privada correspondente, ele consegue ler o conteúdo da mensagem. Eve então cifra e envia a mensagem para Bob. Bob, sem desconfiar, pois recebeu uma mensagem cifrada com sua chave pública, responde ao atacante com a mensagem {nonceA,nonceB}pk{Alice}. O agente malicioso simplesmente encaminha a mensagem para Alice. Para confirmar o recebimento do *nonce* nonceB, Alice responde ao atacante com {nonceB}pk{Eve}. O atacante decifra a mensagem, descobre o valor de nonceB, cifra e envia a mensagem para Bob. Bob confirma o valor de nonceB (autenticação) e acredita que está comunicando-se com Alice. A partir deste ponto, todas as mensagens trocadas entre Alice e Bob passam primeiro pelo atacante. Como pode ser observado, a execução do ataque é simples, porém, eficaz. Basta que Eve conheça a chave pública de Bob e intermedie a comunicação entre Alice e Bob.

```
./scyther.py --all-attacks --max-runs=5 protocolo_ns.spdl

Verification results:
claim id [NS,Alice1], Secret(nonceA)  : No attacks.
claim id [NS,Alice2], Secret(nonceB)  : No attacks.
claim id [NS,Alice3], Nisynch         : No attacks.
claim id [NS,Bob1],   Secret(nonceA)  : Exactly 1 attack.
claim id [NS,Bob2],   Secret(nonceB)  : Exactly 1 attack.
claim id [NS,Bob3],   Nisynch         : Exactly 1 attack.
```

Verificação 3.4: Execução e saída da Scyther para o Algoritmo 3.4

Vale observar que a falha de segurança do NS foi descoberta cerca de 17 anos após sua concepção com ajuda de ferramentas computacionais de análise de protocolo [Lowe 1996]. O protocolo NSL (ou NS-Lowe) corrige a falha como segue [Lowe 1995b, Lowe 1989, Cremers 2006]. Bob deve adicionar a sua identidade na resposta à primeira mensagem de Alice (*i.e.*, *nonceA*,*nonceB*,*Bob*), na linha 21 do Algoritmo 3.4, como ilustrado no fluxograma da Figura 3.6. Com isso, Alice consegue descobrir que a identidade contida na mensagem é diferente da identidade fornecida por Eve, para quem está enviando as suas mensagens. Nesse caso, ao detectar tal inconsistência, Alice simplesmente encerra a troca de mensagens.

Com o algoritmo corrigido (linha 21), o resultado da análise da Scyther pode ser visto na Verificação 3.5. Como pode ser observado, um simples detalhe de especificação pode comprometer toda a segurança do protocolo. Isso demonstra o quão importante é a utilização de ferramentas de verificação automática de protocolos.

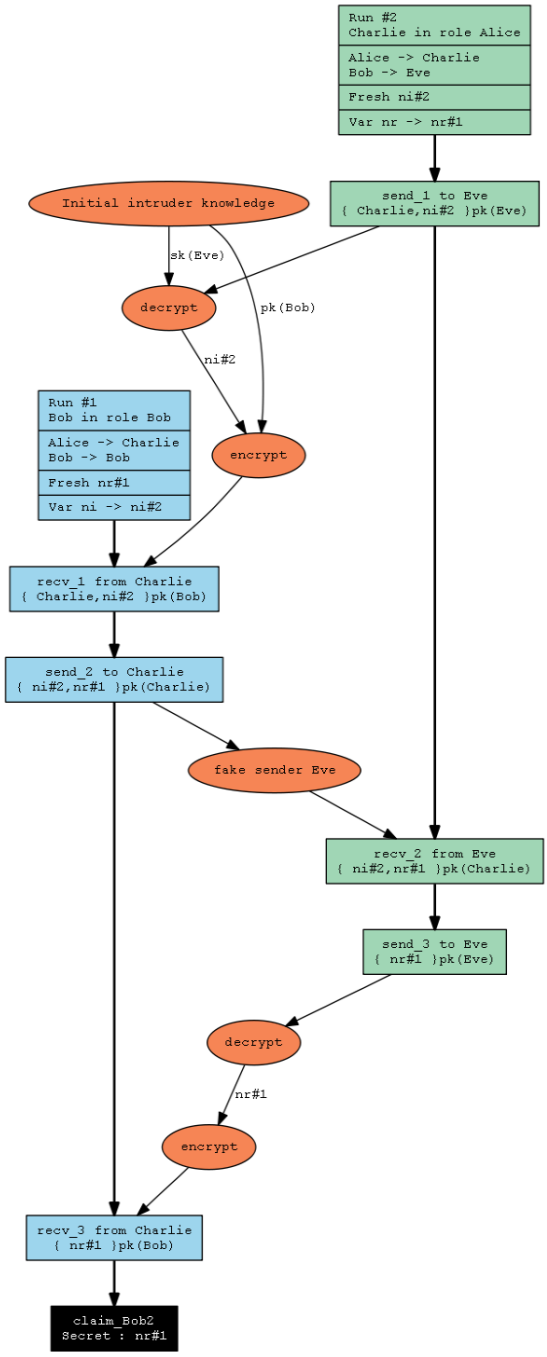


Figura 3.5: Diagrama do ataque ao protocolo NS

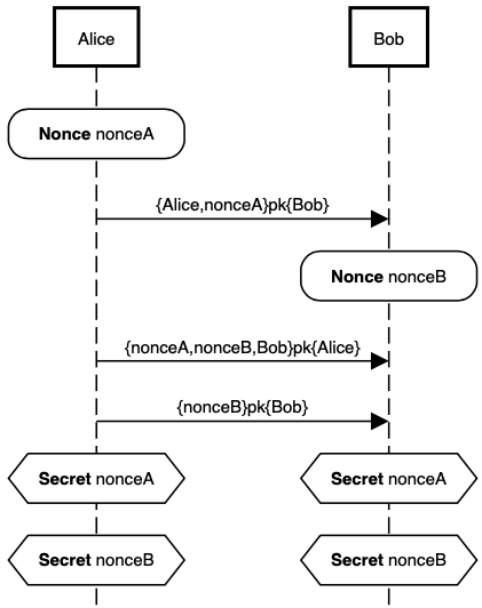


Figura 3.6: Diagrama do protocolo Needham-Schroeder-Lowe (NSL) [Cremers 2006]

```
./scyther.py --all-attacks --max-runs=5 protocolo_ns_corrigido.spdl

Verification results:
claim id [NS,Alice1], Secret(nonceA)  : No attacks.
claim id [NS,Alice2], Secret(nonceB)  : No attacks.
claim id [NS,Alice3], Nisynch         : No attacks.
claim id [NS,Bob1],   Secret(nonceA)  : No attacks.
claim id [NS,Bob2],   Secret(nonceB)  : No attacks.
claim id [NS,Bob3],   Nisynch         : No attacks.
```

Verificação 3.5: Execução e saída da Scyther para o Algoritmo 3.4 (corridigo)

3.6. O protocolo Generalized NSL public-key protocol (β)

3.6.1. Família de Protocolos para Autenticação entre Múltiplas Entidades

Em sistemas modernos é comum a necessidade de autenticação entre múltiplas (p) entidades, como no processo de atendimento de um técnico a um cliente, ambos coordenados por um gestor do Provedor de Acesso à Internet [Quincozes et al. 2020] (onde $p = 3$). Uma maneira simples de obter autenticação mútua entre múltiplas entidades (*i.e.*, $p > 2$) é realizar autenticação aos pares usando protocolos de autenticação mútua projetados para duas entidades ($p = 2$). Seguindo esse processo, para realizar autenticação mútua entre p entidades são necessárias $(p * (p-1))/2$ instâncias de cada protocolo de autenticação mútua entre duas entidades, e pelo menos três mensagens para cada instância [Cremers and Mauw 2006].

Para minimizar o número de mensagens trocadas durante o processo de autenticação entre múltiplas entidades, foi proposto um arcabouço de protoco-

los [Cremers and Mauw 2006]. Esse arcabouço permite generalizar protocolos de autenticação mútua entre duas entidades ($p = 2$) para múltiplas entidades ($p > 2$) que desejam autenticar entre si. Um protocolo generalizado através do arcabouço permite autenticação mútua entre p entidades trocando $2p-1$ mensagens. Assim, por exemplo, para $p = 4$, a quantidade de mensagens a serem trocadas usando um protocolo NSL $(4*(4-1))/2*3 = 18$ é reduzida para $(2*4)-1 = 7$ usando protocolo NSL generalizado para quatro entidades.

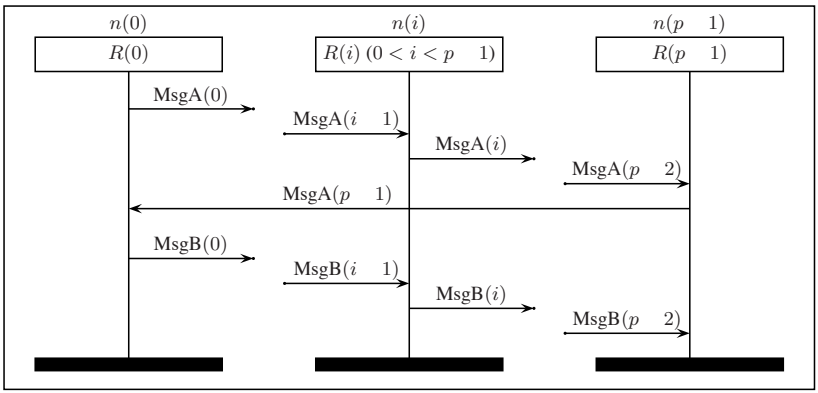


Figura 3.6: Estrutura de comunicação de múltiplas entidades [Cremers and Mauw 2006]

A Figura 3.6 apresenta uma visão geral do arcabouço de generalização. O arcabouço define uma coleção de p entidades $R_0, \cdots, R_{p-1}$ [Cremers and Mauw 2006]. Cada entidade controla uma mensagem **nounce** $n_0, \cdots, n_{p-1}$. A primeira entidade (R_0) encaminha o desafio para a segunda entidade (R_1), que, por sua vez, encaminha para a terceira entidade e assim sucessivamente até a última entidade (R_{p-1}) do sistema.

A primeira entidade (R_0) resolve seu desafio quando recebe a resposta da última entidade (R_{p-1}), confirmando que todas as outras entidades responderam ao seu desafio. A primeira entidade então encaminha as respostas restantes para a segunda entidade (R_1) encerrar o ciclo dela. A segunda entidade, então, encerra seu ciclo e encaminha as respostas restantes para a entidade seguinte. Esse processo se repete até a última entidade (R_{p-1}) encerrar seu próprio ciclo. Observe-se que as primeiras p mensagens (denotadas como MsgA) contém tanto desafios como respostas, e que as últimas $p-1$ mensagens (do tipo MsgB) contém apenas respostas dos desafios.

Há uma família de protocolos para demonstrar a aplicabilidade do arcabouço proposto [Cremers and Mauw 2006]. Cada membro da família corresponde a uma generalização para múltiplas entidades de um protocolo de autenticação mútua entre duas entidades, incluindo Needham-Schroeder (NS), Needham-Schroeder-Lowe (NSL) e Bilateral Key Exchange (BKE).

3.6.2. Protocolo NSL Generalizado para Quatro Entidades

Por razões didáticas, vamos focar em apenas uma instância de um protocolo generalizado da referida família de protocolos. Especificamente, nesta seção vamos apresentar a generalização do NSL (*Generalized Needham-Schroeder-Lowe*) com chave pública, denotado como protocolo β da família, e considerando quatro entidades (*i.e.*, $p = 4$).

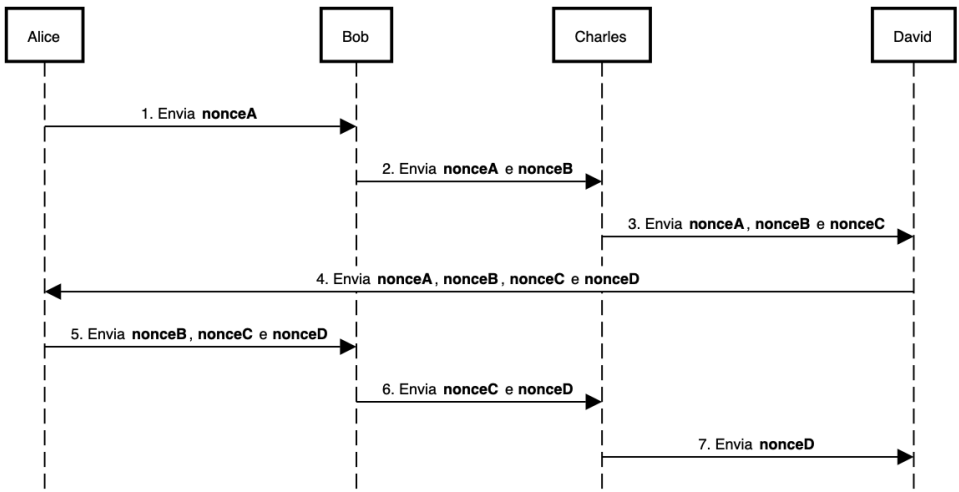


Figura 3.7: Diagrama de Alice, Bob, Charles e David utilizando o protocolo NSL generalizado

A Figura 3.7 e o Protocolo 3.6 ilustram e detalham o funcionamento do protocolo *Four-party Generalized NSL*, respectivamente. No primeiro ciclo, Alice envia uma mensagem contendo um desafio e a identificação das outras entidades para Bob cifrada com a chave pública do receptor (linha 1). Por sua vez, Bob decifra a mensagem com a sua chave privada e envia uma nova mensagem para Charles cifrada com a chave pública do receptor (linha 2). Essa mensagem contém, além do desafio de Alice, o desafio do Bob, e a identificação das outras entidades. Charles repete o processo parecido em uma comunicação com David (linha 3). O primeiro ciclo se encerra quando David envia uma mensagem para Alice contendo os desafios de todas entidades cifrada com a chave privada da receptora (linha 4).

É oportuno lembrar que o segundo ciclo permite encerrar o processo de desafio. Nesse sentido, Alice confere o seu desafio e envia para Bob uma mensagem contendo os desafios restantes (linha 5). Processos similares se repetem entre Bob e Charles (linha 6) e, por fim, entre Charles e David (linha 7).

3.6.3. Verificação automática do protocolo

O Algoritmo 3.5 apresenta uma instância do protocolo Generalized Four-Party Needham-Schroeder-Lowe na semântica da Scyther. Observe-se que há quatro entidades no protocolo. O primeiro ciclo de mensagem foi denotado com sufixo _11, _12, _13, _14 e o segundo ciclo com sufixo _21, _22 e _23.

Protocolo 3.6: Especificação do NS entre Alice e Bob

1. Alice → Bob	[$E_{pk_{Bob}}$(Alice,Charles,David,nonceA)]
2. Bob → Charles	[$E_{pk_{Charles}}$(Alice,Bob,David,nonceA,nonceB)]
3. Charles → David	[$E_{pk_{David}}$(Alice,Bob,Charles,nonceA,nonceB,nonceC)]
4. David → Alice	[$E_{pk_{Alice}}$(Bob,Charles,David,nonceA,nonceB,nonceC,nonceD)]
5. Alice → Bob	[$E_{pk_{Bob}}$(nonceB,nonceC,nonceD)]
6. Bob → Charles	[$E_{pk_{Charles}}$(nonceC,nonceD)]
7. Charles → David	[$E_{pk_{David}}$(nonceD)]

Algoritmo 3.5: *Four-party Generalized NSL* na semântica da Scyther

```
1  protocol 4GNSL(Alice, Bob, Charles, David){
2    role Alice {
3      fresh nounceA: Nonce;
4      var nounceB, nounceC, nounceD: Nonce;
5      send_11(Alice,Bob, {nounceA, Alice, Charles, David}pk(Bob));
6      recv_14(David,Alice, {nounceA, nounceB, nounceC, nounceD, Bob,Charles,David}pk(Alice));
7      send_21(Alice, Bob, {nounceB, nounceC, nounceD}pk(Bob));
8    }
9    role Bob {
10     fresh nounceB: Nonce;
11     var nounceA, nounceC, nounceD: Nonce;
12     recv_11(Alice,Bob, {nounceA, Alice, Charles, David}pk(Bob));
13     send_12(Bob,Charles, {nounceA, nounceB, Alice, Bob, David}pk(Charles));
14     recv_21(Alice, Bob, {nounceB, nounceC, nounceD}pk(Bob));
15     send_22(Bob, Charles, {nounceC, nounceD}pk(Charles));
16   }
17   role Charles {
18     fresh nounceC: Nonce;
19     var nounceA, nounceB, nounceD: Nonce;
20     recv_12(Bob,Charles, {nounceA, nounceB, Alice, Bob, David}pk(Charles));
21     send_13(Charles, David, {nounceA, nounceB, nounceC, Alice, Bob, Charles}pk(David));
22     recv_22(Bob, Charles, {nounceC, nounceD}pk(Charles));
23     send_23(Charles, David, {nounceD}pk(David));
24   }
25   role David {
26     fresh nounceD: Nonce;
27     var nounceA, nounceB, nounceC: Nonce;
28     recv_13(Charles, David, {nounceA, nounceB, nounceC, Alice, Bob, Charles}pk(David));
29     send_14(David, Alice, {nounceA, nounceB, nounceC, nounceD, Bob, Charles, David}pk(Alice));
30     recv_23(Charles, David, {nounceD}pk(David));
31   }
32 }
```

A Verificação 3.6 apresenta o resultado da avaliação do protocolo acima usando a ferramenta Scyther com parâmetros `--auto-claims` e `--max-runs=7`. Observe-se que é possível obter prova de corretude para alguns claims gerado pelo Scyther enquanto para outros claims não foram detectados ataques dentro do número de rodadas empregrado.

```
./scyther.py --auto-claims --all-attacks --max-runs=7 protocolo_4gnsl.spdl

Verification results:
claim id [4GNSL,Alice1], Secret(nonceA)   : Proof of correctness
claim id [4GNSL,Alice2], Secret(nonceD)   : No attacks within bounds.
claim id [4GNSL,Alice3], Secret(nonceC)   : No attacks within bounds.
claim id [4GNSL,Alice4], Secret(nonceB)   : Proof of correctness.
claim id [4GNSL,Alice5], Alive            : Proof of correctness.
claim id [4GNSL,Alice6], Weakagree        : Proof of correctness.
claim id [4GNSL,Alice7], Niagree          : Proof of correctness.
claim id [4GNSL,Alice8], Nisynch          : Proof of correctness.
claim id [4GNSL,Bob1], Secret(nonceB)     : No attacks within bounds.
claim id [4GNSL,Bob2], Secret(nonceD)     : No attacks within bounds.
claim id [4GNSL,Bob3], Secret(nonceC)     : No attacks within bounds.
claim id [4GNSL,Bob4], Secret(nonceA)     : No attacks within bounds.
claim id [4GNSL,Bob5], Alive              : No attacks within bounds.
claim id [4GNSL,Bob6], Weakagree          : No attacks within bounds.
claim id [4GNSL,Bob7], Niagree            : No attacks within bounds.
claim id [4GNSL,Bob8], Nisynch            : No attacks within bounds.
claim id [4GNSL,Charles1], Secret(nonceC) : No attacks within bounds.
claim id [4GNSL,Charles2], Secret(nonceD) : No attacks within bounds.
claim id [4GNSL,Charles3], Secret(nonceB) : No attacks within bounds.
claim id [4GNSL,Charles4], Secret(nonceA) : No attacks within bounds.
claim id [4GNSL,Charles5], Alive          : No attacks within bounds.
claim id [4GNSL,Charles6], Weakagree      : No attacks within bounds.
claim id [4GNSL,Charles7], Niagree        : No attacks within bounds.
claim id [4GNSL,Charles8], Nisynch        : No attacks within bounds.
claim id [4GNSL,David1], Secret(nonceD)   : No attacks within bounds.
claim id [4GNSL,David2], Secret(nonceC)   : No attacks within bounds.
claim id [4GNSL,David3], Secret(nonceB)   : No attacks within bounds.
claim id [4GNSL,David4], Secret(nonceA)   : No attacks within bounds.
claim id [4GNSL,David5], Alive            : No attacks within bounds.
claim id [4GNSL,David6], Weakagree        : No attacks within bounds.
claim id [4GNSL,David7], Niagree          : No attacks within bounds.
claim id [4GNSL,David8], Nisynch          : No attacks within bounds.
```

Verificação 3.6: Execução e saída da Scyther para o Algoritmo 3.5

3.7. Considerações Finais

Neste minicurso, nós apresentamos uma introdução às semânticas operacionais e a utilização prática da ferramenta de verificação automática de protocolos de segurança Scyther (Seção 3.2). Para o desenvolvimento do minicurso, foram selecionados seguintes protocolos: o Atualização de Chave Simétrica (ACS), na Seção 3.3; o Wide Mouth Frog (WMF), na Seção 3.4; o Needham-Schroeder (NS), na Seção 3.5; e o Generalized NSL public-key protocol (β), na Seção 3.6. Os protocolos foram apresentados de maneira didática, incluindo representação através de fluxogramas simplificados e uma semântica genérica de especificação de protocolos de segurança, que é independente das semânticas operacionais e linguagens das ferramentas de verificação.

Para cada um dos protocolos utilizados como exemplo, foi detalhado o algoritmo de verificação automática na semântica operacional da ferramenta Scyther. As verificações automáticas (ver Verificações 3.1, 3.2, 3.3, 3.4, 3.5, 3.6) demonstraram falhas de protocolos como o WMF e o NS, que foram discutidas e corrigidas através da modificação de mensagens ou da introdução de versões atualizadas do protocolo, como o

WMF-Lowe (ver Protocolo 3.4 na Seção 3.4).

Acreditamos que o minicurso tenha cumprido o seu objetivo principal, isto é, fazer com que estudantes e profissionais percebam a importância de ferramentas de verificação automática e formal de protocolos de segurança. Por exemplo, como comentado anteriormente na Seção 3.5, a falha do NS foi descoberta apenas 17 anos após sua concepção do protocolo. Outros exemplos práticos similares, recentes, podem ser encontrados na literatura.

Agradecimentos

Agradecemos ao editor Gustavo Griebler, da Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação (ReABTIC)[12], pela autorização expressa de re-utilização integral do conteúdo do artigo "Verificação Automática dos Protocolos de Segurança Needham-Schroeder, WMF e CSA com a ferramenta Scyther" [Jenuario et al. 2020], de autoria de co-autores deste minicurso. As Seções 3.4 e 3.5 deste minicurso, apesar de apresentarem novos dados e conteúdos, foram baseadas no conteúdo original do artigo.

Referências

[Affeldt and Marti 2013] Affeldt, R. and Marti, N. (2013). Towards Formal Verification of TLS Network Packet Processing Written in C. In *7th PLPV*, pages 35–46. ACM.

[Arapinis et al. 2012] Arapinis, M., Mancini, L., Ritter, E., Ryan, M., Golde, N., Redon, K., and Borgaonkar, R. (2012). New privacy issues in mobile telephony: fix and verification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 205–216.

[Armando, A., et al. 2005] Armando, A., et al. (2005). The AVISPA tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer.

[Atkinson 1995] Atkinson, R. (1995). Security Architecture for the Internet Protocol. RFC 1825.

[Baelde et al. 2017] Baelde, D., Delaune, S., Gazeau, I., and Kremer, S. (2017). Symbolic verification of privacy-type properties for security protocols with xor. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 234–248. IEEE.

[Bai et al. 2018] Bai, X., Cheng, Z., Duan, Z., and Hu, K. (2018). Formal modeling and verification of smart contracts. In *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, ICSCA 2018, page 322–326, New York, NY, USA. Association for Computing Machinery.

[Blanchet 2006] Blanchet, B. (2006). A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, pages 140–154, Oakland, California.

[Blanchet et al. 2018] Blanchet, B., Smyth, B., Cheval, V., and Sylvestre, M. (2018). ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf.

[12] https://revistas.setrem.com.br/index.php/reabtic

[Burrows 1989] Burrows, M. (1989). Wide mouthed frog. http://www.lsv.fr/Software/spore/wideMouthedFrog.html.

[Burrows et al. 1990] Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM TRANSACTIONS ON COMPUTER SYSTEMS*, 8:18–36.

[Carrel and Harkins 1998] Carrel, D. and Harkins, D. (1998). The Internet Key Exchange (IKE). RFC 2409.

[Chen et al. 2016] Chen, S., Fu, H., and Miao, H. (2016). Formal verification of security protocols using spin. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–6. IEEE.

[Chudnov et al. 2018] Chudnov, A., Collins, N., Cook, B., Dodds, J., Huffman, B., MacCárthaigh, C., Magill, S., Mertens, E., Mullen, E., Tasiran, S., Tomb, A., and Westbrook, E. (2018). Continuous Formal Verification of Amazon s2n. In *Computer Aided Verification*, pages 430–446.

[Cohn-Gordon et al. 2020] Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., and Stebila, D. (2020). A formal security analysis of the signal messaging protocol. *Journal of Cryptology*, 33(4):1914–1983.

[Cook 2018] Cook, B. (2018). Formal reasoning about the security of amazon web services. In *International Conference on Computer Aided Verification*, pages 38–47. Springer.

[Cremers et al. 2016] Cremers, C., Horvat, M., Scott, S., and v. d. Merwe, T. (2016). Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *IEEE SP*.

[Cremers and Mauw 2006] Cremers, C. and Mauw, S. (2006). A family of multi-party authentication protocols. In *First Benelux Workshop on Information and System Security (WISSec)*.

[Cremers 2008] Cremers, C. J. (2008). The scyther tool: Verification, falsification, and analysis of security protocols. In *International conference on computer aided verification*, pages 414–418. Springer.

[Cremers 2006] Cremers, C. J. F. (2006). *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology Eindhoven. https://doi.org/10.6100/IR614943.

[Dalal et al. 2010] Dalal, N., Shah, J., Hisaria, K., and Jinwala, D. (2010). A comparative analysis of tools for verification of security protocols. *Int. J. of Comm., Network and System Sciences*, 3(10):779.

[Delia Jurcut et al. 2018] Delia Jurcut, A., Liyanage, M., Chen, J., Gyorodi, C., and He, J. (2018). On the security verification of a short message service protocol. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6.

[Jenuario et al. 2020] Jenuario, T., Chervinski, J. O., Paz, G., Fernandes, R., Beltran, R., and Kreutz, D. (2020). Verificação Automática dos Protocolos de Segurança Needham-Schroeder, WMF e CSA com a ferramenta Scyther. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação (ReABTIC)*, pages 1–11. http://dx.doi.org/10.5281/zenodo.3833260.

[Kelsey et al. 1997] Kelsey, J., Schneier, B., and Wagner, D. (1997). Protocol interactions and the chosen protocol attack. In *International Workshop on Security Protocols*, pages 91–104. Springer.

[Krawczyk et al. 1997] Krawczyk, D. H., Bellare, M., and Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. RFC 2104.

[Kreutz et al. 2019] Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2019). ANCHOR: Logically centralized security for software-defined networks. *ACM Trans. Priv. Secur.*, 22(2):8:1–8:36.

[Li et al. 2018] Li, L., Sun, J., Liu, Y., Sun, M., and Dong, J. (2018). A Formal Specification and Verification Framework for Timed Security Protocols. *IEEE Trans. on Soft. Engineering*, 44(8):725–746.

[Liu and Liu 2019] Liu, J. and Liu, Z. (2019). A survey on security verification of blockchain smart contracts. *IEEE Access*, 7:77894–77904.

[Lonvick and Ylonen 2006] Lonvick, C. M. and Ylonen, T. (2006). The Secure Shell (SSH) Transport Layer Protocol. RFC 4253.

[Lowe 1989] Lowe, G. (1989). Lowe's fixed version of needham-schroder public key. http://www.lsv.fr/Software/spore/nspkLowe.html.

[Lowe 1995a] Lowe, G. (1995a). An Attack on the Needham- Schroeder Public- Key Authentication Protocol. *Information processing letters*, 56(3).

[Lowe 1995b] Lowe, G. (1995b). An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133.

[Lowe 1996] Lowe, G. (1996). Breaking and fixing the needham-schroeder public-key protocol using fdr. In Margaria, T. and Steffen, B., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Lowe 1997a] Lowe, G. (1997a). A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester. http://www.cs.ox.ac.uk/gavin.lowe/Security/Papers/multiplicityTR.ps.

[Lowe 1997b] Lowe, G. (1997b). Lowe modified wide mouthed frog. http://www.lsv.fr/Software/spore/wideMouthedFrogLowe.html.

[Meier et al. 2013] Meier, S., Schmidt, B., Cremers, C., and Basin, D. (2013). The tamarin prover for the symbolic analysis of security protocols. In *International Conference on Computer Aided Verification*, pages 696–701. Springer.

[Needham and Schroeder 1978] Needham, R. M. and Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999.

[Quincozes et al. 2020] Quincozes, V. E., Temp, D., Quincozes, S. E., Kreutz, D., and Mansilha, R. B. (2020). Sistema para Autenticação entre Clientes, Técnicos e ISPs. In *5o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*, Joinville-SC, Brasil.

[Rainer et al. 2020] Rainer, D., Matthias, P., and Helmut, K. (2020). A comprehensive model of information security factors for decision-makers. *Computers & Security*, 92:101747.

[Rescorla 2018] Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446.

[Steiner et al. 1996] Steiner, M., Tsudik, G., and Waidner, M. (1996). Diffie-hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 31–37.

[Velibor et al. 2016] Velibor, S., Ivana, O., and Ramo, S. (2016). Comparative analysis of some cryptographic systems. Technical Report 15, Business Information Security Conference.