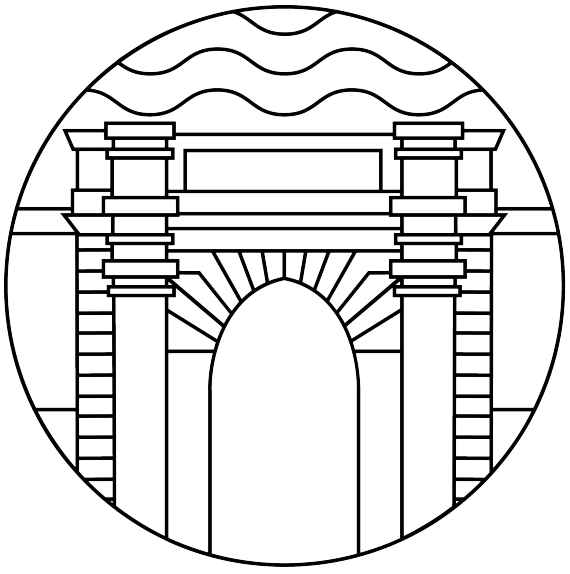
XXI SBSeg 2021
Belém - PA

MINICURSOS

XXI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2021)

04 A 07 DE OUTUBRO DE 2021 • EVENTO ONLINE



XXI SBSeg 2021
Belém - PA

Minicursos do XXI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais

Editora
Sociedade Brasileira de Computação – SBC

Organizadores
Altair Olivo Santin, PUC-PR
Roberto Samarone dos Santos Araujo, UFPA
Antônio Jorge Gomes Abelém, UFPA

Realização
Sociedade Brasileira de Computação – SBC Universidade Federal do Pará – UFPA

Promoção
Sociedade Brasileira de Computação – SBC

Dados Internacionais de Catalogação na Publicação (CIP)

S612 Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (21. : 2021 : Belém, PA)

Minicursos do XXI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais [recurso eletrônico] – organizadores: Altair Olivo Santin, Roberto Samarone dos Santos Araujo, Antônio Jorge Gomes Abelém – Porto Alegre : SBC, 2021.

ISBN 978-65-87003-65-8

1. Segurança da informação. 2. Sistemas computacionais. I. Santin, Altair Olivo. II. Araujo, Roberto Samarone dos Santos. III. belém, Antônio Jorge Gomes. IV. Título.

CDU 004

Catalogação elaborada por Francine Conde Cabral
CRB-10/2606

Capa (Foto):
Roberto Samarone dos Santos Araujo, UFPA

Capa (Edição):
Andreza Jackson de Vasconcelos, NITAE-UFPA
Karina Cristina Martins de Souza, UFPA

Editoração:
Adiel Dos Santos Nascimento, UFPA
Antônio Jorge Gomes Abelém, UFPA
Roberto Samarone dos Santos Araujo, UFPA

Sociedade Brasileira de Computação – SBC

Contato

Av. Bento Gonçalves, 9500. Setor 4, Prédio 43.412, Sala 219. Bairro Agronomia.

Caixa Postal 15012. CEP 91501-970. Porto Alegre - RS.

CNPJ: 29.532.264/0001-78

https://www.sbc.org.br

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

Coordenação

Michele Nogueira (UFPR), Coordenadora

Igor Moraes (UFF), Vice-Coordenador

Comitê Gestor

Aldri Luiz dos Santos (UFMG) - Convidado

Altair Olivo Santin (PUCPR)

Daniel Macêdo Batista - (USP)

Eduardo Luzeiro Feitosa - (UFAM) - Convidado

Fábio Borges de Oliveira (LNCC)

Luciano Paschoal Gaspary - (UFRGS)

Luis Antonio Brasil Kowada (UFF)

Marco Aurelio Amaral Henriques (UNICAMP)

Roberto Samarone dos Santos Araujo (UFPA)

Mensagem da Coordenação Geral

Organizar a XXI edição do Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg2021) é um privilégio, por poder contribuir com a comunidade de Segurança do Brasil e do exterior, repleta de profissionais competentes e com reconhecido destaque. O SBSeg2021 é o segundo evento da série realizado totalmente on-line, como consequência da pandemia provocada pela COVID-19. Os desafios foram muitos e as incertezas provocaram indefinições que só puderam ser sanadas próximo ao evento. Apesar disso, o SBSeg continua se destacando como o mais importante evento nacional na área de segurança, celeiro para a discussão, troca de conhecimento e apresentação de trabalhos científicos de qualidade.

A programação do SBSeg 2021 está diversificada e discute temas relevantes no cenário nacional e internacional. A contribuição da comunidade científica brasileira foi de fundamental importância para manter a qualidade técnica dos trabalhos e fortalecer a ciência, a tecnologia e a inovação no Brasil. Após um cuidadoso processo de avaliação, foram selecionados 27 artigos completos e 4 artigos curtos (organizados em sessões técnicas), e 12 ferramentas para apresentação durante o Salão de Ferramentas. Além disso, o evento contou com 6 palestras principais proferidas por pesquisadores internacionalmente renomados, 1 painel de discussões e debates, todos sobre temas atuais, 4 minicursos, bem como 5 Workshops, o Hackathon e o Fórum de Segurança Corporativa; nesse último, estabelecendo um diálogo entre a comunidade acadêmica, o setor produtivo e os órgãos do Governo Federal e Estadual encarregados da execução das políticas de segurança dos sistemas e redes governamentais.

O apoio incondicional da SBC, da Coordenação e do Comitê Gestor da Comissão Especial de Segurança da Informação da SBC foram determinantes para o sucesso do evento. A realização do evento também contou com o importante apoio do Comitê Gestor da Internet no Brasil (CGI.br), do CNPq, do Banpará, da Prodepa, da Clavis Segurança da Informação e da Ripple. Nosso especial agradecimento à Universidade Federal do Pará (UFPA) e ao Parque de Ciência e Tecnologia Guamá, pelo indispensável suporte à realização do evento.

Nosso agradecimento também para os competentes e incansáveis coordenadores do comitê do programa (Marco Amaral Henriques/Unicamp e Eduardo Souto/UFAM), ao coordenador dos minicursos (Altair Santin/PUCPR), ao coordenador do Salão de Ferramentas (Emerson Ribeiro de Mello/IFSC), aos coordenadores de palestras e tutoriais (Ricardo Dahab/Unicamp e Daniel Batista/USP), ao Coordenador do Workshop de Trabalhos de Iniciação Científica e de Graduação (Eduardo Feitosa/UFAM), ao coordenador do Hackathon (Jean Martina/UFSC), à coordenadora do Workshop de Gestão de Identidades Digitais (Michelle Wangham/Univali), à

coordenadora do Workshop de Regulação, Avaliação da Conformidade, Certificação e Educação em Cibersegurança (Lucila Bento/Inmetro), aos coordenadores do Workshop de Tecnologia Eleitoral (Paulo Matias/UFSCAR e Jeroen van de Graaf/UFMG), aos coordenadores do Workshop de Forense Computacional (Marjory da Costa/Sheffield Hallam University e Josivaldo Araújo/UFPA) e aos coordenadores do Fórum de Segurança Corporativa (Davison Brocardo/Clavis Segurança da Informação e Rodrigo Quites/UFPA). Agradecemos também o excelente trabalho do comitê de organização local.

Por fim, desejamos a todos um produtivo SBSeg.

Roberto Samarone e Antônio Abelém
Coordenadores Gerais do SBSeg 2021.

Mensagem da Coordenação de Minicursos

Os minicursos do SBSeg têm se adaptado para atender os anseios do público do evento. Isto significa estar alinhado não só às expectativas de quem deseja conteúdos mais práticos, mas também de quem espera algo mais próximo da fronteira do conhecimento na área de cibersegurança. Assim, temos minicursos mais teóricos e mais aplicados nesta edição do SBSeg.

A apresentação dos minicursos desta edição é online, mas neste livro os minicursos aceitos são apresentados como 4 capítulos de livro do SBSeg 2021, que apresento brevemente a seguir.

No cap. 1 "Introdução à criptografia completamente homomórfica com implementação em Sage" os autores mostram como a encriptação completamente homomórfica pode ser construída usando Sage – uma linguagem de programação que facilita a implementação de esquemas criptográficos. Também, é apresentado o código fonte funcional para os principais componentes, permitindo ao estudante fazer experimentação com diferentes parâmetros, ou construir novas aplicações.

No cap. 2 "Segurança e Escalabilidade em Sharding Blockchain" são apresentados alguns modelos de sharding destacando suas particularidades e soluções de segurança. Neste caso, o sharding visa superar a limitação de escalabilidade de transações de uma blockchain tradicional como a Ethereum, por exemplo.

No cap. 3 "Autenticando aplicações nativas da nuvem com identidades SPIFFE" são apresentados o modelo confiança-zero e o padrão SPIFFE, mostrando como são utilizados na autenticação de aplicações nativas da nuvem. É usada uma aplicação distribuída como estudo de caso e, seus microsserviços são atestados (via Kubernetes) de modo a receber identidades SPIFFE que permitirão autenticação mútua de acordo com os princípios de confiança zero.

E no cap. 4 "Segurança em Redes 5G: Oportunidades e Desafios em Detecção de Anomalias e Predição de Tráfego baseadas em Aprendizado de Máquina" é contextualiza a segurança das redes móveis de quinta geração (5G), discutindo técnicas de predição de tráfego e detecção de anomalias. Além das técnicas clássicas são abordadas as baseadas em aprendizado profundo. Também, são apresentados os desafios da próxima geração de redes móveis (6G) e um estudo de caso com exercício prático.

Em nome de CESeg da Sociedade Brasileira de Computação (SBC) gostaríamos de agradecer a todos os autores que submeteram suas propostas de minicursos ao SBSeg 2021. Em especial agradecemos aos autores dos minicursos aceitos que preparam os capítulos deste livro e aos revisores de minicursos que se dedicaram a ler e dar feedback aos autores.

Eu agradeço aos coordenadores gerais, professores Roberto Samarone Araujo e Antônio Abelém por confiar em mim para coordenar os trabalhos de minicursos desta edição do SBSeg. Desejo que todos aproveitem ao máximo os conteúdos deste livro!

Altair Olivo Santin (PUCPR)
Coordenador de Minicursos do SBSeg 2021

Comitês

Comitê de Organização

Coordenação Geral
Roberto Samarone dos Santos Araujo, UFPA
Antônio Jorge Gomes Abelém, UFPA

Coordenação de Minicursos
Altair Olivo Santin, PUC-PR

Comitê de Programa dos Minicursos

Aldri dos Santos (UFMG)
Alysson Bessani (Universidade de Lisboa)
Diogo Mattos (UFF)
Eduardo Feitosa (UFAM)
Eduardo Souto (UFAM)
Eduardo Viegas (PUCPR)
Igor Moraes(UFF)
Jean Martina (UFSC)
Luiz Rust (Inmetro)
Marcos Simplicio (USP)
Raul Ceretta (UFSM)
Ricardo Dahab (Unicamp)
Routo Terada (USP)

Revisores

Aldri dos Santos (UFMG)
Alysson Bessani (Universidade de Lisboa)
Diogo Mattos (UFF)
Eduardo Feitosa (UFAM)
Eduardo Souto (UFAM)
Eduardo Viegas (PUCPR)
Igor Moraes(UFF)
Jean Martina (UFSC)
Luiz Rust (Inmetro)
Marcos Simplicio (USP)
Raul Ceretta (UFSM)
Ricardo Dahab (Unicamp)
Routo Terada (USP)

Sumário

Chapter 1

Introdução à criptografia completamente homomórfica com implementação em Sage

Hilder V. L. Pereira (imec-COSIC, KU Leuven) e Eduardo Morais (Disigma)

Abstract

When new cryptographic schemes are proposed in the literature, their implementation can't be based on previous code, and a common strategy on these situations is to first implement such constructions using mathematical programming languages like Sage*. By doing that we require less effort to obtain a proof of concept implementation, which later can be used as a base for production-level development in a different programming language. This way we can also postpone dealing with some abstract mathematical concepts, which are provided in* Sage*, making it a good language for prototyping cryptographic schemes. In this course we are going to demonstrate how fully homomorphic encryption can be constructed using* Sage*. We will provide working code for the main building blocks, therefore the student can experiment with different parameters, or construct new applications.*

Resumo

Quando novos esquemas criptográficos são propostos na literatura, suas implementações não podem ser baseadas em código existente, e uma estratégia nestas situações é primeiramente implementar tais construções usando linguagens de programação como Sage*. Com isso exigimos menos esforço para obter uma prova de conceito, que pode depois servir de base para a implementação que será colocada em produção, e que pode ser desenvolvida em outra linguagem. Desta forma podemos postergar a solução de detalhes matemáticos abstratos, que estão disponíveis em* Sage*, o que a torna uma boa linguagem para prototipagem de esquemas criptográficos. Neste curso nós demonstraremos como a encriptação completamente homomórfica pode ser construída usando* Sage*. Apresentaremos código-fonte funcional para os principais componentes, logo o estudante pode experimentar com diferentes parâmetros, ou pode construir novas aplicações.*

1.1. Introdução

Com a evolução da internet e a popularização do acesso a conexões de alta velocidade, cada vez mais, aplicações que antes eram executadas localmente têm sido delegadas para terceiros. Por exemplo, em vez de buscar, baixar e armazenar vídeos e músicas para reproduzi-los posteriormente, utilizam-se sites e programas com catálogos online e reprodução por *stream*, como Netflix e Spotify; em vez de armazenar dados localmente e criar rotinas de *backup* por conta própria, utilizam-se serviços de armazenamento como Dropbox e Microsoft OneDrive; provedores de e-mail como Gmail substituíram em grande parte programas que gerenciavam e-mails localmente, como o Outlook e o Mozilla Thunderbird.

Esses serviços remotos são conhecidos vulgarmente como nuvem e são atrativos, principalmente, por conta da praticidade, já que é necessário menos conhecimento técnico e também uma infraestrutura mais simples para usar um serviço em nuvem do que para implementá-lo e gerenciá-lo localmente. Por outro lado, geralmente abre-se mão da privacidade, já que muitos dados dos usuários são enviados aos servidores. Em particular, o fato de uma funcionalidade depender de dados sensíveis pode inviabilizar sua utilização. Um exemplo clássico é o de um hospital que tem uma base de dados com informações sobre exames feitos em pacientes e de um servidor na nuvem que tem um modelo de aprendizagem de máquina, como uma rede neural, capaz de gerar um diagnóstico. O hospital gostaria então de enviar os dados de pacientes à nuvem para obter um diagnóstico que poderia auxiliar seus médicos. No entanto, os dados dos pacientes são sigilosos e não podem ser compartilhados com terceiros. Para proteger a privacidade dos dados, seria possível cifrá-los e enviar ao servidor apenas os criptogramas, porém, com esquemas de criptografia tradicionais, o servidor não seria capaz de executar sua rede neural usando apenas dados cifrados como entrada.

Esse é o problema que criptografia homomórfica resolve: com ela, é possível fazer computação sobre dados cifrados como se estivéssemos computando sobre os dados originais em claro. Além da entrada, o resultado da computação também é cifrado, logo, apenas o usuário que cifrou os dados é capaz de decifrar o resultado. Em um cenário ideal, seria possível, por exemplo, cifrar um texto, enviar o criptograma correspondente ao Google, receber o resultado da busca cifrado e então decifrá-lo. Ou seja, seria possível fazer uma busca na internet sem revelar o que se está buscando nem o que foi encontrado.

Mas, como acontece quase sempre, adicionar uma nova funcionalidade também adiciona custo computacional. Nesse caso, a nova funcionalidade é manter a privacidade dos dados e o custo computacional adicional é enorme tanto em termos de memória quanto de processamento, pois os criptogramas são geralmente centenas ou milhares de vezes maiores que os dados originais e cada operação homomórfica é ordens de magnitude mais lenta que a operação correspondente em texto claro. A despeito desse custo adicional, criptografia homomórfica já é razoavelmente prática para algumas aplicações e há muita pesquisa voltada a usá-la para implementar protocolos que processem dados preservando a privacidade, principalmente envolvendo algoritmos de aprendizagem de máquina, como classificadores. Por exemplo, [BMMP18] projetou uma rede neural homomórfica que recebe imagens de dígitos escritos a mão, cifradas, e devolve um criptograma que cifra um valor entre 0 e 9 correspondendo ao dígito contido na imagem. Essa rede neural

tem uma acurácia de 96% e leva apenas 1,6 segundos para classificar uma imagem. Apesar de a base de dados ser relativamente simples, esses resultados já são impressionantes, tendo em vista a tarefa sendo executada: a rede neural classifica uma imagem sem ter acesso à ela, mas apenas aos criptogramas, que não revelam qualquer informação sobre as mensagens que eles cifram.

Outras linhas de pesquisa atuais relacionadas à criptografia homomórfica focam em melhorar a eficiência, na análise dos problemas criptográficos utilizados, na criptoanálise dos esquemas existentes e em descobrir outras primitivas criptográficas mais avançadas que podem ser construídas a partir da criptografia homomórfica. Em suma, esse é um tópico em voga, que tem gerado muito interesse da comunidade acadêmica, com diversas publicações nas mais importantes conferências de criptografia, e também da indústria, com grandes empresas investindo nessa tecnologia[1]. Inclusive, recentemente iniciou-se o esforço para padronizar cifras homomórficas existentes e definir critérios mínimos de segurança [CCD+17], uma interface comum para facilitar compatibilidade e modularidade [BDH+17], e formas de utilização adequadas a diversos cenários práticos [ACC+17]. O processo de padronização deve durar aproximadamente 5 anos e trazer maturidade para que esta tecnologia seja amplamente adotada.

1.1.1. Objetivos do curso

O objetivo deste curso é desmistificar a criptografia completamente homomórfica e mostrar que os conceitos principais desse tipo de primitiva criptográfica, mesmo os utilizados nos artigos científicos mais recentes da área, estão ao alcance até mesmo de alunos de graduação.

Utilizaremos uma abordagem prática em que os aspectos teóricos serão apresentados, explicados e então implementados em Sage [Sag20], que é uma linguagem simples, com sintaxe praticamente idêntica à da linguagem Python, mas que tem uma coleção de bibliotecas para as mais diversas áreas da matemática. Poderemos, portanto, dar ênfase às ideias principais por trás dos esquemas criptográficos e nos preocupar pouco com os detalhes técnicos de implementação, todavia, obtendo códigos funcionais conforme avançamos nos tópicos.

Espera-se que, ao final do curso, os alunos possam: analisar e entender esquemas de criptografia completamente homomórfica; ter familiaridade com os dois principais problemas criptográficos utilizados para construir cifras completamente homomórficas; implementar esses esquemas, incluindo o *bootstrapping*; usar esses esquemas em aplicações como computação segura na nuvem.

1.1.2. O que é criptografia (completamente) homomórfica

Dizemos que uma cifra é homomórfica para uma operação $\star$ se dados dois criptogramas $\mathsf{Enc}(m_1)$ e $\mathsf{Enc}(m_2)$, é possível gerar um novo criptograma $\mathsf{Enc}(m_1 \star m_2)$ sem usar a chave secreta. Por exemplo, considerando a cifra assimétrica RSA na sua versão mais básica,

[1]Para citar apenas alguns exemplos: Google (https://github.com/google/fully-homomorphic-encryption), IBM (https://youtu.be/JyK1BnmXQwU), Intel (https://arxiv.org/abs/2103.16400) e Microsoft (https://www.microsoft.com/en-us/research/project/homomorphic-encryption/).

temos $\mathsf{Enc}(m_i) \equiv m_i^e \pmod{n}$ e n é uma informação pública. Então, é fácil ver que

$$\mathsf{Enc}(m_1) \cdot \mathsf{Enc}(m_2) \equiv m_1^e \cdot m_2^e \equiv (m_1 \cdot m_2)^e \pmod{n}.$$

Ou seja, ao multiplicar dois criptogramas, obtém-se o *produto* das mensagens, encriptado sob a mesma chave pública e. Por isso, dizemos que o RSA é homomórfico com relação à multiplicação. Como é possível multiplicar vários criptogramas, pode-se gerar $\mathsf{Enc}(m_1 \cdot ... \cdot m_\ell)$ para qualquer $\ell \in \mathbb{N}$, no entanto, não é possível gerar $\mathsf{Enc}(m_1 + m_2)$, ou seja, o RSA não é homomórfico para a adição. Já outros esquemas criptográficos, como o Paillier [Pai99], são homomórficos para a adição, mas não para a multiplicação, o que significa que é possível gerar criptogramas como $\mathsf{Enc}(m_1 + ... + m_\ell)$ para qualquer $\ell \in \mathbb{N}$.

Vemos que com essas cifras, o conjunto de funções que podem ser avaliadas homomorficamente é bastante limitado: mesmo funções simples como uma média ponderada usam pelo menos duas operações diferentes, isto é, requerem ao menos que se possa calcular $\mathsf{Enc}(m_1 \cdot m_2 + m_3 \cdot m_4)$. desde a criação do RSA, em 1977, construir uma cifra que permitisse avaliar homomorficamente qualquer função computável se mostrou uma tarefa difícil. Apenas em 2005, foi proposta uma cifra homomórfica tanto para a adição quanto para a multiplicação [BGN05], no entanto, apenas um produto era possível, ou seja, as funções que podem ser calculadas homomorficamente ainda eram bem restritas. Foi apenas em 2009, cerca de 30 anos depois da publicação do RSA, que a primeira cifra *completamente* homomórfica foi proposta [Gen09]. Basicamente, com essa cifra, é possível encriptar bits e aplicar portas lógicas como *AND*, *OR*, e *NOT*, denotados respectivamente pelas operações Booleanas $\wedge$, $\vee$ e $\neg$. Ou seja, a partir de $\mathsf{Enc}(m_i)$ e $\mathsf{Enc}(m_j)$, pode-se gerar $\mathsf{Enc}(m_i \wedge m_j)$, $\mathsf{Enc}(m_i \vee m_j)$ e $\mathsf{Enc}(\neg m_i)$. Como qualquer circuito pode ser expresso em termos dessas três portas lógicas, pode-se avaliar qualquer circuito homomorficamente. Finalmente, como qualquer função computável pode ser expressa por um circuito, é possível calcular $\mathsf{Enc}(f(m_1, ..., m_\ell))$ para qualquer função computável f. Por isso, dizemos que essa cifra é completamente homomórfica[2], enquanto RSA, Paillier, e outras cifras que suportam apenas um tipo de operação são chamadas de cifras *parcialmente* homomórfica.

Grosso modo, as cifras completamente homomórficas cifram uma mensagem m da seguinte forma:

- combinam a chave secreta sk com algum valor aletório, obtendo $a \cdot \mathsf{sk}$;
- adicionam um ruído r, que é apenas um valor aletório pequeno, obtendo $a \cdot \mathsf{sk} + r$;
- finalmente, adicionam a mensagem, então o criptograma é da forma $a \cdot \mathsf{sk} + r + m$.

Para decifrar, primeiro usa-se a chave secreta para remover o termo $a \cdot \mathsf{sk}$, então aplica-se alguma técnica de correção de erros elementar para remover o ruído r e recuperar m. No entanto, é importante notar que o deciframento só funciona se o ruído for relativamente pequeno e esse é o fator que dificulta a criação de uma cifra completamente homomórfica, pois cada operação aumenta o ruído. Por exemplo, para calcular $m_1 \wedge m_2 \vee m_3$

[2]O termo original, em inglês, é *fully homomorphic encryption*.

homomorficamente, começa-se com $c_i := \mathsf{Enc}(m_i)$, $i = 1,2,3$, todos com pouco ruído, então calcula-se $c_4 := \mathsf{Enc}(m_1 \wedge m_2)$, que tem mais ruído que os criptogramas originais, e finalmente aplica-se um *OR* homomórfico em c_4 e c_3, obtendo $c_5 := \mathsf{Enc}(m_1 \wedge m_2 \vee m_3)$ com ruído ainda maior que o de c_4. Como há um limite para a quantidade de ruído que um criptograma pode surportar, fica claro que não se pode calcular qualquer função homomorficamente. Para resolver esse problema e transformar uma cifra que suporta uma quantidade limitada de operações em uma cifra completamente homomórfica, usa-se um procedimento chamado *bootstrapping*. Seu funcionamento é explicado em detalhe na seção 1.3.3, mas, por ora, saiba que ele serve para reduzir o ruído contido em um criptograma e que ele é a parte mais complexa e computacionalmente cara de toda cifra completamente homomórfica. Então, para avaliar uma função f homomorficamente, primeiro representa-se f em termos de operações homomórficas disponíveis, e.g, portas lógicas *AND*, *OR* e *NOT*, então prossegue-se aplicando essas operações até que o ruído atinga um limiar, e então executa-se o *bootstrapping* antes de continuar aplicando as próximas operações. Esse processo é repetido até que f seja inteiramente calculada.

1.1.3. Brevíssima introdução à linguagem de programação Sage

Sage[3] ou SageMath é uma linguagem de programação livre e código aberto bastante versátil: contém elementos dos paradigmas procedural, funcional e orientado a objetos, trabalha com cálculo simbólico e numérico, e oferece funcionalidades relacionadas com diversas áreas da matemática e criptografia. Além disso, é uma linguagem simples, com sintaxe praticamente igual a do Python. A principal exceção é que `x^y` significa "x elevado a y" em Sage e "x XOR y" em Python.

Sage trabalha nativamente tem diversos tipos de dados avançados, como números complexos, polinômios, vetores e matrizes. Além disso, em geral há conversão automática entre os tipos, como ilustrado no programa a seguir:

```
A = Matrix(ZZ, 2, 2, [[1, 2], [3, 4]]) # matriz integral 2x2
print(A.det()) # determinante de A: 1*4 - 2*3 = -2
u = vector(QQ, [0, 1/4]) # QQ = conjunto dos racionais
v = A*u
print(v) # imprime (1/2, 1)
```

É possível trabalhar diversas estruturas algébricas, como grupos, anéis e corpos. Também pode-se facilmente definir quocientes. Por exemplo, o anel $\mathbb{Z}/q\mathbb{Z}$, formado pelo quociente de $\mathbb{Z}$ e $\langle q \rangle$ pode ser obtido com o comando `ZZ.quotient(q)`. Dado um anel A, pode-se criar o anel de polinômios $A[X]$ usando `P.<x> = A['x']`. Novamente, é possível criar um quociente, como $A[X]/\langle X^N+1 \rangle$, o anel de polinômios módulo X^N+1 e com coeficientes em A, usando o comando `P.quotient(x^N + 1)`. Ademais, geralmente é possível obter um elemento aleatório de um conjunto qualquer, digamos, P, usando o comando `P.random_element()`. O código a seguir constrói o anel $R = \mathbb{Z}_q[X]/\langle X^N - 1 \rangle$ e imprime um vetor com n elementos aleatórios de R.

```
# -*- coding: utf-8 -*-
```

[3] Sigla em inglês para *system for algebra and geometry experimentation* (sistema algébrico e geométrico de experimentações).


```
N, n = 4, 10
q = next_prime(16) # primeiro primo maior que 16
Zq = ZZ.quotient(q) # inteiros mod 17
P.<x> = Zq['x'] # anel de polinômios mod q
f = x^N - 1 # definindo um polinômio
R = P.quotient(f) # quociente entre P e <f>
u = (R^n).random_element() # vetor aleatório
print(u) # imprime os n elementos
```

1.2. O problema do divisor comum aproximado (ACD)

Nesta seção, vamos estudar o problema computacional conhecido como ACD, do inglês *Approximate Common Divisor problem*[4]. Ele é o problema subjacente de uma família de cifras homomórficas conhecida como *FHE over the integers* (criptografia completamente homomórfica sobre os inteiros). Vamos implementar em Sage as distribuições relacionadas a esse problema e um algoritmo para resolvê-lo. Posteriormente, na Seção 1.3, vamos implementar uma cifra homomórfica baseada no ACD.

1.2.1. Definição do problema

Imagine que lhe foram dados vários múltiplos de um número primo p desconhecido, i.e., vários inteiros da forma $x_i := pq_i$ com q_i aleatório, e que sua tarefa é descobrir o valor de p. Como proceder? Claramente, esse problema pode ser resolvido com o cálculo de $\mathsf{mdc}(x_1, ..., x_\ell)$, i.e., dados vários múltiplos de p, calcule o máximo divisor comum e ele será provavelmente p. No entanto, se em vez de múltiplos de p, lhe forem dados "múltiplos aproximados", valores próximos dos múltiplos, da forma $x_i := pq_i + r_i$, onde r_i é um número aleatório relativamente pequeno em comparação com p. Como proceder agora? Esse problema simples é o ACD e todos os algoritmos para resolvê-lo levam tempo exponencial, mesmo algoritmos quânticos, por isso, esquemas criptográficos baseados no ACD são chamados de *pós-quânticos*.

Antes precisamos definir a distribuição de probabilidade subjacente, conforme segue.

Definição 1.2.1. *Sejam ρ, η e γ inteiros usados para parametrizar o problema e tais que $\gamma > \eta > \rho > 0$. Seja p um número primo que satisfaz $2^{\eta-1} \leq p \leq 2^\eta$. A distribuição de probabilidade $\mathscr{D}_{\gamma,\rho}(p)$, cujo suporte é $[\![0, 2^\gamma[\![$, é definida como*

$$\mathscr{D}_{\gamma,\rho}(p) := \{\textit{Amostre } q \leftarrow [\![0, 2^\gamma / p\,[\![\;\; e \;\; r \leftarrow]\!] - 2^\rho, 2^\rho [\![\,, \textit{ devolva } x := pq + r\}.$$

Agora conseguimos definir o problema ACD.

Definição 1.2.2 (ACD)**.** *O problema do divisor comum aproximado, com parâmetros ρ, η e γ, ou simplesmente $(\rho, \eta, \gamma) - \mathsf{ACD}$, é o problema de descobrir p dada uma amostra arbitrariamente grande da distribuição $\mathscr{D}_{\gamma,\rho}(p)$.*

A versão decisional do problema, chamada $(\rho, \eta, \gamma) - \mathsf{ACD}$ decisional, é o problema de distinguir entre a distribuição $\mathscr{D}_{\gamma,\rho}(p)$ e a distribuição uniforme $\mathscr{U}([\![0, 2^\gamma[\![)$.

[4]Também conhecido como AGCD *problem*, do inglês *Approximate Greatest Common Divisor problem*.

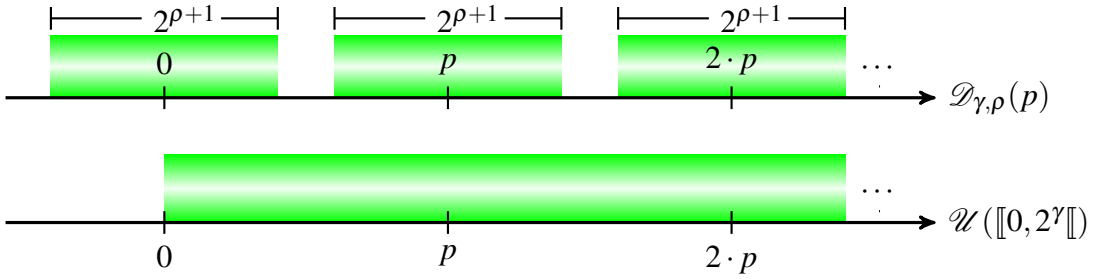


Figure 1.1. As duas distribuições que devem ser distinguidas na versão decisional do problema ACD**. As áreas verde representam os valores (inteiros) que podem ser amostrados de ambas as distribuições.**

Note que se x é uniformemente distribuído em $[\![0,2^{\gamma}[\![$, então usando a divisão Euclideana para escrever $x = pq + r$, temos que a distribuição de r é próxima da uniforme em $[\![0, p-1[\![$, em outras palavras, r pode ser igual a qualquer valor entre 0 e $p-1$. No entanto, todos os valores de $pq+r$ com $2^{\rho} \leq r < p \approx 2^{\eta}$ têm probabilidade zero em $\mathscr{D}_{\gamma,\rho}(p)$. Portanto, quanto maior for $\eta - \rho$, maior será a distância estatística entre $\mathscr{D}_{\gamma,\rho}(p)$ e $\mathscr{U}([\![0,2^{\gamma}[\![)$, logo, mais fácil o problema se torna. Isso é ilustrado na Figura 1.1.

Implementar a distribuição $\mathscr{D}_{\gamma,\rho}(p)$ em Sage é simples: basta usar o comando `ZZ.random_element`, como ilustrado no código a seguir:

```
def sample_r(rho):
    return ZZ.random_element(-2^rho,2^rho)
def sample_q(gamma, eta):
    return ZZ.random_element(0, 2^(gamma - eta))
def sample_acd(gamma, p, rho):
    eta = ceil(log(p, 2))
    r = sample_r(rho)
    q = sample_q(gamma, eta)
    return p*q + r
```

1.2.2. Criptoanálise do problema ACD

Os parâmetros γ, η e ρ não são variáveis livres que podem ser escolhidos livremente, pois a segurança dos esquemas criptográficos baseados no ACD depende desses parâmetros. Em geral, eles são escolhidos de tal forma que todos os algoritmos conhecidos para resolver o ACD levem tempo exponencial em λ, um parâmetro de segurança, i.e., se um algoritmo $\mathscr{A}$ resolve o ACD em tempo $T(\gamma,\eta,\rho)$, então $T(\gamma,\eta,\rho) = \Omega(2^{\lambda})$. Com isso, "quebrar a segurança" de um esquema baseado no ACD requer ao menos 2^{λ} operações e dizemos que o esquema oferece λ bits de segurança.

As principais duas famílias de algoritmos usados para resolver o ACD são os ataques por mdc (máximo divisor comum) e os ataques de reticulados ortogonais. Nesta seção, vamos estudar a estratégia geral dos ataques por mdc, implementar uma versão simples desse algoritmo e deduzir quais restrições ele impõe aos parâmetros.

A principal ideia por trás desse tipo de ataque é tentar remover o termo r_i de

$x_i := pq_i + r_i$ para obter múltiplos de p, então prosseguir calculando mdc deles para obter múltiplos cada vez menores. Note que se $a = p\prod_{i=1}^{n} q_i$ e $b = p\prod_{i=1}^{m} s_i$, então $\mathsf{mdc}(a,b) = p\prod_{t_i \in T} t_i$ sendo $T = \{q_1,...,q_n\} \cap \{s_1,...,s_m\}$. Então, quando o múltiplo obtido tem apenas η bits, é porque na verdade ele já é igual a p.

Para obter um múltiplo a partir de $x_i := pq_i + r_i$, basicamente, fazemos uma busca exaustiva no elemento r_i calculando o seguinte produto: $m = \prod_{r=-2^\rho}^{2^\rho}(x_i - r)$. Note que algum fator é igual a pq_i, logo $m \in p\mathbb{Z}$, como esperado.

Uma implementação em Sage do algoritmo é apresentada a seguir. Ele recebe uma lista $x_0,...,x_\ell \leftarrow \mathscr{D}_{\gamma,\rho}(p)$, da qual obtém os múltiplos de p. Para reduzir o custo de calcular o mdc, os fatores primos pequenos são removidos manualmente já no começo. Como todos os $\Theta(2^\rho)$ valores possíveis de r são calculados, o tempo de execução é exponencial em ρ, por isso, sugerimos que você o execute usando valores pequenos para os parâmetros, como $(\rho,\eta,\gamma) = (10,20,30)$ e uma lista com poucos elementos, e.g., $\ell = 25$. Existem várias formas de melhorar esse algoritmo, por exemplo, acelerando o cálculo de $\prod_{r=-2^\rho}^{2^\rho}(x_i - r)$ [CN12] ou tornando o algoritmo probabilístico [CNT12]. Assim, a complexidade temporal se torna $O(\mathsf{poly}(\gamma,\rho)\cdot 2^\rho)$, então, para garantir que o custo seja ao menos 2^ρ operações, basta usar $\rho = \Omega(\lambda)$.

```
# -*- coding: utf-8 -*-
# Este código usa a função sample_agcd definida anteriormente

def remove_fatores_pequenos(a, num_fact=1000):
    q = 2
    for _ in range(num_fact):
        while q.divides(a):
            a /= q
        q = next_prime(q)
    return ZZ(a)

def ataque_por_mdc(gamma, eta, rho, list_samples):
    x0 = list_samples[0]
    mult_p = prod([x0 - r for r in range(-2^rho, 2^rho)])
    mult_p = remove_fatores_pequenos(mult_p) # mult. de p

    for i in range(1, len(list_samples)):
        xi = list_samples[i]
        mi = prod([xi - r for r in range(-2^rho, 2^rho)])
        mult_p = gcd(mult_p, mi) # máximo divisor comum
        print("bitlen mult_p = %d" % mult_p.nbits())
        if eta >= log(mult_p, 2) >= eta-1:
            break

    return mult_p
```

Os ataques por reticulados [CS15,GGM16] têm complexidade $2^{\Omega(\gamma/(\eta-\rho)^2)}$, então basta usar $\gamma = \Omega(\lambda\cdot(\eta-\rho)^2)$ para que o custo seja de ao menos 2^λ operações. Portanto,

considerando todos os ataques, temos $\rho = \Omega(\lambda)$ e $\gamma = \Omega(\lambda \cdot (\eta - \lambda)^2)$, sendo η o único parâmetro livre. Obviamente, ele deve ser maior que λ, senão uma busca exaustiva poderia encontrar p em menos de 2^λ passos, mas quão maior que λ? Isso é definido pelas propriedades dos esquemas criptográficos baseados no ACD, como veremos na Seção 1.3.

1.3. DGHV: Uma cifra homomórfica simples baseada no problema ACD

1.3.1. Definição da cifra e de suas propriedades

Primeiramente vamos apresentar um esquema simétrico. Depois disso descreveremos os detalhes de implementação diretamente com trechos da implementação em Sage. Finalmente, mostraremos como o esquema simétrico pode ser transformado em um esquema assimétrico.

Antes de mais nada precisamos definir uma notação para representar a *redução modular centralizada*, conforme segue.

Definição 1.3.1. *Definimos pela notação* $[x]_n$ *o único inteiro* $-n/2 \leq x' < n/2$ *tal que* $x' \equiv x \pmod{n}$.

Essa notação também pode ser aplicada a polinômios, reduzindo cada coeficiente, e a vetores e matrizes, reduzindo cada entrada. A seguir, mostramos o um script Sage que implementa essas operações e que será nomeado utils.sage, para que possa ser importado em outros programas que serão apresentados nas seções seguintes:

```
def sym_mod(a, n):
    a = ZZ(a) % n
    if 2*a > n:
        return a - n
    return a

def sym_mod_poly(poly, q):
    return Zx([sym_mod(ZZ(ai), q) for ai in poly.list()])

def sym_mod_vec(vec, q):
    return [sym_mod_poly(vi, q) for vi in vec]
```

Algoritmo 1.1. utils.sage

Agora podemos descrever a cifra DGHV em detalhes.

Definição 1.3.2. *A seguir as funções do criptossistema* DGHV *são definidas.*

- $\mathsf{KeyGen}(1^\lambda)$*: recebe o parâmetro de segurança* λ *e gera os valores* γ, η, ρ *como descrito na seção anterior. A chave secreta* sk *é escolhida como sendo um primo* p *de* η *bits.*

- $\mathsf{Enc}(\mathsf{sk}, m)$*: gera um criptograma* c *que cifra* m *sob a chave* $\mathsf{sk} = p$*. Para isso, amostra-se* $c' \leftarrow \mathscr{D}_{\gamma,\rho}(p)$ *até que* $[c']_p$ *seja par e devolve-se* $c = c' + m$*. Note que* c *é da forma* $pq + 2r + m$*.*

- $\mathsf{Dec}(\mathsf{sk}, c)$: *recupera a mensagem m cifrada por c. Para isso, calcula-se* $m = [c]_p \pmod 2$.

Agora vamos mostrar o código Sage que implementa o esquema DGHV. Antes disso, alguns comentários são importantes. Note que a definição 1.3.2 não determina qual é o espaço de texto claro, ou seja, não especifica o domínio da mensagem m. Para que o crescimento do ruído seja minimizado, devemos escolher um espaço de texto claro de tamanho mínimo. Em geral, podemos considerar que $m \in \mathbb{Z}_t$, com t pequeno. Aqui, utilizamos $t = 2$, de modo que $m \in \{0,1\}$. Além disso, é possível perceber uma diferença em relação a distribuição $\mathscr{D}_{\gamma,\rho}(p)$, pois ao cifrar uma mensagem temos que o ruído r é multiplicado por t. Isto é necessário para que seja possível decifrar a mensagem posteriormente utilizando uma redução modular por t, eliminando o ruído.

Outro comentário importante é sobre a variável x_0, calculada como sendo um múltiplo exato de p. Para que seja computacionalmente difícil calcular p dado x_0, é necessário que a fatoração de x_0 seja difícil o suficiente, o que pode ser obtido escolhendo η e γ suficientemente grande. Nós utilizaremos x_0 para limitar o tamanho do texto cifrado, já que o espaço de texto cifrado é dado por $\mathbb{Z}_{x_0}$.

```
load("utils.sage")
load("distribution_acd.sage")

class DGHV:
    def __init__(self, gamma, eta, rho, t = 2, p = 1):
        assert(gamma > eta)
        assert(eta > rho)
        if 1 == p:
            p = random_prime(2^eta, lbound=2^(eta - 1))
        else:
            # if p is given, it must have eta bits
            assert(eta-1 <= p.nbits() <= eta)

        self.gamma = gamma
        self.eta = eta
        self.rho = rho
        self.t = t
        self.p = p
        self.x0 = p * sample_q(gamma, eta) #+ self.sample_r()
        self.Zp = ZZ.quotient(p)
        self.Zx0 = ZZ.quotient(self.x0)

    def enc(self, m):
        q = sample_q(self.gamma, self.eta)
        r = sample_r(self.rho)
        c = self.p*q + self.t * r + m
        c %= self.x0
        return c
```

```
    def dec(self, c):
        noisy_msg = sym_mod(c, self.p) # == t * r + msg
        return noisy_msg % self.t
```

Agora podemos examinar como são realizadas as operações homomórficas. Como o x espaço de texto encriptado é $\mathbb{Z}_0$, então as operações homomórficas são realizadas neste anel, e portanto são de fácil implementação em Sage.

```
    def not_gate(self, c):
        return (1 - c) % self.x0

    def add(self, c1, c2):
        return (c1 + c2) % self.x0

    def mult(self, c1, c2):
        return c1 * c2 %self.x0
```

1.3.2. Análise da evolução do ruído devida às operações homomórficas

Como a multiplicação homomórfica aumenta o ruído significativamente, enquanto a adição praticamente não muda o ruído, a quantidade de produtos efetuados para gerar um criptograma é comumente usada como estimativa do ruído. Nosso objetivo agora é implementar uma comparação homomórfica de inteiros com ℓ bits e verificar como o ruído aumenta conforme as operações são efetuadas.

Primeiro, note que dados bits x e y, a função $c(x,y) = (x+y)+1 \bmod 2$ é igual a 1 se os dois bits são iguais e a 0 se são diferentes – $c(x,y)$ é equivalente a $\neg(x \text{ xor } y)$. Portanto, dados inteiros a e b com n bits, podemos testar se eles são iguais comparando os bits correspondentes e multiplicando o resultado, i.e., $\text{comp}(a,b) = \prod_{i=1}^{n} c(a_i,b_i)$, onde a_i representa o i-ésimo bit de a (e analogamente para b_i). O código a seguir implementa essa comparação. Para executá-lo, é preciso escolher uma quantidade de bits L e inicializar um objeto do tipo DGHV com parâmetros como $(\gamma,\eta,\rho) = (\lambda^2,(L+1)\cdot\lambda,\lambda)$, então invocar a função passando o objeto e n=L.

```
def comparacao_homomorfica(dghv, n=3):
    m0 = ZZ.random_element(0, 2^n)
    m1 = ZZ.random_element(0, 2^n)
    bits0 = m0.digits(base=2, padto=n) # lista com n bits
    bits1 = m1.digits(base=2, padto=n) # lista com n bits
    c0 = [dghv.enc(bi) for bi in bits0] # cifra cada bit
    c1 = [dghv.enc(bi) for bi in bits1] # cifra cada bit

    # compara homomorficamente
    c, m = 1, 1
    for i in range(n):
        cmp_i = dghv.add(c0[i], c1[i]) # enc(0) <==> c0[i] == c1[i]
        cmp_i = dghv.not_gate(cmp_i) # enc(1) <==> c0[i] == c1[i]
```

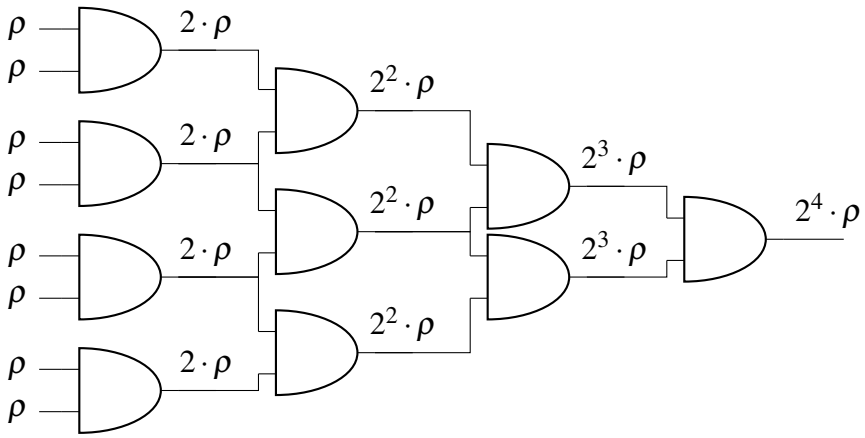


Figure 1.2. Circuito composto apenas de portas lógicas e (multiplicações). Os valores $k\rho$ indicam o logaritmo dos ruídos. A profundidade do circuito é 4, os criptogramas da entrada têm ruído 2^{ρ} e a magnitude do ruído da saída é $2^{2^4 \cdot \rho}$.

```
   c = dghv.mult(c, cmp_i) # c *= cmp_i
# decifra e verifica
res = dghv.dec(c)
assert((m0 == m1) == res)
```

Algoritmo 1.2. Comparação de inteiros homomórfica.

Para estimar o ruído durante a execução, a cada criptograma é associado um nível. Criptogramas que não passaram por nenhuma operação homomórfica são considerados como estando no "nível 1" e o ruído tem magnitude próxima de 2^{ρ}. Ao multiplicar dois criptogramas cujos níveis são n_1 e n_2 e ruídos são aproximadamente $2^{n_1\rho}$ e $2^{n_2\rho}$, obtém-se um criptograma no nível n_1+n_2 com ruído próximo de $2^{(n_1+n_2)\rho}$. Portanto, a comparação homomórfica que implementamos gera um criptograma com ruído aproximadamente $2^{n\rho}$. Mas note que no pior caso a profundidade multiplicativa do circuito corresponde ao logaritmo do nível do criptograma, como ilustrado na Figura 1.2.

Como o deciframento só funciona enquanto o ruído for menor do que $p/2 \approx 2^{\eta}$, a profundidade L dos circuitos que podem ser executados homomorficamente é limitada por $2^{2^L\rho} < 2^{\eta}$, ou seja, $L = O(\log(\eta/\rho))$. Essa é uma enorme limitação do esquema DGHV, pois (sem *bootstrapping*) ele não permite mesmo que circuitos "rasos", com profundidade λ, sejam avaliados homomorficamente, já que para isso seria necessário usar $\eta = \Omega(2^L \cdot \rho) = \Omega(2^{\lambda} \cdot \lambda)$, ou seja, exponencial em λ, logo, como $\gamma > \eta$ e todos os criptogramas tem γ bits, mesmo cifrar uma mensagem levaria tempo exponencial. Por isso, é preciso se limitar a $L = O(\log \lambda)$, para que η e γ sejam apenas polinomiais em λ.

1.3.3. *Boostrapping*: transformando uma cifra homomórfica em uma cifra completamente homomórfica

Como vimos anteriormente, há uma quantidade limitada de operações homomórficas que são permitidas, pois quando o ruído ultrapassa um determinado limiar, então não é mais possível decifrar as mensagens. Sendo assim, gostaríamos de ter um mecanismo que reduzisse o ruído para seu nível inicial, isto é, aquele que é gerado ao cifrar a mensagem,

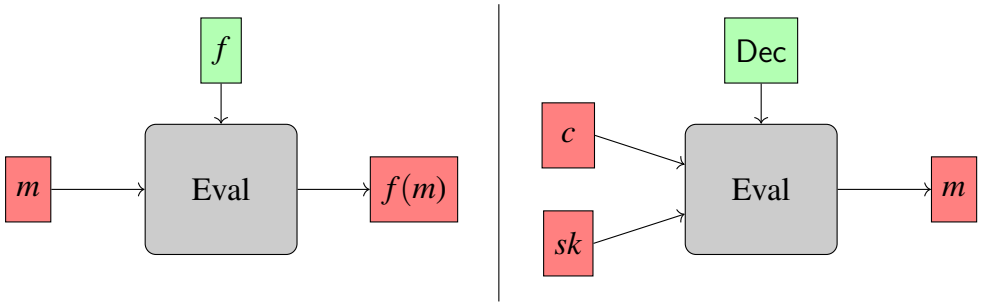


Figure 1.3. Como a avaliação homomórfica funciona em geral e como ela é usada durante o *bootstrapping*. Caixas vermelhas representam valores cifrados e verdes representam valores públicos.

ou seja, um ruído com ρ bits. Intuitivamente, gostaríamos de renovar o texto cifrado sempre que o ruído estivesse próximo de ultrapassar o limiar. Uma forma indesejada de resolver tal problema é simplemente decifrando e cifrando novamente, já que decifrar remove completamente o ruído e cifrar novamente gera um novo criptograma com pouco ruído. No entanto, decifrar exige conhecimento da chave privada. A principal observação de Gentry em [Gen09] foi que podem-se usar as operações homomórficas da cifra para avaliar sua própria função de decifração. Como avaliar homomomorficamente f em um texto cifrado $\mathsf{Enc}(m)$ gera $\mathsf{Enc}(f(m))$, é possível avaliar $f = \mathsf{Dec}$ em $\mathsf{Enc}(c)$ e $\mathsf{Enc}(sk)$ para obter $\mathsf{Enc}(f(c,\mathsf{sk})) = \mathsf{Enc}(\mathsf{Dec}_{\mathsf{sk}}(c)) = \mathsf{Enc}(m)$, ou seja, uma nova encriptação de m. Todo o ruído de c é removido pela decifração e o ruído contido no novo criptograma consiste apenas no ruído acumulado pelas operações homomórficas necessárias para computar o circuito Dec. Esse operação, chamada de *bootstrapping*, é ilustrada na figura 1.3.3. Note que a chave secreta nunca é revelada, pois apenas $\mathsf{Enc}(\mathsf{sk})$ é usado. No entanto, isso introduz uma nova hipótese de segurança, a saber, a segurança circular: é preciso supor que é seguro cifrar sk sob a própria chave sk. É importante notar, contudo, que essa hipótese é vista como fraca, já que se acredita que a maioria das cifras é circularmente segura.

1.3.4. Aplicando o *bootstrapping* à cifra DGHV

O esquema descrito na Definição 1.3.2, porém, não é capaz de avaliar homomomorficamente seu próprio circuito de deciframento, em resumo porque reduções modulares são caras. Deste modo, mostraremos modificações sobre a proposta apresentada na seção anterior que tornam possíveis a implementação do *bootstrapping*. Como precisamos ser capazes de avaliar o circuito de deciframento homomorficamente, temos que, informalmente, facilitar o processo de deciframento, sem que isso seja um problema de segurança, pois esta facilidade não pode estar disponível para adversários. Porém, o algoritmo de deciframento descrito anteriormente necessita de operações cujo circuito é muito caro em termos de profundidade multiplicativa, pois precisa calcular reduções modulares. Sendo assim, veremos nesta seção como é possível melhorar este aspecto. Inicialmente, consideremos a seguinte adaptação, que permite construir um esquema assimétrico. Note que esta etapa não é estritamente necessária, mas será apresentada a título de completude.

Definição 1.3.3. - $\mathsf{KeyGen}(\lambda)$*: obtenha o inteiro aleatório p com η bits. Para* $0 \leq$

$i \leq \tau$, calcule $x_i = \mathscr{D}_{\gamma,\rho}(p)$. Renomeie os índices de modo que x_0 corresponda ao maior elemento. Repita até que x_0 seja ímpar e $x_0 \pmod p$ seja par. A chave pública é dada por $\mathsf{pk} = (x_0, \ldots, x_\tau)$ *e a chave privada é dada por* $\mathsf{sk} = p$.

- $\mathsf{Enc}(m)$*: escolha aleatoriamente o conjunto $S \subset \{0, 1, \ldots, \tau\}$ e o inteiro aleatório r no intervalo $(-2^{\rho'}, 2^{\rho'})$ e compute $c = [m + 2r + \sum_{i \in S} x_i]_{x_0}$.*

- $\mathsf{Dec}(c)$*: retorne $m = [c]_p \pmod 2$.*

- $\mathsf{Eval}(C, c_1, \ldots, c_t)$*: dado o circuito C e t textos cifrados, execute as operações de C módulo x_0 sobre os textos cifrados, dados por números de precisão arbitrária, e retorne o resultado.*

Agora podemos mostrar mais uma adaptação, que faz com que o algoritmo de deciframento seja eficiente o suficiente para permitir o *bootstrapping*. Em particular, note que o deciframento precisa de um nível de multiplicações por z_i e após isso pode ser concluído apenas por somas e subtrações.

Definição 1.3.4. *Esta construção usa 3 novos parêmetros: $\kappa = \gamma\eta/\rho'$, $\theta = \lambda$ e $\Theta = \omega(\kappa \log \lambda)$, ou seja, eles são todos polinomiais em relação ao parâmetro de segurança λ.*

- $\mathsf{KeyGen}(\lambda)$*: compute* sk *e* pk *como na definição 1.3.3. Compute $x_p = \lfloor 2^\kappa / p \rceil$, escolha aleatoriamente o vetor $s = \langle s_1, \ldots, s_\Theta \rangle$, com Θ bits e* peso de Hamming θ*. O conjunto S é definido por*

$$S = \{i \mid s_i = 1\}.$$

Escolha aleatoriamente os inteiros u_i, onde $1 \leq i \leq \Theta$, com no máximo κ bits, tal que $\sum_{i \in S} u_i = x_p \pmod{2^{\kappa+1}}$. Compute $y_i = u_i / 2^\kappa$, tal que cada y_i é um inteiro positivo menor ou igual a 2, com κ bits de precisão após a parte fracionária. Com isso temos que $[\sum_{i \in S} y_i]_2 = (1/p) - \Delta_p$, para $\Delta_p < 2^{-\kappa}$.

A chave privada é dada pelo vetor $(s_1, \ldots, s_\kappa)$ e a chave pública é dada por pk *e o vetor $(y_1, \ldots, y_\Theta)$.*

- $\mathsf{Enc}(m)$*: compute c como no esquema original. para $1 \leq i \leq \Theta$, compute $z_i = [c y_i]_2$, mantendo apenas $\lceil \log \theta \rceil + 3$ de precisão para cada z_i. Returne c e o vetor $(z_1, \ldots, z_\Theta)$.*

- $\mathsf{Dec}(c)$*: returne $m' = [c - \lfloor \sum_{i \in S} s_i z_i \rceil]_2$.*

- $\mathsf{Eval}(C, c_1, \ldots, c_t)$*: Somas e multiplicações são calculadas pelas operações usuais sobre os racionais.*

Os detalhes deste processo podem ser encontrados no minicurso [DM12] de 2012. Aqui, o objetivo maior é adquirir a intuição de como o *bootstrapping* funciona. Em resumo, o que há por trás desta modificação é que parte do trabalho de deciframento foi transferida para o algoritmo de encriptação, que passou a ser responsável por encontrar

valores z_i peculiares, tais que o deciframento possa ser realizado basicamente pela soma destes elementos. Porém, para que tal esquema ainda seja seguro, tais elementos devem estar escondidos em meio a um conjunto maior, de modo que um adversário não seja capaz de advinhar quais elementos deste conjunto devem um não ser usados no deciframento. Este é um problema clássico em computação, e é conhecido como SSP (*subset sum problem*). Os parâmetros deste esquema modificado devem então proteger contra adversários que tentem resolver o problema SSP subjacente, além dos problemas já mencionados de força bruta no ruído e o ACD. Em resumo, para obter segurança, precisamos escolher θ suficientemente grande para que o problema ofereça grau de segurança λ. Também é preciso ter Θ pertencente a $\omega(\kappa \log \lambda)$, onde κ é o tamanho em bits dos números que formam a chave pública.

1.3.5. Outras direções

Existem alguns trabalhos que podemos apontar para aqueles que tiverem mais interesse no assunto. Por exemplo:

- é possível operar sobre vetores de textos claros, onde a operação de soma e multiplicação é realizada coordenada a coordenada, oferecendo portanto paralelismo ao sistema. Para isso, podemos usar o teorema Chinês dos restos, como neste artigo [CKLY15]. Resumidamente, o espaço de texto claro ao invés de $\mathbb{Z}_t$ passa a ser $\mathbb{Z}_{t_1} \times \ldots \mathbb{Z}_{t_\ell}$, com t_i primos entre si. Em particular, o esquema pode ser atraente quando ℓ é grande. O texto claro também pode ser interpretado como um único inteiro módulo $\prod_{i=0}^{\ell}(t_i)$;

- um resultado relevante é o trabalho que mostra como implementar o *bootstrapping* em menos de um segundo [Per21a], tornando possível implementar FHE sobre os números inteiros de modo mais eficiente;

- neste outro trabalho importante da literatura [CS15] temos estabelecida uma relação entre o problema ACD e o problema LWE, que será o foco da próxima seção.

1.4. Os problemas criptográficos LWE e RLWE

O problema LWE (aprendizagem com ruídos, ou aprendizagem com erros, do inglês, *Learning With Errors*) [Reg09] se tornou uma ferramenta fundamental para a criptografia moderna, pois é um problema considerado difícil mesmo para computadores quânticos, e é versátil o suficiente para que diversas primitivas criptográficas possam ser baseadas nele. Por exemplo, no processo de padronização de primitivas pós-quânticas organizado pelo NIST[5], há cifras de chave pública, esquemas de encapsulamento de chave e de assinatura digital baseados no LWE (ou variantes do LWE).

Talvez sua característica mais singular seja a garantia de que mesmo instâncias aleatórias do problema são difíceis. Isso não é verdade para outros problemas criptográficos usados frequentemente. Por exemplo, considere a fatoração de inteiros da forma $n = pq$ usada no RSA. É sabido que não se pode escolher p e q como dois números

[5]https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

primos quaisquer, pois é muito mais fácil fatorar n quando seus dois fatores satisfazem certas propriedades (e.g., se $|p-q| < \sqrt{n}$, então o simples método de fatoração de Fermat é suficiente para recuperar p e q).

O problema LWE consiste no seguinte: considere um vetor secreto $\mathbf{s} \in \mathbb{Z}^n$ e m equações lineares $b_i = \mathbf{a}_i \cdot \mathbf{s} \in \mathbb{Z}_q$, onde $\mathbf{s}$ é o mesmo em todas as equações. Convenientemente, dadas $m = n$ equações, podemos construir uma matriz $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ em que cada linha corresponde a um vetor $\mathbf{a}_i$. Da mesma forma, podemos construir um vetor $\mathbf{b} = (b_1, ..., b_n)^T$. então, como sabemos que $\mathbf{b} = \mathbf{A} \cdot \mathbf{s} \pmod q$, basta calcular $\mathbf{A}^{-1} \cdot \mathbf{b} \bmod q$ para encontrar $\mathbf{s}$. No entanto, se em vez de equações, soubéssemos apenas que $b_i \approx \mathbf{a}_i \cdot \mathbf{s} \pmod q$, ou seja, que $b_i = \mathbf{a}_i \cdot \mathbf{s} + e_i \pmod q$, onde e_i é algum inteiro pequeno, como poderíamos encontrar $\mathbf{s}$? Adicionar esses pequenos "erros", ou ruídos, às equações faz com que seja extremamente difícil descobrir o valor de $\mathbf{s}$. Mais formalmente, os termos e_i são ruídos Gaussianos discretos, ou seja, seguem uma distribuição análoga à normal, mas sobre $\mathbb{Z}$ em vez de $\mathbb{R}$. Denotamos essa Gaussiana discreta por χ_σ, ou apenas χ quando σ estiver claro no contexto, e cada inteiro x é amostrado de χ com probabilidade proporcional a $e^{-x^2/(2\sigma^2)}$.

Definição 1.4.1. *Considere $n, q \in \mathbb{N}^*$ e $\sigma \in \mathbb{R}$ tal que $\sigma > 0$. Seja $\mathbf{s} \in \mathbb{Z}_q^n$ um vetor fixo. A distribuição $\mathscr{A}_{\mathbf{s},\sigma}$ é definida como*

- *amostre $\mathbf{a}$ uniformemente de $\mathbb{Z}_q^n$ e amostre e seguindo a distribuição χ_σ;*
- *calcule $b := \mathbf{a} \cdot \mathbf{s} + e \bmod q$;*
- *devolva $(\mathbf{a}, b)$.*

Definição 1.4.2 (LWE)**.** *O problema* LWE *com parâmetros n, q e σ, ou simplesmente $(n, q, \sigma) -$* LWE*, é o problema de encontrar $\mathbf{s}$ dada uma amostra arbitrariamente grande da distribuição $\mathscr{A}_{\mathbf{s},q,\sigma}$.*

O problema LWE *de decisão consiste em distinguir entre as distribuições $\mathscr{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)$ e $\mathscr{A}_{\mathbf{s},q,\sigma}$.*

É sabido que as duas versões desse problema são computacionalmente equivalentes, i.e., se existe um algoritmo A que resolve qualquer uma delas em tempo polinomial, então é possível usar A como uma sub-rotina de um algoritmo A' que resolve a outra versão do problema também em tempo polinomial [Reg09].

Assim como a distribuição $\mathscr{D}_{\gamma,\rho}(p)$ associada com o problema ACD, implementar $\mathscr{A}_{\mathbf{s},q,\sigma}$ em Sage é simples:

```
from sage.stats.distributions.discrete_gaussian_integer \
        import DiscreteGaussianDistributionIntegerSampler \
        as DiscreteGaussian

class LWEDistribution:
    def __init__(self, s, q, sigma=3.2):
        self.n = len(s)
        self.s = s
```

```
        self.sigma = sigma
        self.D = DiscreteGaussian(sigma)
        self.Zq = ZZ.quotient(q)

    def random_noise(self):
        return self.D()

    def random_a(self):
        a = [self.Zq.random_element() for _ in range(self.n)]
        a = vector(self.Zq, a) # converte lista em vetor
        return a

    def sample(self):
        n, s = self.n, self.s
        a = self.random_a() # vetor em Zq^n
        e = self.random_noise()
        b = a*s + e
        return a, b
```

1.4.1. Criptanálise do problema LWE

Os parâmetros n, q, σ e também a distribuição de $\mathbf{s}$ são escolhidos de tal forma que todos os algoritmos que resolvem o LWE levem tempo exponencial no parâmetro de segurança λ. Na verdade, tratar q e σ separadamente é irrelevante para as análises de segurança, pois a dificuldade de resolver o LWE depende de $\alpha := q/\sigma$, i.e., da razão entre q e o tamanho dos ruídos. Para entender intuitivamente isso, note que se $(\mathbf{a}, b := \mathbf{as} + e \pmod q)$ é uma amostra LWE, então, $(k \cdot \mathbf{a}, k \cdot b := k \cdot \mathbf{as} + k \cdot e \pmod{k \cdot q})$ também o é, mas com respeito ao módulo kq. Obviamente, esse novo LWE definido sobre $\mathbb{Z}_{kq}$ não pode ser mais fácil que o original, pois se houvesse um algoritmo eficiente para resolver esse $(n, kq, k\sigma)$-LWE, poderíamos transformar o (n, q, σ)-LWE multiplicando por k e usar esse algoritmo eficiente para recuperar $\mathbf{s}$. Mas note que $kq/(ke) = q/e$, ou seja, podemos aumentar livremente o ruído e o módulo, mantendo a mesma razão ente eles, sem aumentar a dificuldade do problema. Um argumento parecido (mas mais delicado) mostra que também é possível dividir ambos o ruído e o módulo sem alterar a dificuldade do LWE.

Por isso, em vez de trabalhar com a fração q/σ para derivar os parâmetros, é comum simplesmente definir σ como um valor pequeno, como 3.2, e ajustar q para que a fração tenha o valor desejado. Assim, para simplificar a análise, também usaremos essa metodologia.

Como a distribuição normal é concentrada perto da média[6], o valor dos ruídos adicionados nas amostras do LWE são pequenos: com probabilidade praticamente igual a um, cada e_i pertence ao intervalo $[\![-6\sigma, 6\sigma]\!]$, ou seja, há 13σ valores possíveis. Portanto, assim como no caso do problema ACD, é possível tentar eliminar o ruído e obter equações exatas, então resolvê-las para achar $\mathbf{s}$. Para isso, seriam necessárias n amostras de $\mathscr{A}_{\mathbf{s},q,\sigma}$,

[6]Valores que estão mais do que três desvios-padrão de distância da média já se encontram nas caldas da normal e têm probabilidade muito baixa de serem amostrados.

para termos $\mathbf{b} = \mathbf{As} + \mathbf{e} \pmod{q}$ sendo $\mathbf{A}$ uma matriz quadrada, $n \times n$, como descrito anteriormente. Nesse caso, calcularíamos $\mathbf{A}^{-1}$ e então, para cada $\mathbf{e}_i$ possível, definiríamos $\mathbf{b}_i = \mathbf{b} - \mathbf{e}_i$ e $\mathbf{s}_i = \mathbf{A}^{-1}\mathbf{b}_i \bmod q$. Note que quando $\mathbf{e}_i = \mathbf{e}$, o valor correto de $\mathbf{s}$ é recuperado. Como há $(13\sigma)^n$ vetores $\mathbf{e}_i$ possíveis, basta tomar $n = \Omega(\lambda)$ para inviabilizar esse ataque, fazendo sua complexidade temporal ser $2^{\Omega(\lambda)}$.

Considerando ainda que σ é um valor fixo e pequeno, resta saber como escolher q: as restrições sobre q são obtidas considerando ataques por reticulados. Para um n fixo, esses ataques se tornam mais eficientes a medida que q cresce. Basicamente, para um nível de segurança λ, temos a restrição $\log q \in O(n\log(\lambda)/\lambda)$.

A escolha do vetor secreto $\mathbf{s}$ também influencia a segurança do LWE, mas seu impacto é bem pequeno. A forma padrão de usar o LWE é escolhendo $\mathbf{s}$ uniformemente em $\mathbb{Z}_q^n$, mas é possível amostrar cada coordenada s_i seguindo a mesma distribuição que o ruído, i.e., χ, ou até como um bit aleatório, i.e., $s_i \leftarrow \mathscr{U}(\{0,1\})$.

Para obter valores concretos em vez de assintóticos, o estimador *online* apresentado em [APS15] é frequentemente usado.

1.4.2. LWE sobre anéis polinomiais

Em uma amostra do LWE, $(\mathbf{a}, b := \mathbf{as} + e)$, o vetor $\mathbf{a}$ é uniformemente distribuido em $\mathbb{Z}_q^n$. Assumindo que o problema LWE é difícil, obtemos que b também é uniforme, ou mais precisamente, b é computacionalmente indistinguível de um valor uniforme em $\mathbb{Z}_q$. Então, dada uma mensagem m, a soma $b + m \bmod q$ serve como uma cifra de m, pois b age como *one-time pad*. No entanto, para decifrar b, não basta conhecer a chave secreta $\mathbf{s}$, o vetor $\mathbf{a}$ também é necessário, logo, ele deve ser incluído no criptograma. Isso significa que um bit (ou um inteiro pequeno) m gera um criptograma $\mathbf{c} \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, o que representa uma expansão enorme, de $(n+1)\log q$ bits. Se cada criptograma tivesse o mesmo tamanho, mas pudesse encriptar n bits em vez de um, a expansão seria reduzida para $((n+1)\log q)/n = O(\log q)$.

Essa é a principal motivação do problema RLWE (LWE sobre anéis, do inglês *Ring Learning With Errors*). Ele é definido fixando um anel $R = \mathbb{Z}[X]/\langle f(X)\rangle$, onde f é um polinômio de grau n, e substituindo os vetores $\mathbf{a}$ e $\mathbf{s}$ por polinômios $a(X), s(X) \in R$. Note que $R = \{g(X) \in \mathbb{Z}[X] : \mathsf{grau}(g) < n\}$ e todas as operações em R são feitas módulo f. Assim, $a \cdot s \bmod f$ é um polinômio em vez de um inteiro, logo, tem mais espaço para guardar informação, mais especificamente, podemos encriptar um polinômio $m(X) = \sum_{i=0}^{n-1} m_i \cdot X^i$. Para isso, o ruído também é definido como um elemento de R e b é calculado como $(a \cdot s + e \bmod f) \bmod q$. Agora, um criptograma tem $2n\log q$ bits, mas cifra n bits, logo, a expansão é de $(2n\log q)/n = O(\log q)$, como desejado.

Por razões de segurança, escolhemos N tal que $n = \varphi(N)$, onde $\varphi(\cdot)$ é a função phi de Euler[7], e o polinômio f é definido como o N-ésimo polinômio ciclotômico, denotado por $\Phi_N(X)$ e definido como $\Phi_N(X) = \prod_{k \in S_N}(X - \exp(2i\pi k/N))$, onde $S_N := \{x \in \mathbb{N} : x \leq N \text{ e } \mathsf{mdc}(x,N) = 1\}$ e i é a unidade imaginária, i.e., $i^2 = -1$. É fácil ver que o grau de $\Phi_N(X)$ é $|S_N| = \varphi(N)$. Ademais, apesar de ser definido como um polinômio com raízes

[7] $\varphi(N)$ é definida como a cardinalidade do conjunto $\{x \in \mathbb{N} : x \leq N \text{ e } \mathsf{mdc}(x,N) = 1\}$, ou seja, a quantidade de coprimos menores que ou iguais a N.

complexas, todos os seus coeficientes são inteiros e ele é irredutível em $\mathbb{Z}[x]$. Em Sage, há um comando específico para obter o N-ésimo polinômio ciclotômico:

```
print(cyclotomic_polynomial(4)) # x^2 + 1
print(cyclotomic_polynomial(5)) # x^4 + x^3 + x^2 + x + 1
print(cyclotomic_polynomial(6)) # x^2 - x + 1
print(cyclotomic_polynomial(16))# x^8 + 1
print(cyclotomic_polynomial(20))# x^8 - x^6 + x^4 - x^2 + 1
```

Como ilustrado acima, em geral, é difícil prever os coeficientes de $\Phi_N(X)$, mas quando N é uma potência de 2, temos $n := \varphi(N) = N/2$ e $\Phi_N(X)$ é simplesmente $X^n + 1$. Além disso, essa escolha de N facilita a implementação dos esquemas e os torna mais eficientes, já que a reduzir um polinômio módulo $X^n + 1$ requer apenas substituir X^n por -1. Por exemplo, $X^{2n} + 3X^{n+2} + 4 \equiv (-1)^2 + 3 \cdot (-1) \cdot X^2 + 4 \equiv -3X^2 + 5 \pmod{X^n + 1}$. Ademais, para multiplicar polinômios módulo $X^n + 1$, pode-se usar algoritmos de transformadas rápidas de Fourier mais simples do que para o caso geral, módulo $\Phi_N(X)$. Por isso, é comum considerar apenas potências de dois ao se construir cifras baseadas no RLWE e também faremos isso ao definirmos o esquema GSW na seção 1.5.2.

Além do anel R, também é preciso definir $R_q := R/qR = \mathbb{Z}_q[X]/\langle \Phi_N(X) \rangle$, i.e., anel de polinômios com grau menor que $n := \varphi(N)$ e coeficientes em $\mathbb{Z}_q$. Todas as operações nesse anel são feitas módulo $\Phi_N(X)$ e módulo q. Considerando o que foi discutido até então, o problema RLWE é definido como a seguir:

Definição 1.4.3 (Distribuição adjacente do RLWE)**.** *Considere $N, q \in \mathbb{N}^*$ e $\sigma \in \mathbb{R}$ tal que $\sigma > 0$. Sejam $n := \varphi(N)$, $R := \mathbb{Z}[X]/\langle \Phi_N(X) \rangle$ e $R_q := R/qR$. Finalmente, seja $s \in R_q$ um polinômio fixo. A distribuição $\mathscr{R}_{s,n,q,\sigma}$ é definida como*

- *amostre a uniformemente de R_q;*
- *para $0 \leq i < n$, amostre $e_i \leftarrow \chi_\sigma$ e defina $e = \sum_{i=0}^{n-1} e_i X^i \in R$;*
- *calcule $b := a \cdot s + e$ em R_q;*
- *devolva $(a, b) \in R_q^2$.*

Definição 1.4.4 (RLWE)**.** *O problema* RLWE *com parâmetros N, q e σ, ou simplesmente $(N, q, \sigma) - \mathsf{LWE}$, é o problema de encontrar s dada uma amostra arbitrariamente grande da distribuição $\mathscr{R}_{s,n,q,\sigma}$. O problema* RLWE *decisional é o problema de distinguir entre as distribuições $\mathscr{U}(R_q \times R_q)$ e $\mathscr{R}_{s,n,q,\sigma}$.*

O código a seguir mostra como implementar a distribuição $\mathscr{R}_{s,n,q,\sigma}$ em Sage. É notável a semelhança com o código que implementa $\mathscr{A}_{\mathbf{s},q,\sigma}$.

```
from sage.stats.distributions.discrete_gaussian_integer \
        import DiscreteGaussianDistributionIntegerSampler \
        as DiscreteGaussian

Zx.<x> = ZZ['x']
```

```
class RLWEDistribution:
    def __init__(self, s, N, q, sigma=3.2):
        self.n = N
        self.f = x^N + 1 # assume N = 2^k
        self.s = s
        self.sigma = sigma
        self.D = DiscreteGaussian(sigma)
        self.Zqx = ZZ.quotient(q)['x']
        self.Rq = self.Zqx.quotient(self.f)


    def random_noise(self):
        # polinômio com grau <= n-1 e coeficientes gaussianos
        return Zx([self.D() for _ in range(self.n)])

    def random_a(self):
        return self.Rq.random_element()

    def sample(self):
        s = self.s
        a = self.random_a() # coeficientes em Zq
        e = self.random_noise()
        b = (a*s + e) # mod f e q calculado automaticamente
        return [a, b]
```

1.5. Diminuindo a taxa de crescimento do ruído

A velocidade com que o ruído cresce ao se efetuar operações homomórficas é um dos fatores mais importantes para determinar a eficiência da cifra, pois se o ruído acumulado por cada operação é grande, são necessários parâmetros grandes, o que implica em criptogramas longos e operações mais lentas. Além disso, a classe de circuitos que podem ser avaliados homomorficamente sem *bootstrapping* também diminui conforme a taxa de crescimento do ruído aumenta. Por exemplo, na Seção 1.3.2, vimos que o ruído do DGHV cresce de forma exponencial, portanto, para avaliar um circuito de profundidade L, é preciso escolher parâmetros maiores que 2^L, e isso dificulta inclusive a execução do *bootstrapping*. Por isso, diversos artigos apresentaram ideias para diminuir o crescimento do ruído [BGV12, FV12, GSW13, CS15].

Enquanto as adições homomórficas praticamente não aumentam o ruído, as multiplicações têm um efeito desastroso. Ao se analisar o porquê disso, observa-se que ao multiplicarmos dois criptogramas, os ruídos são multiplicados por valores cuja magnitude é enorme. Por exemplo, no DGHV, $c_1 \cdot c_2$ resulta em $c_1 \cdot 2r_2$ e, finalmente, em $4r_1 \cdot r_2$. Então, uma forma de reduzir o crescimento do ruído é fazendo com que os ruídos sejam multiplicados apenas por valores pequenos. Mas como fazer isso? A ideia é a seguinte: em vez de multiplicar c_1 e c_2 diretamente, primeiro aplica-se uma decomposição a um dos

operandos, então calcula-se o produto entre essa decomposição e o outro criptograma, assim, em vez de $\mathsf{err}(c_1) \cdot \mathsf{err}(c_2)$, como no DGHV, obtém-se $\mathsf{decomp}(c_1) \cdot \mathsf{err}(c_2)$. Como a decomposição é constituída de valores pequenos, o ruído não aumenta muito.

Nas próximas seções, essa técnica será explicada em detalhes e ela será usada para construir a cifra conhecida como GSW.

1.5.1. Decomposição dos criptogramas

Considere o módulo q usado no (R)LWE. Fixe uma base B, defina $\ell := \lceil \log_B q \rceil$ e $\mathbf{g} = (B^0, B^1, ..., B^{\ell-1})$. Represente $a \in \mathbb{Z}_q$ por um inteiro entre $-q/2$ e $q/2$ e defina $g^{-1}(a)$ como o vetor $(a_0, ..., a_{\ell-1}) \in [\![-B, B]\!]^\ell$ representando a decomposição de $|a|$ na base B, mas multiplicada pelo sinal de a e com o bit ou a palavra menos significa à esquerda. Por exemplo, se $q = 2^7$ e $B = 4$, então $\ell = 4$. Portanto, $g^{-1}(5) = (1,1,0,0)$, $g^{-1}(-24) = (0,-2,-1,0)$ e $g^{-1}(-64) = (0,0,0,-1)$. O código a seguir implementa g^{-1}:

```
def inv_g_ZZ(a, B, q):
    a = sym_mod(ZZ(a), q) # definida em utils.sage
    l = ceil(log(q, B))
    return vector(a.digits(base=B, padto=l))
```

Note que multiplicar $g^{-1}(a)$ por $\mathbf{g}$ tem o efeito de combinar as palavras com as respectivas potências, o que resulta novamente em a, i.e., $g^{-1}(a) \cdot \mathbf{g} = \sum_{i=0}^{\ell-1} a_i B^i = a$. Por isso a notação g^{-1} é comumente usada.

Para decompor um polinômio $a(X)$, basta decompor cada coeficiente a_i, multiplicar por X^i, obtendo um vetor $(a_{i,0}X^i, ..., a_{i,\ell-1}X^i)$, e somar todos os vetores. O resultado é um vetor com ℓ polinômios cujos coeficientes pertencem a $[\![-B, B]\!]$. Como B é em geral pequeno, a norma do vetor resultante também o é. Mais explicitamente, abusamos da notação e definimos $g^{-1}(a) := \sum_{i=0}^{N-1} g^{-1}(a_i)X^i$ para $a \in R$, onde $g^{-1}(a_i)$ é a decomposição de inteiros. Note que $g^{-1}(a) \cdot \mathbf{g} = \sum_{i=0}^{N-1} g^{-1}(a_i) \cdot \mathbf{g} \cdot X^i = \sum_{i=0}^{N-1} a_i \cdot X^i = a$, logo, assim como para os inteiros, ao multiplicar a decomposição por $\mathbf{g}$, obtemos novamente a.

```
def inv_g_poly(a, B, q, n):
    l = ceil(log(q, B))
    result = vector(Zx, [0] * l)
    pow_x = 1
    for i in range(n):
        result += pow_x * inv_g_ZZ(a[i], B, q)
        pow_x *= x
    return result
```

Finalmente, estendemos essa decomposição para um vetor de polinômios simplesmente aplicando g^{-1} a cada coordenada do vetor e concatenando as decomposições, i.e., fixando-se uma dimensão d, para qualquer $\mathbf{c} \in R_q^d$, definimos

$$G^{-1}(\mathbf{c}) := (g^{-1}(c_0), ..., g^{-1}(c_{d-1})) \in R_q^{d\ell}.$$

Como anteriormente, queremos obter novamente $\mathbf{c}$ a partir de $G^{-1}(\mathbf{c})$. Como agora temos um vetor de dimensão $d\ell$, não podemos multiplicar por $\mathbf{g}$, cuja dimensão é ℓ.

Então, definimos a matriz $\mathbf{G} := \mathbf{I}_d \otimes \mathbf{g}^T \in \mathbb{Z}^{d\ell \times d}$, i.e., o produto tensorial entre a identidade $d \times d$ e o vetor-coluna $\mathbf{g}$ transposto. Note que esse produto é calculado substituindo cada entrade $a_{i,j}$ de $\mathbf{I}_d$ pelo vetor $a_{i,j} \cdot \mathbf{g}^T$, assim, temos que

$$\mathbf{G} = \begin{pmatrix} \mathbf{g} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{g} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{g} \end{pmatrix}.$$

Por exemplo, se $d = 2$, $q = 2^3$ e $B = 2$, então $\ell = 3$ e

$$\mathbf{G} = \begin{pmatrix} 1 & 0 \\ 2 & 0 \\ 4 & 0 \\ 0 & 1 \\ 0 & 2 \\ 0 & 4 \end{pmatrix}.$$

Observe o que ocorre ao multiplicar $G^{-1}(\mathbf{c})$ por $\mathbf{G}$: considerando a primeira coluna de $\mathbf{G}$, as ℓ primeiras componentes de $G^{-1}(\mathbf{c})$ são multiplicadas por $\mathbf{g}$ e as outras $(d-1)\ell$ componentes são multiplicadas por zero, logo, o resultado é $g^{-1}(c_0)\mathbf{g} = c_0$. Ao multiplicar $G^{-1}(\mathbf{c})$ pela segunda coluna de $\mathbf{G}$, obtém-se $g^{-1}(c_1)\mathbf{g} = c_1$, e assim sucessivamente. Ou seja, $G^{-1}(\mathbf{c})\mathbf{G} = (g^{-1}(c_0)\mathbf{g}, ..., g^{-1}(c_{\ell-1})\mathbf{g}) = \mathbf{c}$, como desejado.

Essa matriz $\mathbf{G}$ é conhecida como *gadget matrix* e a ideia principal do esquema homomórfico GSW é incluí-la nos criptogramas para que seja possível aplicar G^{-1} durante a multiplicação homomórfica, reduzindo assim o ruído inserido por cada produto.

1.5.2. GSW: uma cifra cuja evolução do ruído é apenas linear

Nesta seção, a cifra GSW [GSW13] será apresentado, mas usando basicamente o formato sugerido [DM15], portanto, baseado no problema RLWE em vez do LWE. Para evitar confusão com os parâmetros do problema LWE usado na cifra que será introduzida na seção 1.7.1, a chave secreta será denotada por z em vez de s e usaremos N em vez de n, ou seja, doravante, definimos $R := \mathbb{Z}[X]/\langle X^N + 1\rangle$, onde N é uma potência de 2.

O GSW pode ser vista como um compromisso entre a memória e o crescimento do ruído. Mais especificamente, cada criptograma é definido com $O(\log q)$ amostras RLWE em vez de apenas uma, portanto, usa-se mais memória, no entanto, o crescimento do ruído devido a cada produto homomórfico é quase linear, ou seja, ao multiplicar criptogramas c_0 e c_1, obtém-se um novo criptograma c tal que $\mathsf{erro}(c) = O(\mathsf{erro}(c_0) + \mathsf{erro}(c_1))$. Isso é possível pois, durante a multiplicação homomórfica, os ruídos dos criptogramas não são multiplicados entre si, ao contrário de outros esquemas, como o DGHV, onde o ruído do criptograma resultante tem um termo $r_1 \cdot r_2$.

As funções de geração de chave, ciframento e deciframento do GSW são mostradas em detalhe a seguir:

- $\mathsf{GSW.ParamGen}(1^\lambda)$: Escolha um valor real $\sigma > 0$ e inteiros N, B e ℓ, sendo N uma potência de dois. Defina $q := B^\ell$. Os parâmetros N, q e σ devem guarantir λ bits de segurança considerando o problema RLWE. Devolva $\mathsf{params} := (N, q, \sigma, B, \ell)$.

- $\mathsf{GSW.KeyGen}(\mathsf{params})$: Defina $z(x) = \sum_{i=0}^{N-1} z_i X^i$ com cada z_i amostrado uniformemente de $\{-1, 0, 1\}$. Devolva $\mathsf{sk} := z$.

- $\mathsf{GSW.Enc}(\mathsf{sk}, m, \mathsf{params})$: Para cifrar um polinômio $m \in R$ cujos coeficientes pertencem a $\mathbb{Z}_B$, amostre $(a_i, b_i) \leftarrow \mathscr{R}_{s,N,q,\sigma}$ para $0 \leq i < \ell$ e defina $\mathbf{C}' \in R_q^{2\ell \times 2}$ com cada linha i igual a (a_i, b_i). Note que $\mathbf{C}'$ é da forma $[\mathbf{a}, \mathbf{b}]$ com $\mathbf{b} = \mathbf{a} \cdot z + \mathbf{e} \pmod q$. Finalmente, devolva $\mathbf{C} = \mathbf{C}' + m \cdot \mathbf{G} \bmod q$.

- $\mathsf{GSW.Dec}(\mathsf{sk}, \mathbf{C}, \mathsf{params})$: Seja $(a, b) \in R_q^2$ a última linha de $\mathbf{C}$. Calcule $u := b - a \cdot \mathsf{sk} \in R_q$, interprete u como um polinômio em $\mathbb{Z}[X]$ e devolva $\lfloor B \cdot u / q \rceil \bmod B$.

Como a matriz $\mathbf{G}$ tem potências de B nas primeiras ℓ entradas da primeira coluna, e elas são multiplicadas por m e somadas às amostras RLWE, os criptogramas têm este formato distinto, em que as primeiras ℓ linhas guardam a mensagem na primeira coluna, ou seja, no "termo a" do RLWE, em vez de ser no "termo b":

$$\mathbf{C} = \begin{pmatrix} a_0 + B^0 \cdot m & a_0 \cdot z + e_0 \\ \vdots & \vdots \\ a_{\ell-1} + B^{\ell-1} \cdot m & a_{\ell-1} \cdot z + e_{\ell-1} \\ a_\ell & a_\ell \cdot z + e_\ell + B^0 \cdot m \\ \vdots & \vdots \\ a_{2\ell-1} & a_{2\ell-1} \cdot z + e_{2\ell-1} + B^{\ell-1} \cdot m \end{pmatrix} \in R_q^{2\ell \times 2}.$$

No deciframento, fica claro como os criptogramas têm muita informação redundante, pois apenas a linha $2\ell - 1$ é necessária para recuperar a mensagem. Essa linha pode ser escrita como $(a, b) \in R_q^2$ com $b = a \cdot z + e + B^{\ell-1} \cdot m$. Mas como $q = B^\ell$, temos $B^{\ell-1} = q/B$ e $u := b - a \cdot \mathsf{sk} = e + m \cdot q/B \pmod q$. Portanto, sobre os inteiros, existe um polinômio v tal que $u = e + m \cdot q/B + v \cdot q \in \mathbb{Z}[x]$. Em seguida, calcula-se

$$w := \left\lfloor \frac{B \cdot u}{q} \right\rceil = \left\lfloor \frac{B \cdot e}{q} + m + vB \right\rceil = \left\lfloor \frac{B \cdot e}{q} \right\rceil + m + vB,$$

onde a última igualdade vale porque todos os coeficientes de m e v são inteiros.

Este é o ponto que define o critério de corretude do deciframento. A saber, como o valor devolvido é $w \bmod B$, é preciso que $\lfloor B \cdot e/q \rceil$ seja zero para que o resultado seja exatamente $m \bmod B$. Mas para isso, é necessário que todos os coeficientes dessa fração sejam menores que meio, i.e., $\|B \cdot e/q\|_\infty < 1/2$, o que é equivalente a $\|e\|_\infty < q/(2B)$. Em outras palavras, se $\|e\|_\infty < q/(2B)$, então $\left\lfloor \frac{B \cdot e}{q} \right\rceil = 0$ e $\left\lfloor \frac{B \cdot u}{q} \right\rceil = m \pmod B$, logo, o deciframento devolve corretamente $m \bmod B$.

Portanto, para garantir a corretude do deciframento, é preciso garantir que o ruído não seja maior que essa fração de q. Perceba a semelhança com o esquema DGHV, apresentado na Seção 1.3, em que o ruído não pode ser maior que uma fração de p.

Antes de apresentar a implementação do GSW, precisamos incluir as seguintes funções no arquivo utils.sage definido na seção 1.3.1:

```
def round_poly(h):
  return Zx([round(hi) for hi in h.list()])

def infinity_norm(poly):
  if 0 == poly:
    return 0
  return vector(ZZ, poly.list()).norm(Infinity)

def infinity_norm_vec(vec):
  return max([infinity_norm(vi) for vi in vec])
```

Assim, o código em Sage que implementa esses procedimentos do GSW é apresentado a seguir. O arquivo *decompositions.sage* contém as funções mostradas na seção 1.5.1.

```
load("utils.sage")
load("decompositions.sage")
load("distribution_rlwe.sage")

class GSW:
  def __init__(self, n, q, sigma, B = 2):
    self.n, self.sigma, self.B = n, sigma, B
    f = x^n + 1
    self.l = ceil(log(q, B))
    self.q = B^self.l
    g = Matrix(ZZ, self.l, 1, [B^i for i in range(self.l)])
    I = Matrix.identity(2)
    self.G = I.tensor_product(g) # gadget matrix

    self.Rq = ( ZZ.quotient(q) )['x'].quotient(f)
    self.sk = self.keygen()
    self.dist_rlwe = RLWEDistribution(self.sk, n, q, sigma)

  def keygen(self):
    sk = 0
    while 0 == sk:
      sk = Zx([ZZ.random_element(-1, 2) for _ in range(self.n)])
    return sk

  def enc(self, m):
    Rq, l, sk = self.Rq, self.l, self.sk
    C = Matrix(Rq, 2*l, 2)
    for i in range(2*l):
      C[i] = self.dist_rlwe.sample()
    C += m * self.G
```

```
    return C

  def dec(self, C):
    B, l, q, sk = self.B, self.l, self.q, self.sk
    a = C[2*l - 1, 0]
    b = C[2*l - 1, 1] # == a*sk + e + (q/B) * m
    noisy_m = Zx((b - a*sk).lift())
    rounded_m = round_poly(B * noisy_m / q)
    return sym_mod_poly(rounded_m, B)
```

Algoritmo 1.3. Procedimentos básicos da cifra GSW

As operações homomórficas do GSW são definidas da seguinte forma:

- GSW.Add$(\mathbf{C}_0, \mathbf{C}_1)$: simplesmente adicione as entradas correspondentes, i.e., devolva $\mathbf{C}_{add} := \mathbf{C}_0 + \mathbf{C}_1 \in R_q^{2\ell \times 2}$.

- GSW.Mult$(\mathbf{C}_0, \mathbf{C}_1)$: decomponha $\mathbf{C}_0$ em base B e multiplique por $\mathbf{C}_1$ fazendo operações em R_q, i.e., devolva $\mathbf{C}_{mult} := G^{-1}(\mathbf{C}_0) \cdot \mathbf{C}_1 \in R_q^{2\ell \times 2}$.

O código em Sage que implementa as operações homomórficas do GSW é apresentado a seguir e deve ser inserido na definição da classe GSW apresentada no Algoritmo 1.3.

```
    def add(self, C0, C1):
        C_add = C0 + C1
        return C_add

    def inv_g_row_ciphertex(self, c):
        B, l, n = self.B, self.l, self.n
        a, b = c[0], c[1]
        res = vector(self.Rq, [0] * 2 * l)
        res[0 : l] = inv_g_poly(a, B, l, n)
        res[l : 2*l] = inv_g_poly(b, B, l, n)
        return res

    def mult(self, C0, C1):
        result = Matrix(self.Rq, 2*self.l, 2)
        for i in range(2*self.l):
            decomp = self.inv_g_row_ciphertex(C0[i])
            prod_in_Rq = decomp * C1
            result[i] = prod_in_Rq
        return result
```

Analisar a adição homomórfica é trivial. Considerando que os operandos são da

forma $\mathbf{C}_i = [\mathbf{a}_i, \mathbf{b}_i] + m_i\mathbf{G}$, com $\mathbf{b}_i = \mathbf{a}_i \cdot z + \mathbf{e}_i \in R_q^{2\ell}$, temos

$$\begin{aligned}\mathbf{C}_{add} &= [\mathbf{a}_0 + \mathbf{a}_1,\ \mathbf{b}_0 + \mathbf{b}_1] + (m_0 + m_1)\mathbf{G} \\ &= [\underbrace{\mathbf{a}_0 + \mathbf{a}_1}_{\mathbf{a}_{add}},\ (\mathbf{a}_0 + \mathbf{a}_1) \cdot z + \underbrace{\mathbf{e}_0 + \mathbf{e}_1}_{\mathbf{e}_{add}}] + (m_0 + m_1)\mathbf{G}.\end{aligned}$$

Logo, vemos que $\mathbf{C}_{add} = [\mathbf{a}_{add},\ \mathbf{a}_{add} \cdot z + \mathbf{e}_{add}] + (m_0 + m_1)\mathbf{G}$, ou seja, um criptograma válido que cifra a soma das mensagens. Além disso, o ruído cresce apenas aditivamente, i.e., $\mathsf{erro}(\mathbf{C}_{add}) \leq \mathsf{erro}(\mathbf{C}_0) + \mathsf{erro}(\mathbf{C}_1)$.

Analisar a multiplicação homomórfica é ligeiramente mais complicado. Primeiro, note que cada operando $\mathbf{C}_i$ é uma matriz $2\ell \times 2$ e que a decomposição expande cada linha com 2 elementos transformando-as em linhas com 2ℓ elementos, ou seja, $G^{-1}(\mathbf{C}_0)$ é uma matriz $2\ell \times 2\ell$, portanto, o produto $G^{-1}(\mathbf{C}_0) \cdot \mathbf{C}_1$ está bem definido e resulta em uma matriz $2\ell \times 2$, ou seja, com as dimensões de um criptograma válido.

Agora, note que sobre R_q, temos

$$\mathbf{C}_{mult} = G^{-1}(\mathbf{C}_0) \cdot ([\mathbf{a}_1,\ \mathbf{b}_1] + m_1\mathbf{G}) = [G^{-1}(\mathbf{C}_0) \cdot \mathbf{a}_1,\ G^{-1}(\mathbf{C}_0) \cdot \mathbf{b}_1] + m_1 G^{-1}(\mathbf{C}_0) \cdot \mathbf{G}.$$

Definindo $\mathbf{a}' := G^{-1}(\mathbf{C}_0) \cdot \mathbf{a}_1$ e notando que $\mathbf{b}_1 = \mathbf{a}_1 \cdot s + \mathbf{e}_1$, vemos que

$$\mathbf{C}_{mult} = [\mathbf{a}',\ \mathbf{a}' \cdot z + G^{-1}(\mathbf{C}_0) \cdot \mathbf{e}_1] + m_1 G^{-1}(\mathbf{C}_0) \cdot \mathbf{G}.$$

Além disso, como o produto entre a decomposição de $\mathbf{C}_0$ e $\mathbf{G}$ resulta novamente em $\mathbf{C}_0$, temos $m_1 G^{-1}(\mathbf{C}_0) \cdot \mathbf{G} = m_1([\mathbf{a}_0,\ \mathbf{b}_0] + m_0\mathbf{G}) = [m_1\mathbf{a}_0,\ m_1\mathbf{a}_0 \cdot z + m_1\mathbf{e}_0] + m_1 m_0 \mathbf{G}$, portanto,

$$\mathbf{C}_{mult} = [\mathbf{a}' + m_1\mathbf{a}_0,\ (\mathbf{a}' + m_1\mathbf{a}_0) \cdot z + G^{-1}(\mathbf{C}_0) \cdot \mathbf{e}_1 + m_1\mathbf{e}_0] + m_1 m_0 \cdot \mathbf{G}. \tag{1}$$

Assim, vemos que $\mathbf{C}_{mult} = [\mathbf{a}_{mult},\ \mathbf{a}_{mult} \cdot z + \mathbf{e}_{mult}] + m_0 m_1 \mathbf{G}$, onde $\mathbf{a}_{mult} := \mathbf{a}' + m_1\mathbf{a}_0$ e $\mathbf{e}_{mult} := G^{-1}(\mathbf{C}_0) \cdot \mathbf{e}_1 + m_1\mathbf{e}_0$, ou seja, é um criptograma válido e realmente cifra o produto das mensagens.

Agora fica claro o porquê de o ruído crescer apenas de forma quase aditiva, pois o produto $\mathbf{e}_0 \cdot \mathbf{e}_1$ não aparece em $\mathbf{e}_{mult}$. Para analisar como o ruído cresce, ou seja, quão grande é a norma de $\mathbf{e}_{mult}$ em comparação às normas de $\mathbf{e}_0$ e $\mathbf{e}_1$, primeiro precisamos do seguinte fato básico – sua prova é facilmente derivada usando o produto do vetor de coeficientes pela matriz anticirculante, ambos definidos na seção 1.7.6.1. Denotando a norma infinita por $\|\cdot\|$, temos:

Lema 1.5.1. *Sejam f e g elementos de $R := \mathbb{Z}[X]/\langle X^N + 1\rangle$. Então, $\|f \cdot g\| \leq N\|f\|\|g\|$.*

Usando esse lema, é fácil ver que se $\mathbf{A} \in R^{d \times \ell}$, $\mathbf{v} \in R^\ell$ e $\mathbf{u} := \mathbf{A} \cdot \mathbf{v}$, então $\|\mathbf{u}\| \leq N\ell\|\mathbf{A}\| \cdot \|\mathbf{v}\|$, pois $\|\mathbf{u}\| = \max\{\|u_0\|, ..., \|u_{d-1}\|\}$ e cada coordenada u_i satisfaz

$$\|u_i\| = \left\| \sum_{j=0}^{\ell-1} a_{i,j} v_j \right\| \leq \sum_{j=0}^{\ell-1} \left\| a_{i,j} v_j \right\| \leq \sum_{j=0}^{\ell-1} N \left\| a_{i,j} \right\| \left\| v_j \right\| \leq N\ell \|\mathbf{A}\| \cdot \|\mathbf{v}\|.$$

Assim, podemos ver que

$$\begin{aligned}\|\mathbf{e}_{mult}\| &\leq \left\|G^{-1}(\mathbf{C}_0)\cdot\mathbf{e}_1\right\| + \|m_1\mathbf{e}_0\| && \text{(desigualdade triangular)}\\ &\leq 2N\ell\left\|G^{-1}(\mathbf{C}_0)\right\|\|\mathbf{e}_1\| + \|m_1\mathbf{e}_0\| &&\\ &\leq 2NB\ell\|\mathbf{e}_1\| + \|m_1\mathbf{e}_0\| && \text{(pois } \left\|G^{-1}(\mathbf{C}_0)\right\| \leq B).\end{aligned}$$

Geralmente, os coeficientes de m_1 são pequenos, por exemplo, menores que B, e B é uma constante pequena. Nesses casos, temos $\|m_1\mathbf{e}_0\| \leq NB\|\mathbf{e}_0\|$. Mas em vários cenários, como no *bootstrapping* apresentado na Seção 1.7.2, as mensagens cifradas pertencem à $\{-1,0,1\}$ ou são monômios da forma $\pm 1\cdot X^k$. Nesses casos, temos $\|m_1\mathbf{e}_0\| = \|\mathbf{e}_0\|$ e o crescimento do ruído é ainda menor.

Uma observação importante é que m_0 não aparece no ruído de $\mathbf{C}_{mult}$, portanto, ele não depende da mensagem cifrada pelo operando a esquerda, assim, se for sabido que a mensagem de um dos operandos é possivelmente maior que a do outro, pode-se sempre utilizá-lo como o termo $\mathbf{C}_0$. Essa assimetria é importante durante o *bootstrapping*, pois em um dado momento, é preciso multiplicar homomorficamente a mensagem $\lfloor q/8 \rceil$, que é extremamente grande (em relação a q).

Note também que é importante colocar a esquerda o criptograma que tem o maior ruído, ou seja, que já passou por mais operações homomórficas, pois assim seu ruído é multiplicado apenas pela mensagem do outro criptograma, que normalmente, é pequena. Por exemplo, imagine que $\mathbf{C}$ é um criptograma "fresco", i.e., devolvido por $\mathsf{GSW.Enc}$ e que não passou por qualquer operação homomórfica. Considere que $\mathbf{C}$ cifra $m = 1$ com ruído $\mathbf{e}$. Então, temos $\mathsf{erro}(\mathbf{C}) = \|\mathbf{e}\| \approx \sigma$. Além disso, considere um criptograma $\mathbf{C}'$ obtido após uma multiplicação homomórfica envolvendo dois criptogramas frescos e suponha que $\mathbf{C}'$ cifra $m' = 1$ com ruído $\mathbf{e}'$. Como visto acima, temos $\mathsf{erro}(\mathbf{C}') = \|\mathbf{e}'\| \approx 2NB\ell\sigma$. Então, a magnitude do ruído de $\mathsf{GSW.Mult}(\mathbf{C}', \mathbf{C})$ é

$$\left\|G^{-1}(\mathbf{C}')\mathbf{e} + m\mathbf{e}'\right\| \leq 2NB\ell\sigma + \left\|\mathbf{e}'\right\| \approx 4NB\ell\sigma,$$

no entanto, o ruído de $\mathsf{GSW.Mult}(\mathbf{C}, \mathbf{C}')$ é

$$\left\|G^{-1}(\mathbf{C})\mathbf{e}' + m'\mathbf{e}\right\| \leq 2NB\ell\left\|\mathbf{e}'\right\| + \|\mathbf{e}\| \approx (2NB\ell)^2 + \sigma.$$

É fácil também analisar a evolução do ruído ao se efetuar uma sequência de produtos do tipo $\prod_{i=0}^{k} m_i$. Nosso primeiro objetivo é estudar o formato do ruído.

Lema 1.5.2. *Seja $k \geq 1$ um inteiro. Para $0 \leq i \leq k$, seja $\mathbf{C}_i \in R_q^{2\ell\times 2}$ um criptograma que cifra uma mensagem $m_i \in R_B$ com ruído $\mathbf{e}_i$. Defina $\mathbf{C}'_0 := \mathbf{C}_0$ e, para $1 \leq i \leq k-1$, defina $\mathbf{C}'_{i+1} := \mathsf{GSW.Mult}(\mathbf{C}'_i, \mathbf{C}_{i+1})$. Note que $\mathbf{C}'_1$ cifra $m_0 \cdot m_1$, $\mathbf{C}'_2$ cifra $m_0 \cdot m_1 \cdot m_2$, e assim sucessivamente. Então, o ruído de $\mathbf{C}'_k$ é*

$$\mathbf{e}'_k = \mathbf{e}_0 \prod_{i=1}^{k} m_i + \sum_{i=0}^{k-1} G^{-1}(\mathbf{C}'_i)\mathbf{e}_{i+1}\left(\prod_{j=i+2}^{k} m_j\right). \tag{2}$$

Prova. Podemos provar por indução. O caso base, $k = 1$, é trivial, pois como mostrado na Equação 1, temos $\mathbf{e}'_1 = G^{-1}(\mathbf{C}'_0)\cdot\mathbf{e}_1 + m_1\mathbf{e}'_0$, que é idêntica à Equação 2.

Agora, suponha que a Equação 2 vale para um valor arbitrário $k-1$.

$$\mathbf{e}'_{k-1} = \mathbf{e}_0 \prod_{i=1}^{k-1} m_i + \sum_{i=0}^{k-2} G^{-1}(\mathbf{C}'_i)\mathbf{e}_{i+1} \left(\prod_{j=i+2}^{k-1} m_j \right).$$

Como $\mathbf{e}'_k$ é o ruído de $\mathsf{GSW.Mult}(\mathbf{C}'_{k-1}, \mathbf{C}_k)$, temos que

$$\begin{aligned}
\mathbf{e}'_k &= G^{-1}(\mathbf{C}'_{k-1})\mathbf{e}_k + m_k \mathbf{e}'_{k-1} && \text{(Equação 1)} \\
&= G^{-1}(\mathbf{C}'_{k-1})\mathbf{e}_k + m_k \left(\mathbf{e}_0 \prod_{i=1}^{k-1} m_i + \sum_{i=0}^{k-2} G^{-1}(\mathbf{C}'_i)\mathbf{e}_{i+1} \left(\prod_{j=i+2}^{k-1} m_j \right) \right) && \text{(Hipótese indutiva)} \\
&= G^{-1}(\mathbf{C}'_{k-1})\mathbf{e}_k + \mathbf{e}_0 \prod_{i=1}^{k} m_i + \sum_{i=0}^{k-2} G^{-1}(\mathbf{C}'_i)\mathbf{e}_{i+1} \left(\prod_{j=i+2}^{k} m_j \right) \\
&= \mathbf{e}_0 \prod_{i=1}^{k} m_i + \sum_{i=0}^{k-1} G^{-1}(\mathbf{C}'_i)\mathbf{e}_{i+1} \left(\prod_{j=i+2}^{k} m_j \right)
\end{aligned}$$

onde a última igualdade vale pois $\prod_{j=k-1+2}^{k} m_j = 1$, logo o termo $G^{-1}(\mathbf{C}'_{k-1})\mathbf{e}_k$ a esquerda pode ser incluído como o $(k-1)$-ésimo termo somatório. □

Usando esse lema, pode-se finalmente mostrar como o ruído cresce com uma sequência de produtos:

Lema 1.5.3. *Usando a mesma notação do Lema 1.5.2, considere $\mathbf{C}'_k$, que cifra $\prod_{i=0}^{k} m_k$ com ruído $\mathbf{e}'_k$. Então,*

$$\left\|\mathbf{e}'_k\right\| \leq \left\| \mathbf{e}_0 \prod_{i=1}^{k} m_i \right\| + \sum_{i=0}^{k-1} 2NB\ell \left\| \mathbf{e}_{i+1} \left(\prod_{j=i+2}^{k} m_j \right) \right\|. \tag{3}$$

Prova. Usando o Lema 1.5.2, temos

$$\begin{aligned}
\left\|\mathbf{e}'_k\right\| &\leq \left\| \mathbf{e}_0 \prod_{i=1}^{k} m_i \right\| + \sum_{i=0}^{k-1} \left\| G^{-1}(\mathbf{C}'_i)\mathbf{e}_{i+1} \left(\prod_{j=i+2}^{k} m_j \right) \right\| && \text{(Desigualdade triangular)} \\
&\leq \left\| \mathbf{e}_0 \prod_{i=1}^{k} m_i \right\| + \sum_{i=0}^{k-1} 2N\ell \left\| G^{-1}(\mathbf{C}'_i) \right\| \cdot \left\| \mathbf{e}_{i+1} \left(\prod_{j=i+2}^{k} m_j \right) \right\| \\
&\leq \left\| \mathbf{e}_0 \prod_{i=1}^{k} m_i \right\| + \sum_{i=0}^{k-1} 2NB\ell \left\| \mathbf{e}_{i+1} \left(\prod_{j=i+2}^{k} m_j \right) \right\|
\end{aligned}$$

□

Finalmente, considere $S := \{-1, 0, 1\} \cup \{-X^1, ..., -X^{N-1}\} \cup \{X^1, ..., X^{N-1}\}$. Se todas as mensagens m_i pertencerem a S (como ocorre no *bootstrapping*), então todos os produtos de mensagens também pertencem a S, pois serão sempre iguais a zero ou a $\pm 1 \cdot X^k$ para algum k, logo, ao serem reduzidas módulo $X^N + 1$, obtém-se $\pm 1 \cdot X^{k \bmod N}$.

Além disso, para qualquer $e \in R$ e $m \in S$, o produto $m \cdot e$ é zero ou um polinômio com os mesmos coeficientes que e mas possivelmente em outra ordem e multiplicados por -1. Por exemplo, se $n = 4$, $e = 2X^3 - X$ e $m = X^2$, então, $e \cdot m = 2X^5 - X^2 = -X^2 - 2X \pmod{X^N + 1}$. Portanto, $\|e\| = \|e \cdot m\|$. Com isso, tem-se o seguinte corolário:

Corolário 1.6. *Considere a mesma notação do Lema 1.5.2 e assuma que $m_i = 0$ ou $m = \pm 1 \cdot X^k$ para algum $0 \leq k < n$. Além disso, suponha que o ruído $\mathbf{e}_i$ de cada criptograma é limitado por um valor β. Então, $\mathbf{C}'_k$ cifra $\prod_{i=0}^{k} m_k$ com ruído $\mathbf{e}'_k$ e*

$$\left\|\mathbf{e}'_k\right\| \leq \beta + 2kNB\ell\beta. \tag{4}$$

Perceba a diferença em relação ao DGHV: se todos os criptogramas tiverem ruído menor que um valor β, uma multiplicação homomórfica com o DGHV aumenta o ruído para $O(\beta^2)$, a segunda multiplicação o aumenta para $O(\beta^3)$, e assim sucessivamente. Então, avaliar $\prod_{i=0}^{k} m_i$ homomorficamente gera um criptograma com ruído $O(\beta^k)$, ou seja, o crescimento é exponencial em k, enquanto o ruído com o GSW aumenta apenas linearmente em k, i.e., de β para $O(k\beta)$.

1.6.1. Comparando as comparações

Nesta seção, implementaremos a comparação homomórfica de números inteiros, discutiremos o crescimento do ruído e compararemos com a Seção 1.3.2, que usa o DGHV para implementar essa comparação homomórfica.

A única operação homomórfica usada no Algoritmo 1.2 que não está implementada na classe GSW é a negação, mas ela pode ser implementada usando a mesma lógica: basta somar o valor 1. No entanto, no caso do GSW, uma encriptação do 1 trivial e sem ruído não é apenas o inteiro 1, mas sim $1 \cdot \mathbf{G}$. Logo, o seguinte código deve ser incluído na definição da classe GSW:

```
    def not_gate(self, C):
        return C + self.G
```

Note que essa porta lógica funciona módulo 2, portanto, para utilizá-la, é preciso instanciar o GSW com o parâmetro $B = 2$.

Para acompanhar o crescimento do ruído na prática, definiremos também uma função que devolve o logaritmo da norma do ruído. Como um criptograma do GSW é definido como $\mathbf{C} = [\mathbf{a}\ ,\ \mathbf{b}] + m \cdot \mathbf{G}$ e o ruído está presente no termo $\mathbf{b}$, i.e., $\mathbf{b} = \mathbf{a} \cdot z + \mathbf{e}$, primeiro subtraímos $m \cdot \mathbf{G}$ para recuperar $\mathbf{a}$ e $\mathbf{b}$, então calculamos $\log(\left\|[\mathbf{b} - \mathbf{a} \cdot z]_q\right\|)$, como no código a seguir, que também deve ser incluído na definição da classe GSW:

```
  def get_noise(self, C, msg):
    l, sk = self.l, self.sk
    C -= msg * self.G
    a = vector(C[:, 0])
    b = vector(C[:, 1]) # == a*sk + e (mod q)
    e = b - a*sk
    e = sym_mod_vec(e, self.q) # definida em utils.sage
    norm_e = infinity_norm_vec(e) # definida em utils.sage
```

```
    if norm_e == 0: # log de zero não está definido
      return -1

    return log(norm_e, 2).n()
```

Uma vez definida a porta *not* e essa função que devolve o ruído, é trivial converter o Algoritmo 1.2: basta substituir o objeto do tipo DGHV por um do tipo GSW, inicializar o criptograma c com a matriz $1 \cdot G$ em vez de simplesmente 1, i.e., `c = gsw.G`, e remover a redução módulo 2 ao calcular m, ou seja, `m *= bits0[i] + bits1[i] + 1`.

Para fins de teste, podemos negligenciar a segurança e escolher um valor de N pequeno, por exemplo, para comparar inteiros com L bits, podemos instanciar um objeto do tipo GSW como `gsw = GSW(n=8, q=3*L, sigma=3.2, B=2)`. Execute o algoritmo algumas vezes e observe como o ruído evolui durante a avaliação homomórfica da comparação. Veja se o ruído do criptograma produzido no fim da comparação tem ruído próximo de $\log q$ – lembre-se que o ruído não pode passar o limiar $q/(2B) = q/4$, senão a decifração não funcionará corretamente. Note que o algoritmo primeiro cria o criptograma `cmp_i` adicionando dois criptogramas frescos e avaliando a porta lógica not. A adição aumenta o ruído para aproximadamente 2σ e a porta lógica não altera o ruído, portanto, o ruído de `cmp_i` é pequeno. No entanto, o criptograma `c` é o resultado de várias multiplicações homomórficas, logo, seu ruído é grande. Por isso, como discutido anteriormente, deve-se usar a assimetria do GSW e colocar `c` a esquerda na multiplicação, i.e., deve-se usar `c = gsw.mult(c, cmp_i)`. Para verificar na prática como essa assimetria é importante, troque essa linha por `c = gsw.mult(cmp_i, c)` e veja como o ruído cresce muito mais rapidamente e faz com que a decifração deixe de funcionar.

Observe que cada criptograma `cmp_i` cifra uma mensagem da forma $m_i = a_i + b_i + 1$, com a_i e b_i binários, logo, m_i pode ser no máximo 3, o que é um valor pequeno. No entanto, como essas mensagens são multiplicadas, o valor $\prod_{i=0}^{k} m_i$ pode ser tão grande quanto 3^k no pior caso, i.e., quando $a_i = b_i = 1$ para $0 \leq i \leq k$. Mas como foi provado no Lema 1.5.3, o ruído final depende do valor da mensagem, portanto, no pior caso, ele será dominado por 3^k e acabará sendo exponencial no número de bits comparados homomorficamente. Mais especificamente, usando o Lema 1.5.3, vemos que o ruído final pode ser aproximadamente $3^k\sigma(1+4kN\log q)$. Como ele deve ser menor que $q/4$ para garantir a corretude da decifração, aplicando o logaritmo, temos a seguinte desigualdade

$$\log(q/4) \geq k\log(3) + \log(\sigma) + \log(1+4kN\log q).$$

Como σ é uma constante pequena e $N = \Theta(\lambda)$, ignorando o fator $\log\log q$, obtemos

$$\log q = \Theta(k + \log(k\lambda)).$$

Finalmente, como cada criptograma é uma matriz de dimensões $2\log q \times 2$, cujas entradas são polinômios de grau menor que n e coeficientes com $\log q$ bits, vemos que o tamanho de cada criptograma necessário para comparar homomorficamente inteiros com k bits é

$$\Theta(N\log^2 q) = \Theta(\lambda\log^2 q) = \Theta(\lambda(k+\log(k\lambda))^2) = \Theta(\lambda k^2 + \lambda k\log(k\lambda)).$$

Ou seja, cada criptograma tem tamanho essencialmente linear no parâmetro de segurança λ e quadrático no número de bits k.

Já no caso do DGHV, o ruído inicial de cada criptograma é $2^{\rho} \approx 2^{\lambda}$, o ruído final é $2^{k\rho} \approx 2^{k\lambda}$ e ele precisa ser menor que $p \approx 2^{\eta}$, o que nos dá $\eta = \Theta(k\lambda)$. O número de bits de cada criptograma é

$$\gamma = \Theta(\lambda(\rho - \eta)^2) = \Theta(\lambda(k\lambda)^2) = \Theta(\lambda^3 k^2),$$

ou seja, para comparar homomorficamente inteiros de k bits, no pior caso, com o DGHV, é preciso manipular criptogramas com tamanho quadrático em k e cúbico no parâmetro de segurança! Obviamente, $\Theta(\lambda^3 k^2)$ é muito pior que $\Theta(\lambda k^2)$.

Evitando que a mensagem cresça

Note que se a mensagem não crescesse a ponto de ser exponencial em k, mais especificamente, 3^k, o ruído final produzido pelo GSW seria muito menor, podendo ser apenas logarítmico em k. Esse é um problema comum com o GSW: em geral, é preciso evitar que as mensagens cresçam muito. Uma abordagem possível seria expressar todas as operações que podem aumentar o tamanho da mensagem em termos de portas lógicas do tipo NAND (AND negado), pois elas garantem que a mensagem será sempre binária, já que $\mathsf{NAND}(a,b) := 1 - a \cdot b \in \{0,1\}$. Lembre-se de que o NAND é uma porta lógica universal, portanto, todo circuito pode ser expresso apenas por ela.

Por exemplo, no Algoritmo 1.2, a operação $a + b + 1$ na verdade corresponde a um XNOR (XOR negado), e é ela que permite que a mensagem cifrada cresça de 1 para 3, então, pode-se substituí-la por uma sequência de portas NAND usando o fato de que

$$\mathsf{XNOR}(a,b) = \mathsf{NAND}(\mathsf{NAND}(a,b), \mathsf{NAND}(\mathsf{NAND}(a,a), \mathsf{NAND}(b,b))).$$

A tradução do NAND para a versão homomórfica é literal, i.e., multiplica-se os dois criptogramas e subtrai-se o resultado do valor 1, que corresponde a $1 \cdot \mathbf{G}$:

$$\mathsf{GSW.Nand}(\mathbf{C}_0, \mathbf{C}_1) := \mathbf{G} - \mathsf{GSW.Mult}(\mathbf{C}_0, \mathbf{C}_1).$$

1.6.2. O *bootstrapping* do esquema GSW

Como apresentado até então, a cifra GSW não é completamente homomórfica, pois ela só pode avaliar circuitos de profundidade limitada. No entanto, como demonstrado no Lema 1.5.3, o ruído cresce lentamente com o GSW, principalmente se as mensagens forem binárias e se mantiverem pequenas com a avaliação homomórfica, como quando o circuito é expresso usando a porta NAND.

Portanto, ao contrário da cifra DGHV, é possível avaliar circuitos com qualquer profundidade L, não apenas $\log \lambda$. Logo, dado L, a profundidade desejada, pode-se escolher os parâmetros q, N, etc, de tal forma que seja possível avaliar todos os circuitos de profundidade L. Por isso, o esquema GSW que obtemos até então é chamado de *cifra completamente homomórfica por nível*[8]. Note a diferença: em uma cifra completamente homomórfica, é possível escolher um conjunto de parâmetros para o qual todos os circuitos podem ser computados homomorficamente, ou seja, pode-se instanciar o esquema

[8]Do inglês *levelled homomorphic encryption scheme.*

e depois avaliar qualquer função sobre os dados cifrados. Já com uma cifra homomórfica por nível, é preciso primeiro decidir as funções que serão calculadas, depois instanciar a cifra usando parâmetros que permitam avaliar tais funções homomorficamente.

Para transformar o GSW em uma cifra completamente homomórfica no sentido estrito (não por nível), como discutido seção 1.3.3, basta que ele seja capaz de avaliar seu próprio circuito de decifração e, de fato, ele o é: basta identificar a profundidade L desse circuito e escolher parâmetros que permitam avaliar circuitos de profundidade $L+k$, para alguma constante pequena k que permita que outras operações úteis sejam feitas entre os *bootstrappings*. Note que não é necessário publicar informação auxiliar sobre a chave secreta para simplificar o circuito de decifração, como foi feito para o DGHV na seção 1.3.4. Portanto, o *bootstrapping* é mais simples, direto e não usa hipóteses de segurança adicionais além da segurança circular.

No entanto, apesar de ser possível e ser muito mais simples do que no caso do DGHV, expressar a função de decifração como um grande circuito binário e avaliar cada porta lógica homomorficamente faz com que o *bootstrapping* seja extremamente lento. Assim, nas seções seguintes, em vez de aplicar o *bootstrapping* ao GSW, o utilizaremos para aplicar o *bootstrapping* a um esquema mais simples, obtendo assim o esquema mais rápido e eficiente que existe atualmente.

1.7. *Bootstrapping in vitro*

Nesta seção, mostraremos como o *bootstrapping* pode ser avaliado eficientemente. Para isso, primeiro definiremos uma cifra homomórfica simples, chamada cifra de base, que pode avaliar apenas uma porta NAND, e então utilizaremos o GSW para decifrar homomorficamente os criptogramas dessa cifra de base.

1.7.1. A cifra homomórfica mais simples possível

Nesta seção, seguiremos a abordagem de [DM15] e utilizaremos o problema LWE para construir uma cifra homomórfica por nível extremamente simples: Ela cifra um bit no nível 1, então pode-se avaliar homomorficamente uma única porta NAND, gerando um criptograma no nível 0, que precisa então ser recifrado com o *bootstrapping* para voltar ao nível 1 e reduzir o ruído. Essa cifra é definida como a seguir. Para simplificar a exposição, assumiremos que o módulo q é um múltiplo de 8, mas pode ser facilmente eliminada:

- $\mathsf{HE.ParamGen}(1^\lambda)$: Escolha os parâmetros n, q e σ do problema LWE de tal forma que se obtenha λ bits de segurança e que $q \in 8\mathbb{Z}$. Devolva $\mathsf{params} := (n, q, \sigma)$.

- $\mathsf{HE.KeyGen}(\mathsf{params})$: Amostre $\mathbf{s}$ uniformemente de $\{0,1\}^n$. Devolva $\mathsf{sk} := \mathbf{s}$.

- $\mathsf{HE.Enc}(\mathsf{sk}, m)$: Para cifrar um bit m em um criptograma no nível 1, amostre $(\mathbf{a}, b') \leftarrow \mathscr{A}_{\mathbf{s},q,\sigma}$, defina $b := b' + (q/4)m \bmod q$ e devolva $c = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$.

- $\mathsf{HE.Dec}(\mathsf{sk}, c)$: Para decifrar um criptograma $\mathbf{c} = (\mathbf{a}, b)$ do nível 1, calcule $c' = b - \mathbf{a} \cdot \mathbf{s} \bmod q$ e devolva $\left[\left\lfloor \frac{4c'}{q} \right\rceil\right]_2$.

Traduzir esses procedimentos para a linguagem Sage é trivial, vide o código a seguir:

```
load("distribution_lwe.sage")

class LWEScheme:
  def __init__(self, n, q, sigma=3.2):
    self.n, self.q = n, q
    self.keygen()
    self.dist_lwe = LWEDistribution(self.sk, q, sigma)

  def keygen(self):
    n = self.n
    bin_list = [ZZ.random_element(0, 2) for _ in range(n)]
    self.sk = vector(ZZ, bin_list)

  def enc(self, m):
    a, _b = self.dist_lwe.sample()
    b = _b + round(self.q / 4) * m
    return [a, b]

  def dec(self, c):
    q, s = self.q, self.sk
    noisy_m = ZZ(c[1] - c[0]*s) # == e + (q / 4)*m + u*q
    return round(4 * noisy_m / q) % 2
```

Algoritmo 1.4. Procedimentos básicos da cifra de base

A segurança desse esquema segue diretamente da versão decisional do problema LWE, pois ela garante que a amostra LWE $(\mathbf{a}, b')$ é indistinguível de um vetor uniformemente aleatório em $\mathbb{Z}_q^{n+1}$, portanto, $(\mathbf{a}, b') + (\mathbf{0}, \mu) \bmod q$ continua sendo indistinguível de um vetor uniforme para qualquer $\mu \in \mathbb{Z}_q$, em particular, para $\mu = (q/4) \cdot m$, como usado na encriptação.

Para verificar a corretude da decifração, note que $c' := b - \mathbf{as} = e + (q/4)m + uq$ para algum $u \in \mathbb{Z}$. Logo,

$$\left\lfloor \frac{4c'}{q} \right\rceil = \left\lfloor \frac{4e}{q} + m + 4q \right\rceil = \left\lfloor \frac{4e}{q} \right\rceil + m + 4q$$

e precisamos que o termo $\lfloor 4e/q \rceil$ seja zero, o que ocorre se $4|e|/q < 1/2$, ou seja, se $|e| < q/8$. Em outras palavras, se $|e| < q/8$, então $[\lfloor 4c'/q \rceil]_2 = [m + 4q]_2 = m$ e a mensagem correta é devolvida.

Apesar da decifração exigir apenas que o ruído seja menor que $q/8$, para que a porta lógica NAND apresentada a seguir funcione corretamente, é necessário que $|e| < q/16$, por isso, será sempre exigido que os criptogramas desse esquema respeitem essa inequação. Para enfatizar esse fato, temos a definição a seguir:

Definição 1.7.1. *Sejam* $\mathbf{c} = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, $\mathbf{s} \in \{0,1\}^n$ *e* $m \in \{0,1\}$. *Dizemos que* $\mathbf{c}$ *é uma* encriptação válida *de m sob a chave secreta* $\mathbf{s}$ *se existe* $e \in \mathbb{Z}$ *tal que* $b = \mathbf{a} \cdot \mathbf{s} + e + (q/4)m$ *e* $|e| < q/16$.

Assim, é preciso garantir que tanto os criptogramas devolvidos pela função Enc quanto o s obtidos após o bootstrapping sejam *válidos*, pois isso garantirá que qualquer função possa ser avaliada homomorficamente, já que, dado um criptograma válido, será possível aplicar a porta NAND e em seguida o *bootstrapping*, que produzirá novamente um criptograma válido.

O NAND homomórfico é obtido com simples adições e subtrações vetoriais, i.e., coordenada a coordenada, como mostrado a seguir:

- $\mathsf{HE.Nand}(\mathbf{c}_0, \mathbf{c}_1)$: Sejam $\mathbf{c}_0$ e $\mathbf{c}_1$ dois criptogramas no nível 1. Devolva

$$\mathbf{c} := (\mathbf{0}, 5q/8) - \mathbf{c}_0 - \mathbf{c}_1 \in \mathbb{Z}_q^{n+1}.$$

Para entender como essa porta lógica homomórfica funciona, note que o criptograma que ela produz é da forma $(\mathbf{a}, b)$ com $\mathbf{a} = -\mathbf{a}_0 - \mathbf{a}_1$ e

$$b = 5q/8 - b_0 - b_1 = \mathbf{a} \cdot \mathbf{s} - e_0 - e_1 + 5q/8 - (q/4)(m_0 + m_1),$$

ou seja, b já contém o termo $\mathbf{a}$ multiplicado pela chave secreta, como esperado. O que não está claro é que ele cifra realmente $\mathsf{NAND}(m_0, m_1) = 1 - m_0 \cdot m_1$. Para ver que esse é realmente o caso, lembre-se de que ambos m_0 e m_1 são binários, logo $m_i^2 = m_i$, portanto, $(m_0 - m_1)^2 = m_0 - 2m_0m_1 + m_1$. Então,

$$(q/4)(m_0 + m_1) = (q/4)((m_0 - m_1)^2 + 2m_0m_1) = (q/4)(m_0 - m_1)^2 + (q/2)m_0m_1.$$

Note agora que, como $5q/8 = q/8 + q/2$, temos

$$5q/8 - (q/4)(m_0 + m_1) = q/8 - (q/4)(m_0 - m_1)^2 + (q/2)(1 - m_0m_1).$$

Portanto, o termo b do criptograma devolvido por $\mathsf{HE.Nand}$ é da forma

$$b = \mathbf{a} \cdot \mathbf{s} + e + (q/2)(1 - m_0 \cdot m_1),$$

como esperado. Seu ruído é então $e := -e_0 - e_1 + q/8 - (q/4)(m_0 - m_1)^2$. Note que a mensagem é exatamente $\mathsf{NAND}(m_0, m_1)$ e que ela é multiplicada por $q/2$ em vez de $q/4$, portanto, temos um criptograma no nível zero.

Para analisar o tamanho desse ruído, perceba que $(m_0 - m_1)^2 \in \{0,1\}$, pois ambas as mensagens são binárias, logo, $q/8 - (q/4)(m_0 - m_1)^2 \in \{q/8, -q/8\}$, portanto, podemos reescrever e como $-e_0 - e_1 \pm q/8$. Como $|e_i| < q/16$ por hipótese, temos

$$|e| < q/16 + q/16 + q/8 = q/4.$$

Resumindo, se ambos $\mathbf{c}_0$ e $\mathbf{c}_1$ forem criptogramas válidos, então $\mathsf{HE.Nand}(\mathbf{c}_0, \mathbf{c}_1)$ devolve um criptograma que cifra $\mathsf{NAND}(m_0, m_1)$ com ruído menor que $q/4$ e no *nível*

zero. Note que esse ruído é muito próximo de q, por isso, não é possível aplicar mais nenhuma operação homomórfica. Para prosseguir avaliar as próximas portas lógicas do circuito, precisamos executar o *bootstrapping*, como veremos nas próximas seções.

A seguir, apresenta-se o código em Sage que implementa o NAND homomórfico. Ele deve ser incluído na definição da classe LWEScheme apresentada no Algoritmo 1.4.

```
def nand(self, c0, c1):
  a0, b0 = c0
  a1, b1 = c1
  a = -a0 - a1
  b = round(5 * self.q / 8) - b0 - b1
  return [a, b]
```

1.7.2. Usando GSW para implementar o *bootstrapping*

Após o NAND homomórfico, obtém-se um criptograma $\mathbf{c} := (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$ no qual $b = \mathbf{a}\cdot\mathbf{s} + e + (q/2)m$ e o ruído e satisfaz $|e| < q/4$. O objetivo do *bootstrapping* é decifrar $\mathbf{c}$ homomorficamente e obter um criptograma válido, segundo a definição 1.7.1. Então, o primeiro passo é definir a função de deciframento que será avaliada pela cifra GSW, por isso, considere a seguinte função:

$\mathsf{Dec}_{\mathbf{s}}(\mathbf{a}, b)$: calcule $c' := [q/4 + b - \mathbf{a}\cdot\mathbf{s}]_q$. Devolva 0 se $c' < q/2$ ou 1 caso contrário.

Note que essa função devolve corretamente m, pois $c' = q/4 + e + (q/2)m$, e, como $-q/4 < e < q/4$, vemos que $0 < q/4 + e < q/2$. Portanto, se $m = 0$, então $c' = q/4 + e < q/2$, mas se $m = 1$, então $c' = q/4 + e + q/2$ e vemos que $q/2 < c' < q$.

Então, o primeiro passo do *bootstrapping* é calcular o produto escalar $\mathbf{a}\cdot\mathbf{s}$ e para fazê-lo, temos criptogramas GSW cifrando cada coordenada s_i de $\mathbf{s}$. Esse conjunto de criptogramas é chamado de *chave do bootstrapping* e denotada por bk. Assim, utilizando bk, poderíamos facilmente usar as operações homomórficas do GSW para calcular $\mathsf{GSW.Enc}(\mathbf{a}\cdot\mathbf{s})$ e então $\mathsf{GSW.Enc}(q/4 + b - \mathbf{a}\cdot\mathbf{s})$, no entanto, o segundo passo envolveria avaliar homomorficamente um circuito que compara com $q/2$ e isso seria muito caro.

A principal observação do bootstrapping apresentado em [DM15] é que a comparação $c' < q/2$ é trivial se, em vez de $\mathsf{GSW.Enc}(c')$, calcularmos $\mathsf{GSW.Enc}(X^{c'})$, ou seja, uma encriptação de X elevado a $q/4 + b - \mathbf{a}\cdot\mathbf{s}$ mod q. Note que para isso, basta ser capaz de usar bk e a_i para calcular $\mathsf{GSW.Enc}(X^{-a_i\cdot s_i})$, pois uma vez que esses criptogramas são gerados, pode-se multiplicá-los homomorficamente para gerar

$$\prod_{i=0}^{n-1} \mathsf{GSW.Enc}(X^{-a_i\cdot s_i}) = \mathsf{GSW.Enc}(X^{-\sum_{i=0}^{n-1} a_i\cdot s_i}) = \mathsf{GSW.Enc}(X^{-\mathbf{a}\cdot\mathbf{s}}),$$

e finalmente, basta multiplicar por $X^{q/4+b}$ para obter o criptograma desejado.

Em [DM15], a técnica apresentada para gerar cada $\mathsf{GSW.Enc}(X^{-a_i\cdot s_i})$ envolve $\Theta(\log q)$ produtos homomórficos, portanto, o primeiro passo do *bootstrapping* requer $\Theta(n \log q)$ multiplicações do GSW. Mas os autores de [CGGI16] mostraram que é possível obter $\mathsf{GSW.Enc}(X^{-a_i\cdot s_i})$ com apenas uma multiplicação se supormos que a chave

secreta $\mathbf{s}$ é binária, portanto, reduziram o custo do *bootstrapping* para $\Theta(n)$ multiplicações homomórficas. Além disso, eles introduziram o "produto externo" homomórfico, que é mais rápido que o produto original do GSW por um fator de $\Theta(\log q)$, portanto, obtiveram um *bootstrapping* cerca de $\log^2(q)$ vezes mais rápido que o de [DM15].

Uma vez calculado $\mathsf{GSW.Enc}(X^{q/4+b-\mathbf{a}\cdot\mathbf{s}}) = \mathsf{GSW.Enc}(X^{e'+(q/2)m})$, o segundo passo consiste em transformar esse criptograma em uma encriptação de $\mathsf{Enc}(m)$, o que é obtido por meio de uma transformação relativamente simples, que será explicada na seção 1.7.6. Essa transformação recebe um criptograma cifrado com o problema RLWE e devolve um novo criptograma baseado no problema LWE, como requerido pelo esquema de base, no entanto, ela não gera um criptograma cifrado sob a chave $\mathbf{s}$ do esquema de base, mas sim sob a chave z da cifra GSW, ou, mais especificamente, sob uma chave $\mathbf{z} := (z_0, ..., z_{N-1}) \in \mathbb{Z}^N$, ou seja, uma chave definida como o vetor de coeficientes de z. Para finalmente obter um criptograma válido (segunda a definição 1.7.1), o último passo é aplicar um procedimento chamado *substituição de chaves*[9], com o qual é possível substituir a chave $\mathbf{z}$ pela chave $\mathbf{s}$.

Essas três etapas do *bootstrapping* são ilustradas na figura 1.4. A primeira etapa recebe um criptograma com ruído próximo do limite aceitável pela decifração e a chave de *bootstrapping* bk, que tem pouco ruído. Cada operação homomórfica efetuada durante o *bootstrapping* acumula ruído sobre o ruído já presente em bk, então a quantidade de ruído contida no criptograma recifrado, i.e., devolvido pelo *bootstrapping*, é igual ao ruído inicial de bk mais o ruído adicionado pelas operações. Para que o *bootstrapping* funcione, é preciso que esse ruído final seja menor do que o ruído contido no criptograma de entrada. Mais especificamente, é preciso garantir que o ruído do criptograma devolvido pelo *bootstrapping* seja menor que $q/16$, pois esse é o limite para a magnitude do ruído de um criptograma válido, segundo a definição 1.7.1.

1.7.3. Produto externo

Em [CGGI16], foi introduzida uma multiplicação homomórfica entre um criptograma usual sob o problema RLWE, ou seja, um vetor $(a,b) \in R^2$, e um criptograma do GSW, ou seja, uma matriz $\mathbf{C} \in R^{2\ell\times 2}$. Ela é efetuada decompondo a amostra RLWE e multiplicando por $\mathbf{C}$, ou seja, multiplicando um vetor por uma matriz, o que gera novamente uma amostra RLWE, ou seja, um vetor. Esse produto foi chamado de *produto externo*. A principal observação é que as linhas de um criptograma do GSW são amostras RLWE, então a multiplicação homomórfica do GSW já é composta por uma série de produtos externos. Além disso, como um criptograma GSW é redundante, isto é, ele armazena mais informação do que o necessário para decifrar a mensagem, já que apenas uma linha é usada para a deciframento, então efetuar uma série de produtos externos e obter apenas uma amostra RLWE no fim, é suficiente para implementar o *bootstrapping*, já que um dos últimos passos do *bootstrapping* de [DM15] consistia em extrair uma linha do criptograma GSW.

Primeiramente, precisamos definir um criptograma RLWE:

Definição 1.7.2. *Dizemos que* $\mathbf{c} := (a,b) \in R_q^2$ *é um* criptograma RLWE *que cifra uma mensagem* $m \in \{0,1\}$ *com ruído* $e \in R$ *e sob uma chave* $z \in R$ *se* $b = a\cdot z + e + (q/8)m \in R_q$.

[9] Do inglês *key switching*.

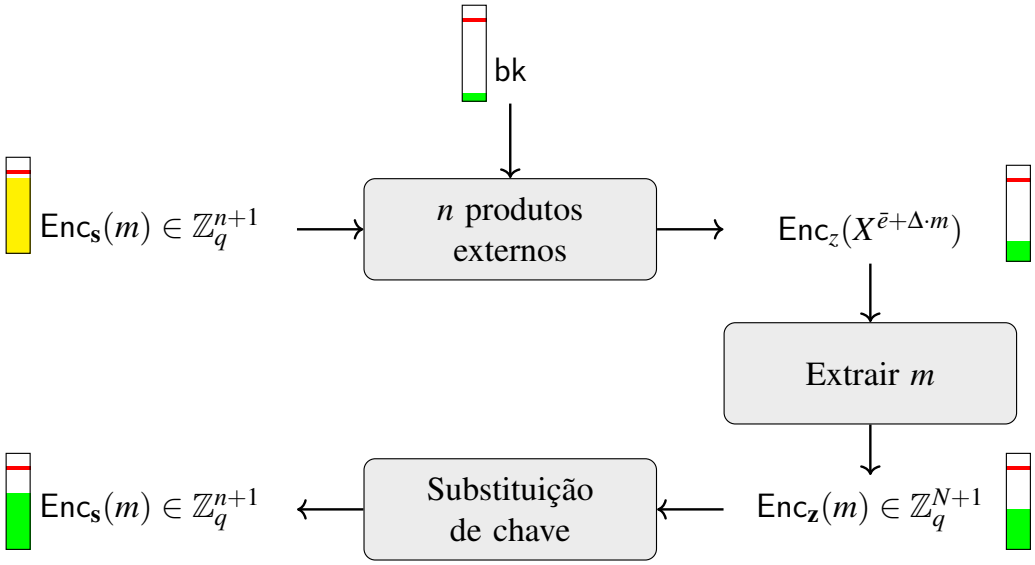


Figure 1.4. Passo a passo da recifração do esquema de base com o GSW. Os retângulos representam o ruído contido nos criptogramas e a linha vermelha representa o limiar que o ruído pode atingir antes de a decifração falhar.

Em vez multiplicar a mensagem por $q/8$, a definição poderia usar qualquer outro valor Δ maior do que a magnitude do ruído para que o deciframento funcionasse, no entanto, o valor $q/8$ é escolhido aqui porque o criptograma obtido no fim da sequência de n produtos externos deve ser transformado em um criptograma válido do esquema de base e, como veremos na seção 1.7.6, esse valor facilita a transformação.

Finalmente, o produto externo é definido como segue:

Definição 1.7.3. *O produto externo entre um criptograma* RLWE $\mathbf{c} \in R_q^2$ *e um criptograma* GSW $\mathbf{C} \in R^{2\ell\times 2}$ *é um criptograma* RLWE $\mathbf{c}'$ *calculado como*

$$\mathbf{c}' := G^{-1}(\mathbf{c}) \cdot \mathbf{C} \in R_q^2.$$

Note que a multiplicação homomórfica do GSW corresponde a 2ℓ produtos externos, onde $\ell = O(\log q)$, logo, cada produto externo é $O(\log q)$ vezes mais barato que uma multiplicação do GSW. Além disso, a corretude e o crescimento do ruído são iguais aos dos produtos do GSW. Como anteriormente, temos $G^{-1}(\mathbf{c}) \cdot \mathbf{G} = \mathbf{c}$, logo, considerando que $\mathbf{c}$ cifra m_0 e $\mathbf{C}$ cifra m_1, o produto externo satisfaz

$$\begin{aligned}
\mathbf{c}' &= G^{-1}(\mathbf{c}) \cdot ([\mathbf{a},\ \mathbf{b}] + m_1\mathbf{G}) \\
&= [G^{-1}(\mathbf{c}) \cdot \mathbf{a},\ G^{-1}(\mathbf{c}) \cdot \mathbf{b}] + m_1 G^{-1}(\mathbf{c}) \cdot \mathbf{G} \\
&= [G^{-1}(\mathbf{c}) \cdot \mathbf{a},\ G^{-1}(\mathbf{c}) \cdot \mathbf{b}] + m_1 \cdot [a, b] \\
&= [a',\ G^{-1}(\mathbf{c}) \cdot \mathbf{b} + m_1 \cdot b] && (a' := G^{-1}(\mathbf{c}) \cdot \mathbf{a} + m_1 \cdot a) \\
&= [a',\ a'z + \underbrace{G^{-1}(\mathbf{c}) \cdot \mathbf{e} + m_1 \cdot e}_{e'} + (q/8)m_0 \cdot m_1]
\end{aligned}$$

Note que o ruído e' é essencialmente o mesmo que o ruído gerado por uma multiplicação do GSW, como mostrado na equação 1, portanto, uma sequência de n produtos

externos aumenta o ruído assim como mostrado no lema 1.5.3 e no corolário 1.6.

O código a seguir implementa o produto externo e deve ser incluído na definição da classe GSW, apresentada no código 1.3.

```
  def extern_prod(self, c, C):
    decomp = self.inv_g_row_ciphertex(c)
    return decomp * C
```

1.7.4. Geração da chave de *bootstrapping*

A chave de *bootstrapping* é um conjunto de criptogramas GSW, em que cada um cifra uma entrada s_i da chave secreta $\mathbf{s}$ do esquema de base. Então, definimos

$$\mathsf{bk} := \{\mathsf{bk}_i := \mathsf{GSW.Enc}(s_i) : 0 \leq i < n\}.$$

Com a classe GSW definida no código 1.3, a geração de bk pode ser feita simplesmente como `bk = [gsw.enc(s[i]) for i in range(n)]`, mas para facilitar a implementação do *bootstrapping*, agruparemos a geração de bk e outras funções relacionadas na classe `Bootstrapper` definida a seguir:

```
# -*- coding: utf-8 -*-
load("lwe_base_scheme.sage")
load("GSW.sage")
load("utils.sage")

class Bootstrapper:
  # gsw: cifra usada como acumulador
  # lwe_base_scheme: gera criptogramas a serem recifrados
  def __init__(self, _gsw, lwe_scheme):
    assert(_gsw.q == lwe_scheme.q)
    self.gsw = _gsw
    self.base_scheme = lwe_scheme
    self.one = self.gsw.G
    self.boot_key_gen()
    self.key_swt_gen() # a ser definido

  def boot_key_gen(self):
    s = self.base_scheme.sk # vetor binário de dimensão n
    n = len(s)
    self.bk = [self.gsw.enc(s[i]) for i in range(n)]
```

Algoritmo 1.5. Definição da classe *Bootstrapper*.

1.7.5. Calculando $\mathsf{Enc}(X^{\bar{e}+\Delta\cdot m})$

Como explicado anteriormente, a primeira etapa do *bootstrapping* recebe um criptograma do esquema de base, $\mathbf{c} = (\mathbf{a}, b) \in \mathbb{Z}_q^n$ e usa a chave bk para calcular um criptograma RLWE de $X^{b-\mathbf{a}\cdot\mathbf{s}}$. No entanto, é preciso que o cálculo feito no expoente seja realizado módulo

q, mas a ordem de X em R é $2N$, que não necessariamente é igual a q. Isto é, como as operações em R são calculadas módulo X^N+1, então $X^N=-1$ em R e $X^{2N}=(X^N)^2=1$. Sendo assim, todas as operações realizadas no expoente de X são feitas módulo $2N$. Por exemplo, se multiplicarmos X^{N+2} com X^{N+5} em R, obteremos $X^{2N+7}=X^{2N}\cdot X^7=X^7$.

Poderíamos restringir a escolha de parâmetros para garantir que q fosse igual a $2N$, no entanto, isso tornaria o *bootstrapping* menos flexível. Em vez disso, antes de começarmos a operar com $\mathbf{c}$, primeiro o multiplicamos por $2N/q$, o que muda o módulo de q para $2N$, ou seja, definimos

$$\mathbf{c}' := (\mathbf{a}', b') := \left(\left\lfloor \frac{2N\mathbf{a}}{q} \right\rceil, \left\lfloor \frac{2Nb}{q} \right\rceil \right).$$

Note que existe $u \in \mathbb{Z}$ tal que $b = \mathbf{a}\cdot\mathbf{s} + e + (q/2)m + u\cdot q \in \mathbb{Z}$, logo,

$$2Nb/q = (2N\mathbf{a}/q)\cdot\mathbf{s} + 2Ne/q + Nm + u\cdot 2N.$$

Além disso, como para todo $r \in \mathbb{R}$ existe um $|\varepsilon| \leq 1/2$ tal que $\lfloor \alpha \rceil = \alpha + \varepsilon$, temos que

$$\lfloor 2Nb/q \rceil = \lfloor 2N\mathbf{a}/q \rceil \cdot \mathbf{s} + \underbrace{2Ne/q + \varepsilon\cdot\mathbf{s} + \varepsilon'}_{e'} + Nm + u\cdot 2N.$$

Ou seja, se o erro e originalmente em $\mathbf{c}$ era menor do que $q/4$, então agora o erro e' em $\mathbf{c}'$ é praticamente igual a $2Ne/q$, logo, menor que $(2N/q)\cdot(q/4) = N/2$ (com grande probabilidade). Assim, $\mathbf{c}' = (\mathbf{a}', b')$ cifra m módulo $2N$. Além disso, se $\mathbf{c}$ é decifrável módulo q, então $\mathbf{c}'$ também o é, mas sobre $\mathbb{Z}_{2N}$, exatamente como necessário para o *bootstrapping*.

O código a seguir implementa essa operação e deve ser incluído na classe *Bootstrapper* definida no algoritmo 1.5.

```
# Recebe um criptograma LWE (a, b) em Z_q^(n+1) e devolve
# outro criptograma LWE que cifra a mesma mensagem, mas
# é definido módulo 2*N
def mod_switch_to_2N(self, a, b):
  q, N = self.base_scheme.q, self.gsw.n
  _a = vector([round(ZZ(ai) * 2 * N / q) for ai in a])
  _b = round(ZZ(b) * 2*N / q)
  return _a, _b
```

Agora, dado $\mathbf{c}' = (\mathbf{a}', b')$ e bk, nosso primeiro objetivo é obter uma encriptação RLWE de $X^{-a'_i\cdot s_i}$, para $0 \leq i < n$. Para isso, note que como s_i é binário, então substituindo $s_i = 0$ e $s_i = 1$ dos dois lados da seguinte igualdade, vemos que ela é válida:

$$X^{-a'_i\cdot s_i} = 1 + (X^{-a'_i} - 1)\cdot s_i.$$

Em [CGGI16], a operação que produz uma encriptação de $X^{-a'_i\cdot s_i}$ a partir de a'_i e de uma encriptação GSW de s_i foi chamada de cmux. Para calculá-la homomorficamente, substitui-se o 1 por $\mathbf{G}$ e s_i por bk_i na expressão anterior, logo, temos o seguinte:

$$\mathsf{cmux}(a'_i, \mathsf{bk}_i) := \mathbf{G} + (X^{-a'_i} - 1)\cdot \mathsf{bk}_i \in R_q^{2\ell\times\ell}. \tag{5}$$

Note que como bk_i é da forma $[\mathbf{a}, \mathbf{b}] + s_i\mathbf{G}$, temos

$$\begin{aligned}\mathsf{cmux}(a'_i, \mathsf{bk}_i) &= \mathbf{G} + (X^{-a'_i} - 1) \cdot [\mathbf{a}, \mathbf{b}] + (X^{-a'_i} - 1)s_i\mathbf{G} \\ &= \mathbf{G} + [\mathbf{a}', \mathbf{b}'] + (X^{-a'_i} - 1)s_i\mathbf{G} \\ &= [\mathbf{a}', \mathbf{b}'] + (1 + (X^{-a'_i} - 1)s_i)\mathbf{G} \\ &= [\mathbf{a}', \mathbf{b}'] + X^{-a'_i \cdot s_i} \cdot \mathbf{G}.\end{aligned}$$

Ou seja, $\mathsf{cmux}(a'_i, \mathsf{bk}_i)$ é uma encriptação GSW de $X^{-a'_i \cdot s_i}$, como desejado. Além disso, se $\mathbf{e}$ é o ruído contido em bk_i, então o ruído de $\mathsf{cmux}(a'_i, \mathsf{bk}_i)$ é igual a $\mathbf{e}' := (1 + (X^{-a'_i} - 1) \cdot \mathbf{e}$. Assim sendo, vemos que $\|\mathbf{e}'\| \leq \|\mathbf{e}\| + \left\|X^{-a'_i} \cdot \mathbf{e}\right\| + \|-\mathbf{e}\| = 3\|\mathbf{e}\|$. Portanto, o cmux praticamente não aumenta o ruído.

Para completar o primeiro passo do *bootstrapping*, usamos o produto interno para multiplicar $X^{N/2+b'}$ por todas as encriptações de $X^{-a'_i \cdot s_i}$. Primeiramente, note que $(0, (q/8) \cdot X^{N/2+b'}) \in R_q^2$ é um criptograma RLWE trivial, pois satisfaz a Definição 1.7.2 com ambos a e e iguais a zero. Assim, podemos usar o produto externo para multiplicar $(0, (q/8) \cdot X^{N/2+b'})$ por $\mathsf{cmux}(a'_0, \mathsf{bk}_0)$ e obter uma encriptação RLWE de $X^{N/2+b'-a'_0 \cdot s_0}$, que pode então ser multiplicada por $\mathsf{cmux}(a'_1, \mathsf{bk}_1)$ com outro produto externo, o que gera uma encriptação de $X^{N/2+b'-a'_0 \cdot s_0 - a'_1 \cdot s_1}$, e assim sucessivamente. Ou seja, a primeira etapa do *bootstrapping* consiste em calcular

$$(0, (q/8) \cdot X^{N/2+b'}) \prod_{i=0}^{n} \mathsf{cmux}(a'_i, \mathsf{bk}_i),$$

onde cada multiplicação é efetuada com um produto externo.

É fácil ver que os expoentes são adicionados módulo $2N$. Portanto, como $b' = \mathbf{a}' \cdot \mathbf{s} + e' + Nm \bmod 2N$, o resultado obtido é um criptograma RLWE da forma $(a, b) \in R_q^2$ com $b = a \cdot z + e + (q/8) \cdot X^y$ e

$$y = N/2 + b' - \mathbf{a}' \cdot \mathbf{s} = \underbrace{N/2 + e'}_{\bar{e}} + Nm \pmod{2N}$$

Lembre-se que $-N/2 < e' < N/2$, logo, tomando $\bar{e} := N/2 + e'$, temos $0 < \bar{e} < N$. Essa inequação é fundamental para a transformação usada no segundo passo do *bootstrapping*.

Em suma, para implementar essa primeira etapa do *bootstrapping*, podemos adicionar o seguinte código à classe *Bootstrapper*:

```
  #  acc: criptograma RLWE cifrando alguma mensagem X^z
  #  a_i: um inteiro módulo 2*N
  #  bk_i: um criptograma GSW cifrando um bit s_i.
  #  Devolve um criptograma RLWE cifrando X^(z - a_i * s_i)
  def cmux_gate(self, acc, a_i, bk_i):
    N = self.gsw.n
    C = self.one + (x^(2*N-a_i) - 1) * bk_i
```

```
    acc = self.gsw.extern_prod(acc, C)
    return acc

  #   Recebe c = (a, b) \in Z_{2N}^{n+1}, i.e., um criptograma LWE
  # definido módulo 2*N e que cifra uma mensagem m sob a chave s.
  #   Devolve um criptograma RLWE cifrando X^(e_bar + N*m)
  # sob a chave z do esquema GSW.
  def linear_part_dec(self, c):
    Q, N, RQ = self.gsw.q, self.gsw.n, self.gsw.Rq
    a, b = c
    acc = [RQ(0), round(Q/8) * RQ(x)^(N // 2 + b)] # a = e = 0
    for i in range(len(a)):
      acc = self.cmux_gate(acc, a[i], self.bk[i])
    return acc
```

1.7.6. Transformando $X^{\bar{e}+N\cdot m}$ em m

Após a sequência de n produtos externos, obtém-se um criptograma RLWE c da forma $c=(a,b)\in R_q^2$ com $b=a\cdot z+e+(q/8)\cdot X^{\bar{e}+N\cdot m}\in R_q$, com $0<\bar{e}<N$. O segundo passo do *bootstrapping* consiste em transformar c em um criptograma $\mathbf{c}$ que cifra m e é baseado no problema LWE em vez do RLWE. Essa transformação consiste basicamente de um produto escalar entre o vetor de coeficientes de b e o vetor $\mathbf{u}=(-1,...,-1)\in\mathbb{Z}^N$. Para entender como ela funciona, primeiro é necessário entender como adições e multiplicações em R podem calculadas usando operações entre vetores e matrizes.

1.7.6.1. Matrizes anticirculantes e vetores de coeficientes

Seja $g=\sum_{i=0}^{N-1} g_i\cdot X^i$ um elemento do anel R, definimos o vetor de coeficientes de g como $\phi(g)=(g_0,...,g_{N-1})\in\mathbb{Z}^N$. É fácil ver que se $g,h\in R$, então $\phi(g+h)=\phi(g)+\phi(h)$, logo, os vetores de coeficientes permitem que as adições em R sejam representadas por somas de vetores. No entanto, as multiplicações não são tão simples e exigem também o que chamamos de matrizes anticirculantes: Para todo $g\in R$, definimos $\Phi(g)$ como a matriz $N\times N$ tal que cada linha $i=0,...,N-1$ é igual a $\phi(x^i\cdot g \bmod x^N+1)$. Note que como $x^N=-1$ em R, então $x\cdot g=g_0\cdot X+...+g_{N-2}\cdot X^{N-1}-g_{N-1}\in R$, portanto, $\phi(x\cdot g)=(-g_{N-1},g_0,g_1,...,g_{N-2})$, ou seja, é uma rotação cíclica de $\phi(g)$, mas com o último coeficiente multiplicado por -1. Em geral, $\phi(x^i\cdot g)$ multiplica as últimas i posições de $\phi(g)$ por -1 e rotaciona todas as entradas em i posições. Logo, temos que a matriz anticirculante de g é igual a

$$\Phi(g)=\begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_{N-1} \\ -g_{N-1} & g_0 & g_1 & \cdots & g_{N-2} \\ -g_{N-2} & -g_{N-1} & g_0 & \cdots & g_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -g_1 & -g_2 & -g_3 & \cdots & g_0 \end{pmatrix}\in\mathbb{Z}^{n\times n}.$$

Agora, é fácil representar o produto de polinômios módulo X^N+1 usando de

ϕ e Φ. Primeiro, note que $\phi(X^i)$ é um vetor com um único 1 na i-ésima posição, i.e., $\phi(X^i) = (0,0,...,0,1,0,...,0)$. Então, $\phi(X^i)\Phi(h)$ é igual a i-ésima linha de $\Phi(h)$, que por definição, é $\phi(X^i h)$, ou seja, $\phi(X^i)\cdot\Phi(h) = \mathrm{lin}_i(\Phi(h)) = \phi(X^i h)$. Com isso, pode-se provar o seguinte lema:

Lema 1.7.1. *Para todo $g,h \in R$, temos que $\phi(gh) = \phi(g)\Phi(h)$*

Proof. Como ϕ é aditivo, é fácil ver que $\phi(g) = \sum_{i=0}^{N-1}\phi(g_i X^i) = \sum_{i=0}^{N-1} g_i\phi(X^i)$. Assim,

$$\phi(g)\Phi(h) = \sum_{i=0}^{N-1} g_i\phi(X^i)\Phi(h) = \sum_{i=0}^{N-1} g_i\phi(X^i h) = \sum_{i=0}^{N-1}\phi(g_i X^i h) = \phi\left(\sum_{i=0}^{N-1} g_i X^i h\right) = \phi(gh)$$

□

Essas funções podem ser implementadas em Sage como a seguir:

```
# Devolve vetor que representa g mod X^N + 1.
def poly_to_vec(g, N):
  f = x^N+1
  v = (g % f).coefficients(sparse=False)
  return vector(v + [0]*(N - len(v)))

# Dado a \in R, devolve matriz A tal que para todo h \in R
#     poly_to_vec(h) * A = poly_to_vec(h*a).
def poly_to_mat(a, N):
  A = Matrix(ZZ, [poly_to_vec(a*x^i, N) for i in range(N)])
  return A
```

1.7.6.2. A transformação RLWE $\rightarrow$ LWE

Agora, usando vetores de coeficientes e matrizes anticirculantes, podemos finalmente mostrar como efetuar o segundo passo do *bootstrapping*. Primeiro note que usando o lema 1.7.1, é fácil provar o seguinte:

Corolário 1.8. *Seja $b = a\cdot z + e + (q/8)\cdot X^j \in R_q$, então, $\phi(b) = \phi(z)\cdot\Phi(a) + \phi(e) + (q/8)\cdot\phi(X^j)$.*

A principal observação agora é que o criptograma obtido após os produtos externos cifra $X^{\bar{e}+Nm}$ com $0 < \bar{e} < N$. Portanto, temos

$$m = 0 \implies X^{\bar{e}+Nm} = X^{\bar{e}}$$

e

$$m = 1 \implies X^{\bar{e}+Nm} = -1\cdot X^{\bar{e}}.$$

Isso significa que $\phi(X^{\bar{e}+Nm})$ é um vetor da forma $(0,..,0,1,0,...,0)$ se $m = 0$ e da forma $(0,..,0,-1,0,...,0)$ se $m = 1$. Mas então, se definirmos $\mathbf{u} = (1,1,...,1) \in \mathbb{Z}^N$, temos

$$m = 0 \implies \phi(X^{\bar{e}+Nm})\cdot\mathbf{u} = -1 = 2\cdot m - 1$$

e

$$m = 1 \implies \phi(X^{\bar{e}+Nm}) \cdot \mathbf{u} = 1 = 2 \cdot m - 1.$$

Ou seja, como $m \in \{0,1\}$, a equação $\phi(X^{\bar{e}+Nm}) \cdot \mathbf{u} = 2 \cdot m - 1$ é sempre válida.

Assim, dado o criptograma (a,b) com $b = a \cdot z + e + (q/8)X^{\bar{e}+Nm} \in R_q$, temos o seguinte sobre $\mathbb{Z}_q$:

$$\begin{aligned}
\phi(b) \cdot \mathbf{u} &= \phi(z) \cdot \Phi(a) \cdot \mathbf{u} + \mathbf{e} \cdot \mathbf{u} + (q/8)\phi(X^{\bar{e}+Nm}) \cdot \mathbf{u} && \text{(corolário 1.8)} \\
&= \mathbf{a}' \cdot \phi(z) + \mathbf{e} \cdot \mathbf{u} + (q/8)\phi(X^{\bar{e}+Nm}) \cdot \mathbf{u} && (\mathbf{a}' := \Phi(a) \cdot \mathbf{u}) \\
&= \mathbf{a}' \cdot \mathbf{z} + \mathbf{e} \cdot \mathbf{u} + (q/8)\phi(X^{\bar{e}+Nm}) \cdot \mathbf{u} && (\mathbf{z} := \phi(z)) \\
&= \mathbf{a}' \cdot \mathbf{z} + \mathbf{e} \cdot \mathbf{u} + (q/8) \cdot (2m-1).
\end{aligned}$$

Lembre-se que em vez de $2m-1$, queremos obter uma encriptação de m. Além disso, em vez de $q/8$, a mensagem precisa ser multiplicada por $q/4$ para ser um criptograma válido segundo a definição 1.7.1. Então, o que resta fazer é simplesmente adicionar $q/8$ à $\phi(b) \cdot \mathbf{u}$. Assim, obtém-se:

$$\phi(b) \cdot \mathbf{u} + q/8 = \mathbf{a}' \cdot \mathbf{z} + \mathbf{e} \cdot \mathbf{u} + (q/4) \cdot m.$$

Em suma, dado o criptograma (a,b) com $b = a \cdot z + e + (q/8)X^{\bar{e}+Nm} \in R_q$, o segundo passo do *bootstrapping* devolve $(\mathbf{a}', b') \in \mathbb{Z}_q^{N+1}$ onde

$$\mathbf{a}' := \Phi(a) \cdot \mathbf{u} \in \mathbb{Z}_q^N \quad \text{e} \quad b' := \phi(b) \cdot \mathbf{u} + q/8 \in \mathbb{Z}_q.$$

E vemos que $b' = \mathbf{a} \cdot \mathbf{z} + e' + (q/4)m \in \mathbb{Z}_q$ com $e' := \mathbf{e} \cdot \mathbf{u}$. Em outras palavras, $(\mathbf{a}', b')$ cifra m sob a chave $\mathbf{z}$.

Essa etapa do *bootstrapping* pode ser implementada da seguinte forma em Sage, considerando que este código é adicionado à definição da classe *Bootstrapper*. Note que ele usa as funções definidas no código 1.7.6.1, que deve então ser importado.

```
  def convert_rlwe_lwe(self, acc):
    N, Q = self.gsw.n, self.gsw.q
    a, b = Zx(acc[0].lift()), Zx(acc[1].lift())
    u = vector([-1] * N)

    lwe_a = (poly_to_mat(a, N) * u) % Q
    lwe_b = poly_to_vec(b, N) * u
    lwe_b = (lwe_b + round(Q/8)) % Q

    return [lwe_a, lwe_b] # in Z_Q^(N+1)
```

1.8.1. Substituição de chaves

Com as duas primeiras etapas do *bootstrapping*, já se pode transformar um criptograma $\mathbf{c}$ do esquema de base, no nível zero e com muito ruído, em um criptograma $\mathbf{c}'$ também baseado no problema LWE, no nível um e com pouco ruído. No entanto, $\mathbf{c}$ é cifrado sob a

chave secreta $\mathbf{s}$, mas $\mathbf{c}'$ usa a chave $\mathbf{z} := \phi(z)$, onde z é a chave secreta cifra GSW. Além disso, $\mathbf{c}$ é definido usando o problema LWE com dimensão n, enquanto $\mathbf{c}'$ tem dimensão N. Portanto, não é possível aplicar a porta NAND homomórfica a $\mathbf{c}'$ e criptogramas do esquema de base.

Uma forma de resolver esse problema é escolher $N = n$ e $\mathbf{s} = \phi(z)$. Porém, essa abordagem restringe muito a escolha de parâmetros e a eficiência do *bootstrapping*, pois, em geral, N precisa ser um valor grande para garantir a segurança (e.g., os artigos [DM15, CGGI16, Per21a] usam $N = 1024$), enquanto n pode ser escolhido com um valor muito menor, digamos, cerca de 600. Como a quantidade de produtos externos e de criptogramas em bk são lineares em n, é desejável escolher $\mathbf{s}$ e z de forma independente.

Então, assumindo que $\mathbf{s} \neq \phi(z)$, é preciso de alguma forma transformar $\mathbf{c}'$ em uma encriptação da mesma mensagem, mas sob a chave $\mathbf{s}$ e sem aumentar muito o ruído. Isso é feito com um procedimento chamado *substituição de chaves*, que funciona da seguinte forma: a chave de *bootstrapping* cifra $\mathbf{s}$ sob z. Então, de certa forma, fechamos esse círculo criando uma chave ksk que cifra $\mathbf{z}$ sob $\mathbf{s}$. Obviamente, a geração de ksk deve ser feita de forma privada, no entanto, a chave ksk é pública. Então, usamos ksk para "substituir" a chave de $\mathbf{c}'$ por $\mathbf{s}$.

Para reduzir o crescimento do ruído durante a substituição de chaves, usamos algo parecido com a decomposição usada no GSW, então, ksk cifra a chave $\mathbf{z}$ multiplicada por potências de dois. Logo, para $0 \leq i < N$ e $0 \leq j < \lceil \log_2(q) \rceil$, definimos

$$\mathsf{ksk}_{i,j} := (\mathbf{a}_{i,j}, b_{i,j} := \mathbf{a}_{i,j} \cdot \mathbf{s} + e_{i,j} + 2^j \cdot z_i) \in \mathbb{Z}_q^{n+1}.$$

Note que cada $\mathsf{ksk}_{i,j}$ cifra $2^j \cdot z_i$ com o problema LWE usando a dimensão n do esquema de base. Finalmente, definimos $\mathsf{ksk} := \{\mathsf{ksk}_{i,j} : 0 \leq i < N \text{ e } 0 \leq j < \lceil \log_2(q) \rceil\}$.

O código que implementa a geração da chave de substituição de chaves é apresentado a seguir e também deve ser incluído na definição da classe *Bootstrapper*:

```
def key_swt_gen(self):
  N, Q = self.gsw.n, self.gsw.q
  l = ceil(log(Q, 2))
  z = poly_to_vec(self.gsw.sk, N)

  # K_{i, j} = enc(z_i * 2^j)
  K = [0] * N
  for i in range(N):
    zi = z[i]
    enc_zi = [0] * l
    for j in range(l):
      c = self.base_scheme.enc(0)
      c[1] = (c[1] + 2^j * zi) % Q
      enc_zi[j] = c
    K[i] = enc_zi
  self.ksk = K
```

Dado um criptograma $(\mathbf{a}, b := \mathbf{a}\cdot\mathbf{z} + e + (q/4)m) \in \mathbb{Z}_q^{N+1}$, a ideia principal da substituição de chaves é que ao subtrair $\mathbf{a}\cdot\mathbf{z} + \hat{\mathbf{a}}\cdot\mathbf{s}$ de b, obtém-se um inteiro da forma $\hat{b} = -\hat{\mathbf{a}}\cdot\mathbf{s} + e + (q/4)m$, logo, $(-\hat{\mathbf{a}}, \hat{b}) \in \mathbb{Z}_q^{n+1}$ é um criptograma válido sob a chave $\mathbf{s}$ em vez de $\mathbf{z}$.

Como $\mathbf{a}\cdot\mathbf{z} = \sum_{i=0}^{N-1} a_i \cdot z_i$, usamos ksk para subtrair cada $a_i \cdot z_i$ de b da seguinte forma: Decompomos a_i em base 2 e obtemos $(a_{i,0}, ..., a_{i,L-1}) \in \{0,1\}^L$, onde $L := \lceil \log q \rceil$. Então, note que como $\mathsf{ksk}_{i,j} = (\mathbf{a}_{i,j}, b_{i,j} := \mathbf{a}_{i,j}\cdot\mathbf{s} + e_{i,j} + 2^j \cdot z_i) \in \mathbb{Z}_q^{n+1}$, se multiplicarmos $a_{i,j}$ por $\mathsf{ksk}_{i,j}$, obtemos uma encriptação de $a_{i,j}\cdot 2^j \cdot z_i$ sob a chave $\mathbf{s}$. Então, somando esses criptogramas, obtemos uma encriptação de $\sum_{j=0}^{L-1} a_{i,j}\cdot 2^j \cdot z_i = a_i \cdot z_i$, como desejado. Finalmente, basta subtrair cada encriptação de $a_i \cdot z_i$ de b.

Ou seja, a substituição de chaves consiste em calcular o seguinte criptograma:

$$(\hat{\mathbf{a}}, \hat{b}) := (\mathbf{0}, b) - \sum_{i=0}^{N-1}\sum_{j=0}^{L-1} a_{i,j}\cdot \mathsf{ksk}_{i,j}.$$

Note que se e é o ruído de $(\mathbf{a}, b)$ e se cada $\mathsf{ksk}_{i,j}$ tem ruído próximo de β, então o ruído de $(\hat{\mathbf{a}}, \hat{b})$ é $O(\|e\| + N\cdot L\cdot\beta)$.

O procedimento de substituição de chaves é implementado em Sage com o seguinte método da classe *Bootstrapper*:

```
#   Recebe um criptograma LWE (a, b) que cifra m sob a
# chave z do GSW e devolve uma encriptação de m sob a
# chave s do esquema de base.
def switch_key(self, c):
  a, b = c
  q, n = self.base_scheme.q, self.base_scheme.n
  L = ceil(log(q, 2))
  K = self.ksk
  out_a = vector([0]*n)
  out_b = 0
  for i in range(len(a)):
    v = vector(ZZ(a[i]).digits(base=2, padto=L)) # decompõe ai
    for j in range(L):
      if 1 == v[j]:
        out_a += K[i][j][0] # adiciona "termo a" de ksk
        out_b += K[i][j][1] # adiciona "termo b" de ksk

  # Agora, out_b = out_a * s + e + a * z
  # então, b - out_b = -out_a*s - e + e_b + (q / 4) * m
  out_b = (b - out_b) % q
  out_a = (-out_a ) % q
  return [out_a, out_b]
```

1.8.2. *Bootstrapping*: o procedimento completo

Para implementar o *bootstrapping*, agora basta agrupar como na figura 1.4 os três procedimentos implementados nas seções anteriores. Isso é feito com a seguinte função, que deve também ser incluída na definição da classe *Bootstrapper*:

```
def refresh(self, c):
    a, b = self.mod_switch_to_2N(c[0], c[1])
    acc = self.linear_part_dec([a, b]) # enc_z(X^(e_bar + N*m))

    c = self.convert_rlwe_lwe(acc) # enc_z(m) in Z_q^(N+1)

    c = self.switch_key(c) # enc_s(m) in Z_q^(n+1)

    return c
```

Como especificado na definição 1.7.1, para que o criptograma devolvido pelo *bootstrapping* seja válido, é preciso que seu ruído seja menor que $q/16$. Então, agora analisaremos o ruído inserido pelas operações feitas durante o *bootstrapping* e mostraremos como podemos escolher os parâmetros para que o ruído final do criptograma recifrado seja aceitável.

Como ilustrado na figura 1.4, o *bootstrapping* recebe como entrada a chave bk, que consiste em um conjunto de criptogramas com pouquíssimo ruído, então, cada uma das três etapas acumula ruído sobre bk.

Como os criptogramas de bk têm seus ruídos gerados com uma distribuição gaussiana discreta cujo desvio-padrão é σ, podemos dizer que o ruído de bk tem magnitude $O(\sigma)$. Assim, o ruído aumenta durante o *bootstrapping* da seguinte forma:

- A primeira etapa consiste de n produtos externos envolvendo criptogramas gerados pelo cmux. Lembre-se que o cmux no máximo multiplica o ruído por 3, então o ruído de cada criptograma envolvido nos produtos externos ainda é $O(\sigma)$. Logo, conforme mostrado no corolário 1.6, que também é válido para produtos externos, o ruído final dessa etapa é $O(n\sigma N\log(q))$.
- Na segunda etapa, o ruído é multiplicado pelo vetor $\mathbf{u} = (-1,...,-1) \in \mathbb{Z}^N$, então, sua magnitude pode ser limitada por $\sum_{i=1}^{N} O(n\sigma N\log(q)) = O(n\sigma N^2\log(q))$.
- Como discutido na seção 1.8.1, a terceira e última etapa adiciona ao ruído um termo igual a $O(N\beta\log q)$, onde β é um limitante superior para o ruído da chave ksk. Em geral, $\beta = O(1)$, portanto, o ruído final do criptograma recifrado é

$$O(n\sigma N^2\log(q) + N\log q) = O(n\sigma N^2\log(q)).$$

É necessário que o ruído final seja menor que $q/16$, portanto, basta escolher

$$q = \Theta(N^3\log N)$$

se assumirmos $N > n$ e $\sigma = O(1)$.

Essa análise do crescimento do ruído pode ser vista como uma análise de pior caso, pois ela foi derivada limitando somas do tipo $e_1 + ... + e_k$ por $k \cdot \max(e_1, ..., e_k)$. Mas foi observado em [ASP14, DM15] que, como os ruídos seguem uma distribuição gaussiana, o valor esperado para somas dessa forma é na verdade $\sqrt{k} \cdot \max(e_1, ..., e_k)$. Com isso, é possível fazer uma análise de caso médio da seguinte forma:

- A magnitude esperada para o ruído obtido na primeira etapa é $O(\sigma \cdot \sqrt{nN\log(q)})$.
- Na segunda etapa, espera-se que a magnitude do ruído seja $\sum_{i=1}^{N} O(\sigma \cdot \sqrt{nN\log(q)}) = O(\sigma N \cdot \sqrt{n\log(q)})$.
- A terceira etapa então adiciona o termo $O(\beta \cdot \sqrt{N\log q})$, onde $\beta = O(1)$ é um limitante para o ruído de ksk. Então, com grande probabilidade, o ruído no final do *bootstrapping* é limitado por

$$O(\sigma N \cdot \sqrt{n\log(q)} + \sqrt{N\log q}) = O(\sigma N \cdot \sqrt{n\log(q)}).$$

Assim, novamente tomando $N > n$ e $\sigma = O(1)$, reduz-se q para $\Theta(N^{1.5}\log N)$.

Para testar o *bootstrapping*, o seguinte código em Sage pode ser usado:

```
def test_bootstrapp(k=5):
    n, N, sigma, = 2^3, 2^5, 3.2
    B, l = 2, 17
    q = Q = B^l # assume q = Q
    lwe_scheme = LWEScheme(n, q, sigma)
    gsw = GSW(N, Q, sigma, B)
    print("%s\n%s" % (lwe_scheme, gsw))

    boot = Bootstrapper(gsw, lwe_scheme)

    m = random_bit()
    c = lwe_scheme.enc(m)
    for i in range(1, k+1):
        mi = random_bit()
        m = (1 - m * mi)
        ci = lwe_scheme.enc(mi)
        cnand = lwe_scheme.nand(c, ci)

        c = boot.refresh(cnand)

        dec_m = lwe_scheme.dec(c, level=1)
        assert(m == dec_m) # verifica se c é válido

        noise = lwe_scheme.get_noise(c, m, level=1)
        print("Refreshed noise: %f" % noise)
```

Note que os valores de n, N e q usados nesse código são muito pequenos para garantir que as cifras sejam seguras, mas já são suficientes para garantir a corretude do *bootstrapping*. Para um nível de segurança de 128 bits, parâmetros típicos são $n \approx 600$, $N \approx 1024$ e $q \approx 2^{32}$ [DM15, CGGI16]. Obviamente, a implementação apresentada neste curso tem como finalidade ser clara e fácil de entender, o que faz com que ela seja ineficiente. Assim, usar tais parâmetros com essa implementação faz com que o *bootstrapping* seja lento. Mas as implementações apresentadas em [DM15] e em [CGGI16] usam a linguagem *C++* e bibliotecas altamente optimizadas para multiplicar polinômios, logo, conseguem executar o *bootstrapping* com nível de segurança $\lambda = 128$ em cerca de 0.1 segundo em um computador comercial comum, em uma única *thread*.

Se considerarmos que os cálculos homomórficos serão normalmente executados em servidores "na nuvem", que são muito mais potentes que computadores usais, e também que esses procedimentos são altamente paralelizáveis, essa estratégia de computação homomórfica feita com a avaliação de uma porta lógica seguida de um *bootstrapping* extremamente rápido se mostra muito promissora.

1.9. Conclusão

Neste minicurso foi apresentado código Sage que implementa criptografia completamente homomórfica. Isto permite ao estudante experimentar diferentes parâmetros e desenvolver novas aplicações. O uso da linguagem Sage torna a compreensão mais fácil, enquanto ao mesmo tempo torna possível a prototipagem rápida das funções necessárias. Componentes que possivelmente tenham sido omitidos no texto por falta de espaço podem ser encontrados no repositório [Per21b].

References

[ACC+17] David Archer, Lily Chen, Jung Hee Cheon, Ran Gilad-Bachrach, Roger A. Hallman, Zhicong Huang, Xiaoqian Jiang, Ranjit Kumaresan, Bradley A. Malin, Heidi Sofia, Yongsoo Song, and Shuang Wang. Applications of homomorphic encryption. Technical report, HomomorphicEncryption.org, Redmond WA, USA, July 2017.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3), 2015.

[ASP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *Advances in Cryptology – CRYPTO 2014*, pages 297–314, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BDH+17] Michael Brenner, Wei Dai, Shai Halevi, Kyoohyung Han, Amir Jalali, Miran Kim, Kim Laine, Alex Malozemoff, Pascal Paillier, Yuriy Polyakov, Kurt Rohloff, Erkay Savaş, and Berk Sunar. A Standard API for RLWE-based Homomorphic Encryption. Technical report, HomomorphicEncryption.org, Redmond WA, USA, July 2017.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology - CRYPTO 2018*, volume 10993 of *Lecture Notes in Computer Science*, pages 483–512. Springer, 2018.

[CCD+17] Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Jeffrey Hoffstein, Kristin Lauter, Satya Lokam, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Security of homomorphic encryption. Technical report, HomomorphicEncryption.org, Redmond WA, USA, July 2017.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[CKLY15] Jung Hee Cheon, Jinsu Kim, Moon Sung Lee, and Aaram Yun. Crt-based fully homomorphic encryption over the integers. *Inf. Sci.*, 310(C):149–162, July 2015.

[CN12] Yuanmi Chen and Phong Nguyen. Faster Algorithms for Approximate Common Divisors: Breaking Fully Homomorphic Encryption Challenges over the Integers. In *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*. Springer, 2012.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT 2012*, pages 446–464, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[CS15] Jung Hee Cheon and Damien Stehlé. Fully Homomophic Encryption over the Integers Revisited. In *EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Sofia, Bulgaria, April 2015.

[DM12] R. Dahab and E. M. Morais. Encriptação homomórfica. In A. L. (Org.) Santos, A. (Org.) Santin, C. (Org.) Maziero, and P. A. S. Gonçalves, editors, *Minicursos do XII Simpósio em Segurança da Informação e de Sistemas Computacionais. 12ed.*, pages 1–195, 2012.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg, 2015. Springer.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://eprint.iacr.org/2012/144`.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[GGM16] Steven D. Galbraith, Shishay W. Gebregiyorgis, and Sean Murphy. Algorithms for the approximate common divisor problem. *LMS Journal of Computation and Mathematics*, 19(A), 2016.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[Per21a] Hilder Vitor Lima Pereira. Bootstrapping fully homomorphic encryption over the integers in less than one second. In *Public-Key Cryptography – PKC 2021*, pages 331–359, Cham, 2021. Springer International Publishing.

[Per21b] Hilder Vitor Lima Pereira. SAGE scripts for DGHV and GSW. github, 2021. `https://github.com/hilder-vitor/FHE_for_SBSeg21`.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), September 2009.

[Sag20] Sage Developers. *SageMath, the Sage Mathematics Software System*, 2020. `https://www.sagemath.org`.

Chapter 2

Segurança e Escalabilidade em Sharding Blockchain

Antonio A. de A. Rocha, Célio V. N. de Albuquerque
Eduardo B. Loivos, Bruno T. Gondim
Arthur A. Vianna, André O. Ferreia

Abstract

The circulation of the real money, on paper, is decreasing, migrating to the virtual. As a result, the need for a secure way to carry out transactions without relying on a trusted third-party increase. The cryptocurrencies that emerged in 2009 with the Nakamoto protocol represent an alternative to this demand, however the initial proposal has a scalability problem that makes it impossible to compete with the credit card network. The need to confirm transactions means that a cryptocurrency that uses the traditional blockchain like Ethereum only has 20 to 30 transactions per second, far away from VisaNet which has around 1700 transactions per second. To solve the scalability, the concept of sharding was created. This work proposes to present some sharding models highlighting their particularities and security solutions.
Keywords*: Blockchain, Sharding, scaling, decentralized ledger.*

Resumo

A circulação do dinheiro real, em papel, está cada vez menor, migrando para o virtual. Com isso, a necessidade de uma forma segura de realizar transações, sem depender de uma terceira parte confiável, aumenta. As criptomoedas, que surgiram em 2009 com o protocolo do Nakamoto, representam uma alternativa para essa demanda. Porém, a proposta inicial possuí um problema de escalabilidade que impossibilita competir com a rede do cartão de credito. A necessidade de confirmar as transações fazem com que uma criptomoeda, que usa a blockchain tradicional como Ethereum, permita de 20 a 30 transações por segundo. Essa taxa é bem distante da VisaNet que possui por volta de 1700 transações por segundo. Para superar a limitação de escalabilidade, foi criado o conceito de sharding. Este trabalho propõe apresentar alguns modelos de sharding destacando suas particularidades e soluções de segurança.
Palavras-chaves*: Blockchain, Sharding, escalabilidade, razão descentralizada.*

2.1. Introdução

Com o surgimento da Internet, um dos pontos que apresentaram deficiência é justamente a questão relacionada à implementação de segurança. Problema este que se agravou com a quantidade de transações que emergiram do uso da rede mundial de computadores, pois aos poucos ela foi se tornando uma grande ferramenta para comprar e fazer transações. Mas, antes de adentrarmos nesse assunto, vamos fazer uma breve análise do dinheiro que conhecemos e a real motivação da Tecnologia Blockchain para os dias atuais.

A história da civilização nos conta que o homem primitivo procurava defender-se do frio e da fome, abrigando-se em cavernas e alimentando-se de frutos silvestres, ou do que conseguia obter da caça e da pesca. Ao longo dos séculos, com o desenvolvimento da inteligência, passou a espécie humana a sentir a necessidade de maior conforto e a reparar no seu semelhante. Assim, como decorrência das necessidades individuais, surgiram as trocas, que foram fundamentais para a mudança do comportamento Humano.

Esse primeiro sistema de troca direta, que durou por vários séculos, deu origem ao surgimento de vocábulos como "salário", o pagamento feito através de certa quantidade de sal, que a partir das grandes navegações, possibilitaram uma expansão territorial de trocas. Daí surgiram as primeiras moedas, tal como conhecemos hoje, feitas geralmente em metal. Com isso, surgiram então a necessidade de guardar as moedas em segurança dando surgimento aos bancos. Assim os negociantes de ouro e prata, por terem cofres e guardas a seu serviço, passaram a aceitar a responsabilidade de cuidar do dinheiro de seus clientes e a dar recibos escritos das quantias guardadas. Esses recibos (então conhecidos como "goldsmith´s notes") passaram, com o tempo, a servir como acordos através de pagamento por seus possuidores, por serem mais seguros de portar do que o dinheiro vivo. Também surgiram as primeiras cédulas de "papel-moeda", ou cédulas de banco, ao mesmo tempo em que a guarda dos valores em espécie dava origem a instituições bancárias [CasaDaMoeda 2015].

A ideia de livro-razão surgiu algum tempo depois, em 1494, desenvolvida por um padre chamado Luca Pacioli. Na sua época, esse livro consistia em promover um balanço de ativos tangíveis e intangíveis, disponibilizando de forma ordenada e detalhada várias operações de crédito e débito e a composição de um balanço final. A tecnologia Blockchain é derivada dessa ideia, sendo por vezes chamada de livro-razão digital. No livro, Summa de Arithmetica, Geometria, Proportioni et Proportionalità, escrito em 1494, o frei franciscano Luca Pacioli (1445-1517) desenvolveu os primeiros estudos de matemática para serem utilizados em contabilidade. Nesses estudos, o religioso utiliza entre outras coisas, a observação da movimentação de feiras livres com o objetivo de compreender o "Método das Partidas Dobradas", que é o sistema padrão universal de débito e crédito utilizado até hoje pelas empresas, governos e mercados mundiais e, não obstante, é estudado ainda hoje como matéria básica nos cursos de Administração e Negócios.

Seguindo a escala de modernização da Economia e da digitalização da sociedade (WEB), com o surgimento da Internet, um dos pontos que apresentaram deficiência foi justamente a questão relacionada à implementação de segurança. Problema este que se agravou com a quantidade de transações que emergiram do uso da Internet, pois aos poucos ela foi se tornando uma grande ferramenta para comprar e fazer transações. Na tentativa de mitigar tal questão, em 1971 foi criado por Horst Feisel (IBM) um algoritmo

de criptografia, denominado Lucifer, baseado em um elevado nível de segurança e uma chave de codificar e descodificar. Já em 1974 um grupo de cientistas da IBM adequou melhor a ferramenta Lucifer e surge o Data Encryption Standard "DES" (um sistema de codificação simétrico por blocos de 64 bits, dos quais 8 bits ou um byte servem de teste de paridade para verificar a integridade da chave). A principal motivação do grupo foi justamente readequar a ferramenta criada e promover maiores índices de segurança para ela. Em 1981, o DES foi adotado com o nome Data Encryption Algorithm (DEA) pela American Standard Institution com o principal objetivo de promover padronização de cifragem e procedimentos para serem utilizados em instituições financeiras. Desta forma, o DES se tornou o principal algoritmo de chave única, mas, de qualquer forma, a criação do DES não foi um sucesso absoluto para evitar fraudes e vazamentos de informações.

Ainda na constante busca de uma solução única e eficaz para promover a segurança das transações eletrônicas, em meados de 1983, David Chaum, grande cientista e entusiasta em criptografia, fundou a DigiCash, uma empresa de moeda eletrônica criptografada. Seu produto foi chamado de E-cash e seu objetivo era que os usuários obtivessem moedas digitais de um banco e não pudessem ser rastreados nem pelo banco nem por qualquer outra pessoa. Tal invenção foi precursora do movimento Cypherpunk, que teve seu início no final da década de 1980, já propondo sistemas *peer-to-peer*. Naquela mesma ocasião, um dos sócios da empresa DigiCash, Nick Szabo, formado em Direito e criptógrafo, promoveu a pesquisa relativa a contratos digitais e moeda digital, denominada "Contratos Digitais". Em 2005, Szabo desenvolveu um mecanismo inovador para tratamento de uma moeda digital que recebeu o nome de Bit Gold, algumas bibliografias se referem a ela como a precursora do Bitcoin.

Finalmente, em 2008, foi lançada a moeda virtual Bitcoin, que se utiliza da plataforma Blockchain para suas transações. O lançamento da moeda foi feito por Nakamoto Satoshi, um pseudônimo, pois até os dias de hoje não se sabe precisamente quem foi o criador da moeda virtual Bitcoin. Não é sabido nem mesmo se trata-se de uma pessoa somente ou uma equipe de cientistas que desenvolveu a tecnologia [Fernando Wosniak Steler 2017].

2.1.1. Então, o que é Blockchain?

A expressão "*In Blockchain We Trust*" vem sendo utilizada para ressaltar a confiança na segurança dessa solução tecnológica. Isto porque, nela as transações entre indivíduos e a transferência de valores é garantida por meio de algorítimos matemáticos e criptográficos. A Blockchain é uma tecnologia de núcleo que possibilita que grandes grupos de pessoas cheguem a um acordo (também definido como, consenso) e registrem transações permanentes, sem uma autoridade central.

O termo (Bloco = block + chain = cadeia) tem origem no seu funcionamento, onde cada bloco validado é criptograficamente selado ao bloco anterior, formando uma cadeia de blocos cada vez maior, diretamente relacionado na construção de uma economia digital justa, inclusiva, segura e democrática. Por isso, a Blockchain é descrita como uma máquina de confiança, não é apenas um *ledger* distribuído em larga escala, mas também como uma trilha de auditoria imutável, onde cada bloco é incorporado em todos os seguintes, impossibilitando a alteração da história de seu conteúdo.

A tecnologia Blockchain também é, portanto, um banco de dados distribuídos,

sendo praticamente invulnerável a falhas e adulterações, e suas múltiplas utilidades descolam-se da tecnologia de Criptomoedas Bitcoin – para a qual foi criada. Antes do desenvolvimento da tecnologia Blockchain, os registros contábeis eram mantidos em bancos de dados centralizados e não públicos. As pessoas precisavam confiar na idoneidade do banco de dados para ter certeza de que não haveria nenhuma alteração nos registros (saldos e transações da conta). Com a Blockchain, os dados são distribuídos entre todos os participantes, com total transparência e descentralização. Logo, torna-se desnecessário confiar em uma terceira parte para que os dados contábeis sejam registrados corretamente e não haja perigo de fraudes. Podemos concluir que o modelo de solução Blockchain assemelha-se a um "livro-razão", ou seja, uma base de informações com entrada de diversos dados e transações. Todas as informações imputadas e contidas na ferramenta são compartilhadas entre vários usuários.

O processamento desta base de dados é feito em blocos, "de tempos em tempos", criando um código de verificação a cada bloco processado. Estes códigos de verificação são criados com base nos blocos processados anteriormente, fazendo com que a Blockchain seja uma solução de alta confiabilidade, pois, uma vez adulterado um bloco, isso impactará nos demais blocos processados.

Idealmente utiliza-se Blockchain em soluções que temos desconfiança mútua pelas partes envolvidas, são exemplos delas: Estabelecimento de contratos; Registro de propriedade intelectual; Estabelecimento de Identidade digital; Prontuário médico; Cartórios digitais; Operações cambiais imediatas; Sistema de voto digital; e, Auditabilidade [REVOREDO 2019].

2.2. Blockchain Tradicional

A *Blockchain* oferece propriedades que proveem benefícios às aplicações e sistemas baseados nesta tecnologia. Conforme definido em [Rebello et al. 2019], as principais propriedades da *Blockchain* são:

- **Descentralização.** A *Blockchain* é executada de maneira distribuída, sem a necessidade de um intermediário confiável para a troca de ativos, através do estabelecimento de consenso entre todos os participantes da rede;
- **Imutabilidade.** Os dados armazenados em uma corrente de blocos são imutáveis. Não é possível modificar ou recriar qualquer dado incluído na corrente de blocos. Toda atualização na corrente de blocos é realizada de forma incremental;
- **Irrefutabilidade.** Os dados são armazenados na corrente de blocos em forma de transações assinadas, que não podem ser alteradas devido à propriedade de imutabilidade da corrente de blocos. Portanto, o emissor de uma transação jamais pode negar sua existência;
- **Transparência.** Todos os dados armazenados na *Blockchain* são acessíveis por todos os participante da rede. Permitindo que todos os participantes possam verificar, auditar e rastrear os dados inseridos na corrente de blocos para encontrar possíveis erros ou comportamentos maliciosos;
- **Disponibilidade.** As correntes de blocos são estruturas replicadas em cada participante da rede e, portanto, a disponibilidade do sistema é garantida mesmo sob

falhas, devido à redundância de informações;

- **Anonimidade.** Os usuários e nós mineradores de uma *Blockchain* são identificados por chaves públicas ou identificadores únicos que preservam suas identidades. Ainda, é possível utilizar uma chave pública em cada transação, evitando a rastreabilidade do usuário e conferindo um grau a mais de anonimidade.

2.2.1. Função Hash

A função hash é uma função matemática que tem como entrada uma string de tamanho variável e a mapeia em uma string de tamanho fixo. A conversão tem como objetivo gerar uma identidade única, resistente a colisões. Uma função hash é resistente à colisão, se houver baixa probabilidade de encontrar dois valores de entrada distintos, que possuam um mesmo valor de saída (Figura 2.1). Dado que mapear uma grande quantidade de dados para um universo menor, quanto menor a probabilidade de encontrar hash iguais de forma proposital, melhor a função de hash.

Figure 2.1. Exemplo de colisão entre dois hashs

Uma característica importante do hash é que, uma pequena variação na entrada resulta em uma grande variação na saída. Como pode ser visto na Figura 2.2, mudar uma uma letra de maiúscula para minúscula gera uma saída completamente diferente. Não deve ser possível adicionar conteúdo “a” um texto sem modificar sua saída. Se uma função H(a) resulta a string fixa “y”, encontrar uma string “b” tal que concatenada com “a” resulte em H(a || b) = y, em um curto espaço de tempo, não deve ser possível. Da mesma forma, se uma entrada tem uma parte gerada de forma aleatória, do hash resultante não pode ser escolhido um hash específico. Outra propriedade da função hash é: Dada uma saída, não é possível determinar a entrada que a originou. Em um espectro pequeno ou repetido de entradas, uma vez vista a solução de para uma dessas entradas, passa a ser possível identificá-la através de seu hash. É possível ocultar a entrada usando um nonce, ou um valor secreto, concatenado a entrada original.

Figure 2.2. Exemplo hash usando SHA256

2.2.2. Estrutura de bloco

A estrutura de bloco consiste principalmente em um conjunto de dados. No universo das criptomoedas, os dados de um bloco representam as transações que ocorrem em um

determinado período de tempo. Além dos dados, um bloco possui seu índice ou altura, e um campo de nonce. Uma vez formado um bloco, uma função de hash é usada para gerar uma assinatura para o mesmo, garantindo que os dados registrados nele não serão alterados. Uma função de hash comum para esta finalidade é a SHA de 256 bits usada pelo Bitcoin.

Ao criar um bloco e assiná-lo, os dados são verificados e apenas as transações válidas entram em um bloco. Para dificultar que um usuário mal-intencionado valide um bloco com dados forjados é necessário um esforço para assinar o bloco. As formas mais comuns de esforço usadas são *Proof of Work (PoW)* e *proof of stake (PoS)*, mais detalhadas à frente.

2.2.3. Consenso

Plataformas Blockchain usam consenso descentralizado para manter a consistência em uma máquina de estado distribuída. Esta pode ser usada para realizar pagamentos completamente descentralizados ou mesmo cálculos de Turing completos, tornando realidade a criação de aplicações descentralizadas. É papel do consenso em sistemas de Blockchain garantir que todos os nós confiáveis em uma rede Blockchain executem as mesmas atualizações de estado na mesma ordem [Muratov et al. 2018].

Há dois tipos bem comuns de algoritmo de consenso, os baseados em prova e os em voto. No conceito básico de algoritmo de consenso baseado em prova, entre os muitos nós que se juntam à rede, o nó que executa a prova terá o direito de acrescentar um novo bloco à corrente e receber a recompensa. Os principais algoritmos com base em prova são *Proof of Work* , *Proof of Stake* e alguma variação desses [Pahlajani et al. 2019]. Já os algoritmos de consenso baseado em votação, além de manter o livro-razão, todos os nós da rede teriam que verificar juntos as transações ou blocos. Eles se comunicam uns com os outros, antes de decidir anexar ou não, os blocos propostos à cadeia. Dentro de quase todas essas variantes, os nós necessitam que um número mínimo de nós aceitem o mesmo bloco proposto para anexá-lo à cadeia [Pahlajani et al. 2019].

Proof of Work (PoW)

O uso de um sistema como PoW impede que usuário utilize seu poder computacional para gerar um spam de determinada informação. No cenário da Blockchain, o PoW impede que diversos blocos sejam criados sequencialmente pelo mesmo usuário. O PoW envolve a busca de um valor de nonce que quando processado por um hash, como com SHA-256, o *output* começa com um determinado número de bits zero. O trabalho médio necessário é exponencial, de acordo com número de bits zero necessários, e pode ser verificado executando um único hash. [Nakamoto 2008]

Proof of Stake (PoS)

Normalmente pergunta-se, se nós devem manter o consumo de energia para ter uma criptomoeda descentralizada. Portanto, demonstrar que a segurança de criptomoedas ponto a ponto não precisa depender de consumo de energia, é um marco importante tanto teórico

quanto tecnologicamente [King and Nadal 2012]. O PoS aproveita o fato de que a posse de moeda é proporcional ao tempo que um nó participa da rede. Desta forma demonstra a sua confiabilidade e assegura a honestidade do mesmo.

Proof of Elapsed Time (PoET)

O consenso PoET atua como uma loteria, na qual, cada nó deve esperar um período de tempo randômico, e o primeiro a concluir este tempo de espera ganha o bloco. O próprio nó deve gerar um valor de tempo pelo qual ele irá ficar em *sleep mode*, o primeiro nó a despertar, ou seja, aquele que tiver o menor período de tempo de espera, irá fazer o commit do bloco para a Blockchain e realizar o brodcast das informações relevantes [Jake Frankenfield 2020]. Geralmente, alguma prova de que ele não adulterou o processo. Para garantir que o nó não gere valores de tempo pequenos para sempre ganhar o bloco, os nós que participam deste consenso precisam executar esta geração de valores em um *TEE(Trusted Execution Environment)*. TEEs são ambientes de execução que têm uma superfície de ataque extremamente pequena e são equipados com procedimentos de verificação criptográfica que podem fornecer atestados verificáveis externamente e evidências de adulteração [Corso 2019].

Como o PoET foi desenvolvido pela Intel em 2016, ele foi pensado para ser usado juntamente com a tecnologia TEE da Intel, o SGX [IntelSGX 2021], tecnologia essa que permite ter áreas seguras no processador, para proteger código e dados selecionados contra modificações. Desta forma, o tempo de espera será gerado por um código que esteja rodando numa área segura do processador. Depois de esperar o tempo necessário, o nó receberá um certificado confirmando que ele esperou o tempo que havia sido sortead. Este certificado pode ser verificado, pois TEEs fornecem atestados verificáveis. Sendo, então, possível verificar que o nó realmente esperou o tempo proposto. Por fim, como mencionado anteriormente, o nó que ganhar o bloco fará o broadcast juntamente com informações relevantes, neste caso, o certificado [Centieiro 2021]. Portanto, este consenso destaca-se pela justiça, uma vez que funciona como uma loteria, e pelo baixo consumo de recursos e energia, uma vez que não é feito nenhuma espécie de processamento pesado e os nós ficam em *sleep mode* enquanto esperam.

Consenso baseado em voto

Nesse tipo de consenso, todos os nós mineradores devem votar pela aprovação ou rejeição de um bloco, e atingir um consenso antes de adicionar o novo bloco à corrente. Como consequência, todos os nós mineradores devem ser conhecidos e identificados. O protocolo de consenso garante que a adição de um bloco à corrente ocorre de forma sincronizada para todas as réplicas .Isto é, as réplicas adicionam a mesma sequência de blocos na corrente.

Dentre os modelos baseados em voto, os tolerantes a falhas bizantinas (Byzantine Fault Tolerant - BFT) são particularmente interessantes. A ideia principal dos protocolos BFT é eleger um líder por uma rodada, que determina e propõe um novo bloco aos participantes do consenso. Os protocolos BFT promovem uma camada de confiança a mais

em comparação a protocolos tolerantes apenas a falhas por parada, pois toleram comportamento malicioso na rede. No entanto, geralmente necessitam de mais réplicas, quando comparado com os protocolos que toleram falhas apenas por parada, para tolerar a mesma quantidade de falhas [Rebello et al. 2019].

2.2.4. Cadeia de blocos

A função hash garante que qualquer alteração em um bloco seja facilmente detectada. Para garantir a integridade dos dados é necessário que um bloco esteja ligado ao próximo através do hash. Assim, para alterar um bloco deve-se alterar todos os blocos posteriores. Os blocos são conectados através de um campo adicional contendo o hash do bloco anterior. O primeiro bloco, conhecido como bloco gênesis, possui esse campo com o valor 0 em todos os caracteres.

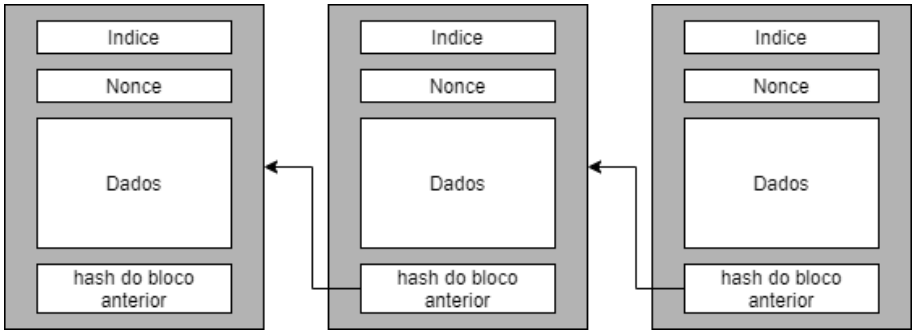


Figure 2.3. Modelo *Blockchain*

Todos os usuários possuem uma cópia da *Blockchain*, e sempre que um minerador confirma um bloco ele transmite sua solução na rede. Devido a latência ou a ação de um nó mal intencionado, duas soluções podem ser aceitas em partes diferentes da rede gerando um *fork* na *Blockchain*. Uma vez identificado o fork os nós consideram a cadeia mais longa como a verdadeira. Logo para falsificar os dados de um bloco da cadeia, um usuário malicioso precisa, no caso de prova de trabalho, de poder computacional suficiente para tornar seu fork mais longo que a *Blockchain* verdadeira.

2.2.5. Bitcoin x Ethereum

Como o protocolo Bitcoin é "Open Source", qualquer um poderia pegar seu protocolo, bifurcar (modificar o código) e iniciar sua própria versão de dinheiro eletrônico. Essa qualidade da Blockchain Bitcoin possuir um código aberto que contribuiu para que, ao longo dos anos, o protocolo Bitcoin fosse modificado centenas de vezes para criar versões alternativas do Bitcoin que são mais rápidas ou mais anônimas. Percebeu-se que o protocolo Blockchain subjacente possibilitava que pessoas desconhecidas, ou que não confiavam entre si, realizassem qualquer tipo de transação de valor, e não apenas dinheiro, sem quaisquer intermediários.

Começam a surgir, então, projetos que buscavam usar a tecnologia Blockchain para transferências, sem intermediários, de outros tipos de valor. Também ganhou corpo a ideia de se afastar da blockchains de propósito único, para criar um protocolo onde qualquer tipo de transação, sem validadores tradicionais de confiança, fosse possível. En-

tão, ao perceber que as adaptações da Blockchain Bitcoin não eram satisfatoriamente eficientes nem flexíveis, Vitalik Buterin introduziu a ideia de dissociar as funcionalidades blockchain e iniciou o projeto da Blockchain Ethereum [Wood et al. 2014] em 2014.

Ao contrário da Blockchain Bitcoin, que é uma Blockchain de propósito único com um único contrato inteligente, a Blockchain Ethereum é projetada como uma rede de computadores descentralizada na qual qualquer tipo de contrato inteligente pode ser programado, permitindo qualquer tipo de troca direta de valor. Outras Blockchains surgiram como solução, pois o surgimento da Ethereum inspirou projetos de Blockchain mais recente (como NEO, EOS, CARDANO, CHAINLINK, QTUM, STELLAR, GOCHAIN, entre outros. Além do aspecto tecnológico, fatores técnicos, econômicos e legais também serão relevantes para avaliar a viabilidade de uma Blockchain [REVOREDO 2019].

2.2.6. Casos de uso e aplicações da Blockchain

De acordo com o site (http://medium.com) entre 2020 e 2025, nosso mundo será povoado por uma nova espécie de 50 bilhões de dispositivos conectados, 100 milhões de robôs e 20 milhões de veículos elétricos. Isso também é confirmado pelo fórum econômico mundial, onde eles assumem que 30% da Contabilidade, Aquisições e Liquidação serão feitas por blockchain e AI. 10% do PIB será baseado em tecnologias de blockchain até 2025 e a sociedade estará cercada por uma nova espécie de máquinas autônomas, carros autônomos, IAs para tomada de decisão e robôs de edição de DNA CRISPR.

Acredita-se que as máquinas serão os clientes do futuro presumindo que mais cedo ou mais tarde, terão carteiras integradas, as primeiras tentativas nesse sentido já foram feitas e com o Ethereum, é possível implantar carteiras com contratos inteligentes mas embora a aplicação como moeda e sistema financeiro sem intermediários seja a aplicação mais famosa da tecnologia de blockchain, várias outras possibilidades e aplicações podem ser vislumbradas., ainda que tenha algumas desvantagens tecnológicas, a tecnologia traz várias possibilidades. [Eichmann 2018]

2.2.6.1. O que é DAO?

Uma Organização Autônoma Descentralizada, ou DAO, é uma organização ou empresa teórica operada por código em vez de pessoas. Os DAOs criam uma maneira de as organizações ou empresas serem estruturadas de forma menos hierárquica, argumentam os defensores, com os investidores direcionando diretamente a direção das empresas, em oposição aos líderes designados.

Os defensores do DAO acreditam que o Ethereum pode dar vida a essa ideia futurista. Ethereum é a segunda maior criptomoeda por capitalização de mercado e é a maior plataforma para usar a tecnologia por trás da criptomoeda - blockchain - para usos além do dinheiro. A ideia é que, se o bitcoin pode eliminar os intermediários nos pagamentos online, a mesma tecnologia ou uma tecnologia comparável pode fazer o mesmo pelos intermediários nas empresas? E se organizações inteiras pudessem existir sem um líder central ou CEO comandando o show?

Com isso erá possível criar múltiplas cópias de segurança: Redundância de infor-

mações é algo fundamental para evitar perdas de dados e nada tem mais backups do que algo controlado por um blockchain, afinal, cada um dos mineradores e até mesmos outros membros da rede (como o caso dos fullnodes do bitcoin) possuem uma cópia completa, atualizada e integral de todos os blocos e, portanto, todas as informações registradas por ele como:

- Registros virtualmente inalteráveis: o hash que valida um bloco é formado pelas informações registradas em um bloco e hash do bloco anterior, que foi gerado pelos dados dele e seu antecessor, e por aí vai. Assim sendo, as informações em um blockchain são incrementais e acumulativas, armazenadas de forma a não ser alteradas ou apagadas.

- A autenticidade de documentos a garantir a autenticidade de documentos é algo custoso, especialmente no Brasil, e ser caro não garante que o processo é à prova de falhas ou fraudes, pois sempre que houver um fator humano que possa ser corrompido e subverter o sistema, a fraude será possível. A startup brasileira OriginalMy surgiu com uma proposta: registrar documentos em um Blockchain, beneficiando-se de, pelo menos, duas características importantes: o fato de que o registro do documento não possa ser alterado ou removido e a transparência que um blockchain público provê, permitindo a verificação deste registro sem burocracias.

Também será possível uma maior Proteção dos direitos autorais, caso um sistema notarial de algum país fosse implementado integralmente em um único blockchain, além de mitigar fraudes por adulteração ou remoção de informações, todos os documentos poderiam ser verificados e confrontados, evitando a fraude do "vidente que registra sua previsão em cartório": algumas pessoas fazem dezenas de previsões aleatórias como "prever" celebridades mortas em acidentes aéreos, por exemplo. Ao registrar tais previsões em dezenas de cartórios diferentes (cada previsão em um cartório diferente), quando o previsto se torna um fato, basta apresentar a previsão que "acertou" o acontecimento, desprezando todas as demais.

Algo semelhante aconteceu na Copa do Mundo de 2014, quando uma conta de Twitter chamada "@fraudefifa" quis provar que a FIFA manipulava os resultados da competição. Curiosa, no entanto, foi a maneira pela qual a conta decidiu "provar" sua tese, realizando postagens com todas as possibilidades de confrontos e vencedores, dias antes das partidas. Quando dois times realmente jogavam e um destes ganhava, bastava remover do Twitter todas as postagens desfavoráveis.

Surgiram outras iniciativas de autenticação de documentos na Internet, como é o caso do site LexisNexis, que também registra os documentos de maneira a comprovar sua autoria futura. Nesse contexto, a gigante Kodak decidiu não investir na tecnologia em razão da grande receita que a empresa adquiria na revelação de filmes fotográficos tradicionais.

A ideia é criar uma plataforma de gerenciamento de direitos de imagem da qual os fotógrafos possam registrar seus trabalhos em um blockchain, que impedirá que tal registro seja alterado ou apagado. Desta maneira, o registro da foto em um blockchain

poderia comprovar um eventual plágio fotográfico, permitindo identificar o autor da obra com facilidade.

Em 2018, pela primeira vez, na França, o Carrefour usou a tecnologia blockchain com uma de suas linhas icônicas de produtos animais: o frango Carrefour Quality Line Auvergne, do qual um milhão é vendido todos os anos , um marco importante para seu plano de transformação chamado Carrefour 2022. A tecnologia blockchain já é utilizada para frangos de linha livre do Carrefour Quality Line Auvergne e será utilizado em mais oito linhas de produtos de origem animal e vegetal, como ovos, queijo, leite, laranja, tomate, salmão e bife moído. Um sistema inovador projetado garante aos consumidores uma completa rastreabilidade do produto.

Existem estudos de Implementação do Smart Governance, votação 2.0, pois além de garantir a privacidade, o contrato traz transparência ao processo de apuração pois, ao verificar as condições para as quais os votos foram apurados, é possível garantir que 1 pessoa = 1 voto. Ao utilizar um blockchain como infraestrutura para que o sistema de votação seja realizado, reduzem - se drasticamente quaisquer fraudes que possam hoje ser realizadas em sistemas eleitorais tradicionais.

A criação de contratos inteligentes trouxe consigo uma nova proposta de arquitetura para aplicações, chamada de DAPP. DAPP significa decentralized application, ou aplicações descentralizadas. Em DAPP, utilizada em aplicativos de smartphones para as mais diferentes necessidades. as aplicações possuem parte de sua programação em frontend (instalados nos smartphones, como geralmente o são), mas a programação mais pesada, conhecida como backend, roda em um blockchain em uma rede descentralizada P2P. Desta forma, as DAPPs se tornam verdadeiras aplicações compartilhadas não sendo possível, nem aos próprios desenvolvedores, modificar condições e regras já estabelecidas.

As blockchains e smart contracts são tecnologias em ascensão. Enquanto os ativos digitais são usados por uma parcela muito pequena da população mundial, a adoção da blockchain é inicial em outros campos e ainda faltam exemplos de grandes cases de uso da tecnologia. No entanto, trata-se de algo que sequer completou dez anos de existência. A cada dia que se passa, novas iniciativas são anunciadas e muitas delas prometem revolucionar os mercados nos quais estão inseridas. Não deixe de acompanhar esta tecnologia que será uma das forças mais transformadoras da próxima década. [Eichmann 2018]

2.3. Escalabilidade na Blockchain

Escalabilidade é um termo utilizado em sistemas, que diz respeito à capacidade de um sistema crescer, tendo como intenção atender mais usuários ou adicionar mais funcionalidades. Sendo assim, um sistema é dito escalável quando o seu desempenho aumenta proporcionalmente com o seu poder computacional.

Podemos dividir escalabilidade em duas vertentes, escalabilidade horizontal e escalabilidade vertical. Na escalabilidade horizontal aumentamos o poder computacional do sistema adicionando nós ao sistema, ou seja, adicionando uma nova máquina. Já na escalabilidade vertical aumentamos o poder computacional do sistema melhorando um nó existente, como por exemplo, adicionando mais memória RAM. Porém há sistemas

em que o aumento no poder computacional não acarreta um aumento no desempenho do sistema, nestes casos é necessário rever a arquitetura.

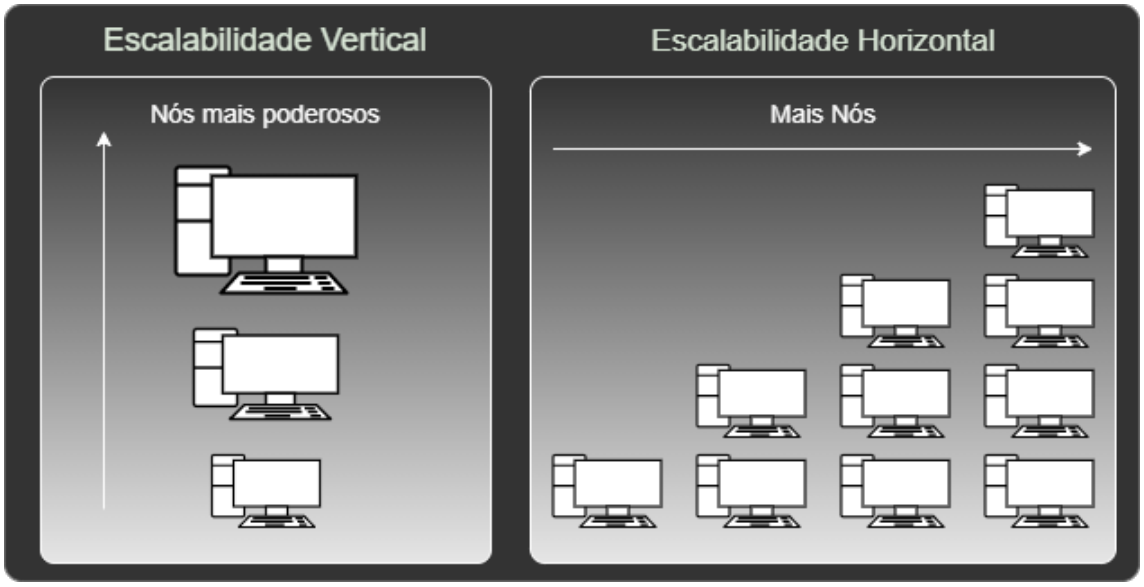


Figure 2.4. Tipos de Escalabilidade

A classe que foi chamada de *Blockchain* Tradicional, apresentada na seção anterior, apresenta problema de escalabilidade, no que diz respeito ao tempo para confirmar uma transação e vazão (quantas transações são confirmadas por segundo) quando comparado a sistemas de cartões de crédito. Que por sua vez são capazes de processar mais de duas mil transações por segundo. Nos sistemas de cartões de crédito, a escalabilidade pode ser tanto vertical, quanto horizontal, pois trata-se de um sistema centralizado. Portanto, a entidade central pode analisar e investir recursos para melhorar o desempenho da forma que julgar conveniente.

Como a *Blockchain* é uma rede P2P, a sua escalabilidade vai se dar principalmente da forma horizontal, no qual novos validadores(mineradores), vão se juntar à rede e o poder computacional do sistema irá aumentar. Se tratando de prova de trabalho, o modelo que é diretamente afetado pelo poder computacional, aumentar a capacidade de um nó não gera ganho para a rede, apenas uma vantagem para este nó em detrimento dos demais. Para os demais modelos essa desigualdade é irrelevante para a escalabilidade, uma vez que um nó possui apenas um voto ou uma maior probabilidade de ser sorteado.

Nas *Blockchains* Tradicionais, apesar de termos esse aumento de poder computacional, nós não temos um aumento proporcional no desempenho do sistema. No caso do Bitcoin, nem mesmo há um aumento, o desempenho da rede se mantém constante, independentemente do poder computacional, na Figura 2.5, temos um comparativo do Bitcoin, que é um sistema que não escala, com uma *Blockchain* ideal, que escala.

2.3.1. Por que a *Blockchain* Tradicional não escala?

A partir da figura 2.5 acima levanta-se um questionamento sobre o porque a *Blockchain* Tradicional não escala. Para responder esta pergunta devemos observar as decisões de

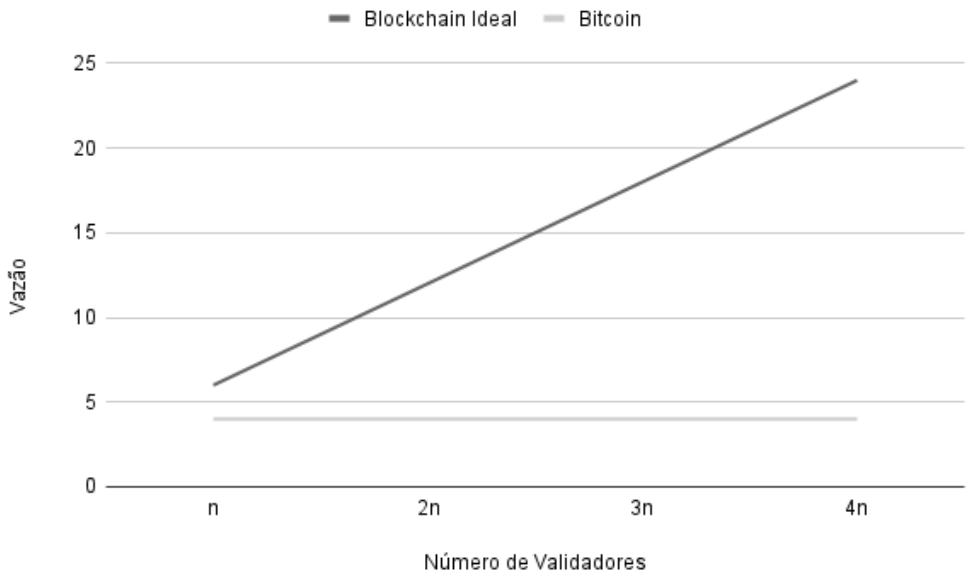


Figure 2.5. Comparativo Bitcoin

arquitetura desta classe de *Blockchain*. Decisões na arquitetura de sistemas são feitas com base em tradeoff. Tratando-se de *Blockchain* e performance, devemos olhar basicamente três pontos principais: a quantidade de blocos que cada nó armazena, o algoritmo de consenso e o tamanho dos blocos da cadeia. Para a análise dos pontos mencionados teremos como caso de estudo a Bitcoin, dado que, outras *Blockchains* que se encaixam na categoria de *Blockchain* Tradicional foram muito influenciadas por ela.

O primeiro ponto diz respeito à quantidade de blocos que cada nó armazena, ou seja, o quanto da cadeia de blocos cada nó guarda. No caso da Bitcoin cada nó armazena toda a cadeia de blocos, isto trará como benefícios, uma maior disponibilidade das informações, uma vez que ela está replicada em todos nós da rede. E desvantagens como uma maior dificuldade de manter a coerência destes dados e um desafio de performance, pois toda vez que for feita uma consulta estaremos consultando a base de dados completa.

Como segundo ponto temos o algoritmo de consenso. Como mencionado anteriormente, o algoritmo de consenso é responsável por manter a consistência dos dados armazenados pelos nós da rede. Essa tarefa se torna consideravelmente mais lenta uma vez que cada nó armazena a cadeia completa. No caso da Bitcoin o algoritmo de consenso utilizado é o Proof of Work, que é executado por máquinas que são chamadas de mineradores.

Ainda sobre o segundo ponto, o próprio algoritmo de consenso Proof of Work é considerado um problema. Neste caso, ele funciona como um puzzle computacional no qual não há atalho. Deve-se simplesmente buscar por um valor que faça parte do conjunto de soluções do puzzle, e esta ideia que faz com que o algoritmo de consenso funcione mas também é o seu maior problema, pois acabamos tendo um desperdício de poder computacional.

Quando um minerador na Bitcoin descobre a solução para uma prova, e ele envia uma mensagem *Broadcast* para avisar aos outros nós da rede que aquela prova foi resolvida e o bloco pode ser fechado. Entretanto existe um tempo de propagação dessa mensagem, e enquanto isso um outro nó pode encontrar uma outra solução para a prova. A partir daí podemos acabar gerando duas cadeias de blocos diferentes, dado que os mineradores podem pegar um conjunto de transações diferentes para tentar formar um bloco, este cenário é chamado de *fork*. É importante ressaltar que, o fork pode ser enxergado como uma anomalia, pois a existência de nós com cadeias de blocos diferentes vai contra a ideia de consenso. Esta situação é resolvida simplesmente adotando a maior cadeia como a correta. Assim temos novamente um desperdício computacional, pois, apenas um *fork* é escolhido para representar a cadeia, desperdiçando todo o poder computacional investido na construção dos outros *forks*.

A respeito do terceiro ponto temos que, na Bitcoin foi definido um tamanho máximo de 1MB para os blocos, este tamanho para o bloco parecia viável quando a Bitcoin foi concebida. Com o passar dos anos a criptomoeda se popularizou muito e por conta disso a quantidade de transações aumentou muito, fazendo com que haja uma demora ainda maior ao confirmar as transações. É necessário que um bloco seja fechado contendo todas as transações, e como os blocos são muito pequenos se comparados ao volume de transações disponíveis, acaba formando-se um grande volume de transações à espera de um bloco para incorporá-las.

Com base nos pontos discutidos, é possível entender porque a Bitcoin não escala. Nesta temos um atraso de confirmação das transações, que se dá justamente por conta do tempo necessário para os mineradores solucionarem o puzzle do Proof of Work. Também por conta do tamanho dos blocos. Esse tempo de confirmação é maior que 10 minutos. Além da demora para a primeira confirmação da transação, temos a situação dos *forks*, que podem fazer com que o *fork* que já tinha processado determinada transação seja descartado em detrimento de um que ainda não a processou.

No fim das contas a Bitcoin só é capaz de processar apenas aproximadamente 4 transações por segundo[L. 2019]. Isso destoa muito se comparado a um sistema de cartão de crédito que atualmente é capaz de processar aproximadamente 1700 transações por segundo[L. 2019], este nível de vazão de sistema é também chamado de padrão Visa. Além de uma vazão muito maior sistemas de cartão de crédito também possuem um tempo de confirmação de transação menor, apresentando então, alta vazão e baixa latência. Tratando-se de criptomoedas e *Blockchain* tem-se como meta um desempenho tão bom quanto o dos sistemas de cartões de crédito.

2.3.2. *Sharding*

À medida que a popularidade da Blockchain cresce, o mesmo acontece com o volume transacional gerenciado pela rede e a carga de trabalho é ajustada para manter a dificuldade. Se pensarmos em uma Blockchain como um banco de dados compartilhado, à medida que mais e mais dados são adicionados à rede, precisa-se encontrar novas maneiras de processar todos esses dados com eficiência e rapidez. É aí que o sharding pode ajudar.[Mearian 2019]

O *sharding* não é uma técnina nova, ele foi originalmente desenvolvido para mel-

horar a performance de bancos de dados muito grandes, e somente, recentemente, passou a ser explorado como solução para escalabilidade na Blockchain. A técnica consiste na fragmentação ou divisão horizontal de bancos de banco de dados, permitindo que processem mais transações por segundo. O *sharding* divide o sistema em partições menores, conhecidas como "*shards*". Cada fragmento é composto por seus próprios dados, tornando-o distinto e independente quando comparado a outros fragmentos, permitindo um melhor manuseio dos mesmos, tornando-os menos pesados e mais fáceis de operar.

O *sharding* pode ajudar a reduzir a latência ou lentidão de uma rede, pois divide uma rede *Blockchain* em fragmentos independentes. No entanto, existem algumas questões de segurança em torno do *sharding* que podem ser exploradas por adversários. Ao subdividir os nós em *shards*, o poder de *hashing* de cada grupo diminuirá consideravelmente. Isso gera um problema de segurança ao permitir que um agente mal-intencionado execute um ataque com mais facilidade. Uma situação que coloca em risco a segurança e a integridade das informações.[bit2me 2020]

Figure 2.6. Tradeoff Sistemas Distribuídos

Devemos ter em mente o tradeoff que estamos fazendo ao implementar uma *Blockchain* que faz uso de *sharding*, pois como ilustrado na Figura 2.6, é impossível atingir os três extremos. O *sharding* por sua vez, tende à escalabilidade e descentralização, introduzindo assim, desafios de segurança à *Blockchain*. Temos então, dois principais desafios introduzidos pelo *sharding* [Wang et al. 2019]:

1. Distribuir os nós de maneira uniforme nos *shards*, de forma a garantir que a maioria deles seja honesta com alta probabilidade;

2. Como garantir que um adversário não obtenha vantagem significativa, enviesando as operações ou criando identidades Sybil, que é quando um único nó consegue burlar a política de identidade e se passar por vários simultaneamente [Douceur 2002].

Outro problema inserido na *Blockchain* tradicional com a implementação de partições ocorre porque os nós, quando atribuídos a um subgrupo, não têm acesso ao saldo dos nós que pertencem aos outros subgrupos. Tornando necessário criar um protocolo para resolver as transações entre os subgrupos.

Portanto, para uma *Blockchain* se manter segura e funcional, enquanto faz *sharding*, ela deve, em geral, ser composta por cinco componentes [Wang et al. 2019], que são:

1. **Identity establishment and committee formation**: Para se juntar ao protocolo, cada nó precisa estabelecer uma identidade, como uma chave pública, um endereço IP e uma solução de prova de trabalho (PoW). Cada nó é então atribuído a um comitê correspondente à sua identidade estabelecida. Nesse processo, o sistema precisa prevenir a identidade Sybil. No entanto, uma blockchain permissionada não requer este processo;

2. **Overlay setup for committees**: Uma vez que os comitês são formados, cada nó se comunica para descobrir as identidades de outros nós em seu comitê. Para uma blockchain, uma sobreposição de um comitê é um subgrafo totalmente conectado contendo todos os membros do comitê. Normalmente, este processo pode ser feito com um protocolo de fofoca(*gossip protocol*);

3. **Intra-committee consensus**: Cada nó dentro de um comitê executa um protocolo de consenso padrão para concordar com um único conjunto de transações. Nesse processo, todos os membros honestos devem concordar com o bloco proposto em seu comitê;

4. **Cross-shard transaction processing**: A transação deve ser confirmada atomicamente em todo o sistema. Para transações cross-shard, os shards relacionados precisam obter consistência. Normalmente, esse processo requer um tipo de transação de "retransmissão" para sincronizar entre os fragmentos relacionados;

5. **Epoch reconfiguration**: Para garantir a segurança dos shards, os shards precisam ser reconfigurados, a cada período de tempo chamado epoch, exigindo uma aleatoriedade. Essa aleatoriedade será usada na próxima epoch.

Com base nestes cinco componentes, iremos analisar diferentes propostas de *sharding*, que serão divididas em três categorias, partial sharding, complete sharding e outras soluções. No modelo de partição parcial, os grupos trabalham em conjunto para gerar uma cadeia única partilhada por todos. Enquanto no completo, cada grupo possui uma cadeia de blocos distinta. Por fim, na categoria outras soluções, iremos analisar propostas que destoam dos padrões vistos nas outras duas. A figura 2.7 ilustra a diferença entre partial sharding e complete sharding.

2.4. Partial Sharding

As primeiras propostas de *sharding* tentam resolver o problema da escalabilidade dividindo o poder computacional da rede, criando grupos capazes de minerar blocos de forma independente. Porém mantêm a estrutura base da *Blockchain* onde todos os nó da rede possuem uma cópia exata da *Blockchain*. Uma vez que cada nó recebe e armazena informações completas do sistema, não existem transações *cross-shard* no sistema, mas os nós ainda sofrem de armazenamento pesado e sobrecarga de largura de banda [Hong et al. 2021]. Uma vez que a fragmentação ocorra em apenas uma das três dimensões processamento,armazenamento e comunicação esse modelo é classificado com partial *sharding*.

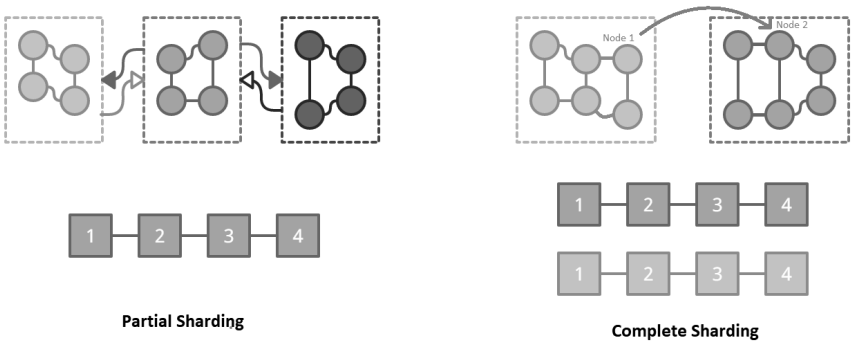


Figure 2.7. Estrutura dos modelos de *sharding*

2.4.1. Elastico

O ELASTICO [Luu et al. 2016] foi uma das primeiras *Blockchains* a fazer uso da técnica de *Sharding*. Em uma *Blockchain*, o sistema depende do poder majoritário para superar os invasores. No entanto, quando o poder de mineração é distribuído para zonas diferentes, um invasor pode reunir seus esforços em direção a uma única zona e pode facilmente exceder o limite de 51% dentro dessa zona[Wang and Wang 2019]. Para evitar que um comitê seja dominado, o ELASTICO utiliza epochs, períodos de tempo no qual os nós são redistribuídos em comitês de forma semi-aleatória de acordo com a identidade gerada, tal estratégia de redistribuição se torna essencial para preservar a segurança da blockchain quando fazemos uso de sharding, por conta disso, veremos variações desta mesma estratégia em todas as propostas.

2.4.1.1. Identity establishment and committee formation

No Elastico existem dois tipos de comitê, final e comum. O comitê final é único e responsável por unificar as soluções de cada comitê, checar as transações entre usuários de grupos diferentes e gerar as possíveis identidades para o próximo epoch. Os membros do comitê final são designados aleatoriamente junto dos demais.

Para estabelecer as identidades, cada nó gera uma **string** identidade usando um grupo de **strings** aleatórias válidas para o epoch. A geração dessas strings será discutida no etapa de epoch reconfiguration. Para gerar sua identidade, um nó realiza uma operação XOR bit a bit das **strings**. Uma vez que cada usuário tenha sua identidade definida, o comitê é gerado a partir dos últimos bits da identidade como mostrado na Figura 2.8. Normalmente o comitê final é composto pelos endereços com 0 nos bits da parte que determina o comitê.

Durante a etapa de formação dos comitês o autor busca boa aleatoriedade. Isso é:

1. Cada usuário tem uma string publicamente aleatória de r bits, gerada de forma verificável no epoch anterior;
2. Nenhum usuário tem acesso a string aleatória verificável mais do que δt antes do início do epoch;

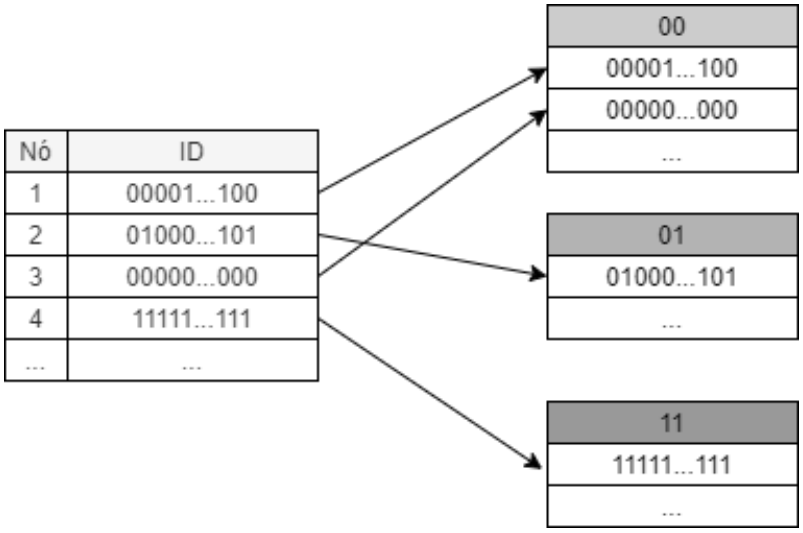


Figure 2.8. Formação de Comitê

3. Usuários mal-intencionados podem influenciar a aleatoriedade com probabilidade insignificante.

Para isso é necessário definir n_0 tal que das n' primeiras identidades criadas, no máximo $n'/3-1$ sejam maliciosas, para todo $n' \geq n_0$. Seja X_i uma variável que assume o valor 1 se a i-ésima identidade for gerada por um processador honesto e $X = \sum_{i=1}^{n'} X_i$. Então X segue a seguinte distribuição binomial:

$$Pr[X \geq 2n'/3] = \sum_{k=0}^{\lceil 2n'/3 \rceil} Pr[X = k] = \sum_{k=0}^{\lceil 2n'/3 \rceil} \binom{n'}{k} f^{n'-k}(1-f)^k \quad (1)$$

Desta forma a probabilidade diminui exponencialmente em n_0. Dado um parâmetro de segurança λ, podemos encontrar n_0 tal que $Pr[X \geq 2n'/3] \leq 2^{-\lambda}.\forall n' \geq n_0$

2.4.1.2. Overlay setup for committees

Uma vez definida a partição de cada usuário da rede, é necessário estabelecer a comunicação entre os membros de um mesmo grupo. É possível perguntar todos os nós da rede por aqueles que pertencem ao comitê, porém seria uma solução não escalável e mesmo que o ELASTICO não busque separar a rede para diminuir a quantidade de mensagens não é interessante aumentar o volume de comunicação da rede. A solução proposta foi que os primeiros nós formam um diretório de identidades, um novo membro envia sua identidade para o diretório e recebe um set de endereços do grupo. Desta forma reduz a complexidade de comunicação de $O(n^3)$ para $O(cn)$ onde c é a quantidade de nós no diretório.

Para diminuir a carga de mensagens na rede foi necessário limitar as visões dos nós da rede, mas isso pode introduzir falhas ao sistema. Essa propriedade busca garantir

que as inconsistências entres as visões sejam limitadas diminuindo o impacto gerado por elas, ou seja:

1. Cada membro tem sua própria opinião sobre quem faz parte do comitê. As visões de dois membros honestos diferem em no máximo 1/3 do tamanho do comitê;
2. Todos os membros honestos têm identidades de outros membros honestos em suas visões;
3. O número total de identidades únicas em todas as visualizações é no máximo $3c/2$ dos quais menos de $1/3$ são maliciosos.

Uma vez que os diretórios honestos aceitam todos os PoWs válidos na última rodada (antes que o comitê seja preenchido por pelo menos c membros), todos os membros honestos do comitê terão outras identidades honestas em sua visão, tornando comportamentos maliciosos a principal fonte de discrepância. Através da propriedade 1 esses comportamentos estão limitados a $1/3$ do tamanho do comitê.

2.4.1.3. Intra-committee consensus

Esta solução para a comunicação faz com que membros de um grupo tenham visões diferentes das identidades do comitê. Os nomes em comum entre as visões tornam o acordo possível. Cada membro tenta entrar em acordo com os membros de sua própria lista através de um algoritmo BFT tratando toda informação externa como falsa. O acordo se propaga entre as listas através dos nós comuns até que só exista uma solução e esta é divulgada para os membros do comitê final. De posse das soluções de cada comitê, o comitê final repete o processo, resolve o bloco e o adiciona na *Blockchain*.

Uma vez que o tamanho máximo das visões é 3c/2 contendo os membros honestos, qualquer protocolo consenso byzantino que aceite comportamentos maliciosos limitados a 1/3 consegue garantir o acordo. Assim, uma vez assinado por 2c+1 nós, ao menos 1 nó honesto executou o algoritmo de consenso e aprovou o valor escolhido.

2.4.1.4. Cross-shard transaction processing

O comitê final tem acesso aos dados de cada comitê. Logo este consegue resolver as transações cross-shard sem a necessidade de envolver os comitês. O comitê final resolve essas transações com o protocolo BFT escolhido.

2.4.1.5. Epoch reconfiguration

Cada membro do comitê final cria uma **string** de tamanho fixo e predeterminado. O comitê final discute e escolhe um set dessas **strings**. Todos os membros transmitem esse set e sua própria **string** junto com o bloco final. Cada nó da rede recebe parte dessa strings devido a latência ou manipulação dos invasores. Cada nó gera sua própria identidade através de um XOR de $2c+1$ dessas strings. O set divulgado e a quantidade de assinaturas

escolhidas garantem que existe pelo menos uma string gerada por um nó honesto entre as escolhas, preservando a aleatoriedade da identidade resultante.

De acordo com a distribuição feita na formação dos comitês, há no mínimo 2c/3 nós honestos no comitê final. Uma vez que cada nó recebe um conjunto de 2c/3 a 3c/2 strings. Como o número máximo de identidades geradas de forma maliciosa é c/2. Ao realizar o XOR em $2c + 1$ strings ao menos uma dessas strings foi gerada por um nó honesto. Como um usuário desonesto não conhece essa identidade antes do comitê final definir o set, a identidade gerada é perfeitamente randômica pois não é possível controlar o resultado para escolher o comitê do próximo epoch.

2.5. Complete Sharding

Sharding completo são aqueles que atingem a fragmentação nas bases computação, armazenamento e comunicação. A divisão do poder computacional permite, assim como no modelo parcial, aumentar o número de blocos minerados ao permitir que as partições minerem simultaneamente. Para o armazenamento o objetivo é diminuir o tamanho da cadeia armazenada em cada nó pois a mesma cresce indefinidamente. O consenso baseado em BFT é muito presente nos modelos que visam a escalabilidade pois as provas demandam tempo e esforço, impedindo que muitos blocos sejam minerados. Esses modelos de consenso requerem um grande volume de comunicação e começam a perder sua eficiência em redes maiores. A fragmentação da comunicação busca resolver esse problema.

Muitas pesquisas foram feitas com o objetivo de atingir o *sharding* completo de forma segura. O ChainSpace, o Omniledger e o RapidChain foram os modelos escolhidos para este estudo. ChainSpace é uma das primeiras pesquisas a atingir o *sharding* completo que usa um sistema a base de cliente que suporta *sharding* de contratos inteligentes genericos. Outro modelo a base de cliente é o Omniledger que usa os clientes para comitar transações entre partições. Já RapidChain propõe um mecanismo de transferência para o output de transações não gastas, baseados em UTXO, a estrutura de moeda que recebeu grande destaque com o crescimento do bitcoin.

2.5.1. Chainspace

Chainspace[Al-Bassam et al. 2017] é uma infraestrutura descentralizada, conhecida como razão distribuída, que suporta contratos inteligentes definidos pelo usuário e executa transações fornecidas pelo usuário em seus objetos. A execução correta de transações de contrato inteligente é verificável por todos.

Chainspace é seguro contra subconjuntos de nós que tentam comprometer sua integridade ou propriedades de disponibilidade através de *Byzantine Fault Tolerânce* (BFT) e técnicas de auditabilidade, e o não repúdio à *Blockchain*. Mesmo quando o BFT falha, mecanismos de auditoria estão disponíveis para rastrear participantes mal-intencionados.

2.5.1.1. Identity establishment and committee formation

A plataforma é agnóstica quanto à linguagem do contrato inteligente, ou infraestrutura de identidade. o ChainSpace suporta recursos de privacidade por meio de técnicas zero-

knowledge modernas como [Bootle et al. 2016] e [Danezis et al. 2014]. [Al-Bassam et al. 2017]

2.5.1.2. Overlay setup for committees

A Figura 2.9 ilustra o design do sistema do Chainspace que é composto por uma rede de nós de infraestrutura que gerenciam objetos válidos e garantem que, apenas as transações válidas nesses objetos sejam confirmadas.

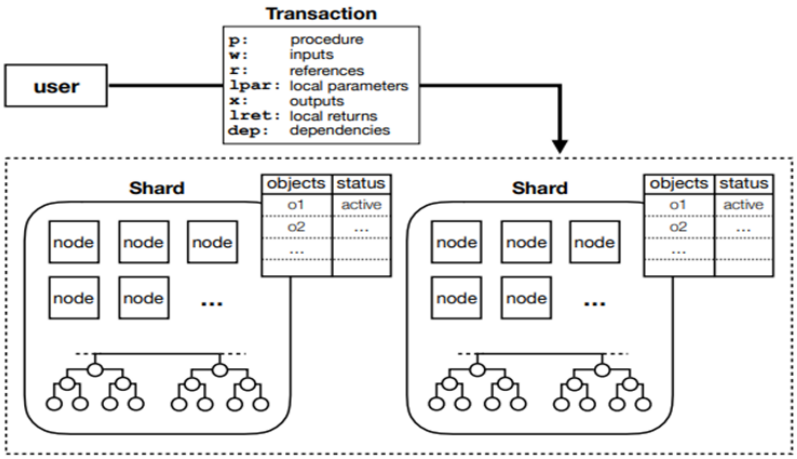


Figure 2.9. Design do Chainspace [Al-Bassam et al. 2017]

2.5.1.3. Intra-committee consensus

Um objeto representa uma unidade de dados no sistema Chainspace, como uma conta bancária, e está em um dos seguintes três estados:

- **ativo:** pode ser usado por uma transação;
- **bloqueado:** está sendo processado por uma transação existente;
- **inativo:** foi usado por uma transação anterior.

Objetos também têm um tipo que determina o identificador exclusivo do contrato inteligente que os define. Os procedimentos de contratos inteligentes podem operar apenas em objetos ativos, enquanto os objetos inativos são retidos apenas para fins de auditoria.

O Chainspace permite a composição de contratos inteligentes de diferentes autores para fornecer recursos do ecossistema. Cada contrato inteligente é associado a um verificador para permitir o processamento privado de transações em nós de infraestrutura, pois os verificadores não usam nenhum parâmetro local secreto. Os verificadores são funções puras, determinísticas e sem efeitos colaterais, que retornam um valor booleano.

Uma transação válida aceita objetos de entradas ativas junto com outras informações auxiliares e gera objetos de saída. Para obter alto rendimento de transação e baixa latência, o Chainspace organiza os nós em fragmentos que gerenciam o estado dos objetos, controlam sua validade e registram as transações canceladas ou confirmadas. Isso foi Implementado pelos autores usando o *Sharded Byzantine Atomic Commit* (S-BAC) - um protocolo que compõe o acordo BFT existente e primitivas de commit atômico de uma maneira nova.

Sharded Byzantine Atomic Commit

O protocolo tem sua origem combinando propriedades de outros dois protocolos primitivos: Byzantine agreement que permite que os nós entrem em consenso se a partição tiver até 1/3 nós maliciosos e Atomic commit uma partição apenas confirma sua transação após esta ser aceita pelas demais partições interessadas.

A Figura 2.10 mostra os passos para aceitar uma transação, uma vez que a partição que propôs uma transação T ele a transmite na rede AllPrepared(commit,T). A transação se torna ativa até que o primeira partição interessada a aceite através do LocalPrepared(commit,T) que trava a transação reservando os recursos. Nesse estado, se alguma partição negar com LocalPrepared(abort,T) os recursos são liberados e a transação retorna ao estado ativo. Uma vez que todos as partições confirmem a transação. O AllPrepared(commit,T) é confirmado e a mesma é registrada no bloco.

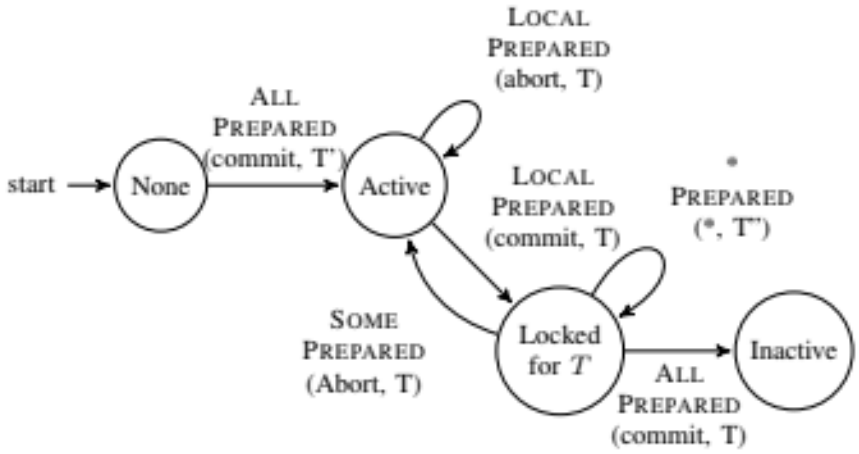


Figure 2.10. Estados de uma transação [Al-Bassam et al. 2017]

O protocolo S-BAC garante três propriedades-chave, que refletem a segurança do Chainspace: vivacidade, consistência e validade.

1. **Vivacidade:** Uma transação T que é proposta a pelo menos um nó honesto, eventualmente resultará em uma decisão única, ou seja, todos os nós decidindo entre accept(commit, T) ou accept(abort, T).

Baseado na propriedade de vivacidade do acordo byzantino. Assumindo que "prepare(T)" foi dado a um nó honesto, será enviado "prepared(commit, T)" ou "prepared(abort, T)" dos nós da partição BFT para os os nós da partição interessada. Ao receber a mensagem, a nova partição agendará um "prepare(T)" para seus nós eventualmente gerando a decisão adequada.

2. **Consistência:** Não serão confirmadas duas transações conflitantes, ou seja, transações que compartilham a mesma entrada. Além disso, existe uma execução sequencial para todas as transações.

   Uma transação só é confirmada se alguns nós concluírem "ALLPREPARED(commit, T)", e para isso esses nós devem receber "LOCALPREPARED(commit, T)" dos nós envolvidos. Duas transações concorrentes possuem um subgrupo de nós envolvidos em comum e uma vez que uma delas receba o accept local a outra será bloqueada, recebendo local abort, até que a primeira seja confirmada ou rejeitada.

3. **Validade:** Uma transação só pode ser confirmada se for válida de acordo com os verificadores de contrato inteligentes combinando as características dos procedimentos que executa.

   Uma transação só é confirmada se todos os nós interessados garantiram a regularidade com seu "local accept" assim uma vez que a transação receber "AllPrepared(commit,T)" ela é válida.

2.5.1.4. Cross-shard transaction processing

Os nós comunicam aos shards envolvidos para decidirem se aceitam ou rejeitam uma transação via consenso cross-shard. Em vez de uma abordagem orientada ao cliente, o ChainSpace executa o protocolo S-BAC colaborativamente entre todos os comitês envolvidos. Isso é alcançado ao fazer com que todos os comitês atuem como um gerente de recursos para as transações que eles gerenciam. [Wang et al. 2019]

2.5.1.5. Epoch reconfiguration

Embora a reconfiguração não seja definida uma vez que a formação do shard fica em aberto, o Chainspace oferece uma ferramenta de auditabilidade para identificar nós maliciosos nesta etapa.

Uma partição maliciosa (com mais de f nós defeituosos) que tentou adicionar uma transação ou objeto inválido no estado de outra partição, pode ser detectado por um auditor realizando uma auditoria completa do sistema Chainspace. Um par hash-chains de partições distintas são válidos se:

- A reexecução das transações levam ao mesmo estado.
- todas as mensagens prepared(Transaction,*) são compatíveis.

Se duas hash-chains se provarem incompatíveis é possível determinar qual é a desonesta isolando as assinaturas.

2.5.2. Ominiledger

O OmniLedger[Kokoris-Kogias et al. 2018] parte de um modelo mais simples de *sharding*, denominado SimpleLedger, e estuda os problemas para encontrar a solução ótima. O SimpleLedger evolui em epochs, e utiliza um coordenador para gerar um valor aleatório que cada nó usa para determinar sua partição. As melhorias necessárias para chegar ao OmniLedger desse modelo são:

- Performance:
  1. ByzCoinX: consenso Byzantino robusto;
  2. Podar a *Blockchain* das partições;
  3. Validação Trust-but-Verify;
- Segurança
  1. *Sharding* através de aleatoriedade distribuida;
  2. Transição de epoch suave;
  3. Atomix: transação entre partições atômica;

2.5.2.1. Identity establishment and committee formation

Para participar de um epoch *e*, o nó deve registrar-se em uma blockchain identidade no epoch *e - 1*, para geração da identidade pode-se utilizar qualquer mecanismo resistente a um ataque sybil, como por exemplo, Proof of Work, Proof of Stake, entre outros.

Um passo importante para a descentralização é definir como associar os validators(nós) às partições. Para isso é necessário um protocolo de geração de distribuição aleatória que seja Imparcial, Imprevisível, Verificável por terceiros e Escalável. O protocolo escolhido foi o RandHound [Syta et al. 2016].

RandHound assume um líder honesto que é responsável por coordenar a execução do protocolo e para fazer a aleatoriedade produzida disponível para os outros. No entanto, nós nem sempre podemos garantir que um líder honesto será selecionado. Cada vez que um líder controlado pelo adversário é eleito e executa RandHound, o adversário pode escolher aceitar a saída aleatória e a atribuição de partição produzida por ela, ou desistir dessa e tentar novamente na esperança de uma mais favorável porém ainda aleatória. Ao atingir o número máximo de tentativas o líder falha, e por padrão este líder não poderá participar da próxima eleição.

No início de cada epoch, cada validator gera um ticket usando uma função aleatória e verificável, e compartilha com os outros nós, o ticket válido com menor valor é escolhido como novo líder.

2.5.2.2. Overlay setup for committees

Como veremos mais adiante, o OmniLedger faz uma reconfiguração gradual dos shards, ou seja, os nós vão entrando aos poucos nos shards para os quais foram designados. Uma

vez que um nó esteja pronto, ele envia uma mensagem ao líder do shard informando-o, o líder então permite a entrada do nó no shard.

2.5.2.3. Intra-committee consensus

Para o consenso intra-shard utiliza-se o ByzCoinX, uma variante do ByzCoin, que utiliza padrões de comunicação mais robustos para processar transações de forma eficiente dentro dos shards, mesmo se alguns dos validadores falharem, e que resolve dependências no nível da transação para obter uma melhor paralelização de blocos.

O OmniLedger pode, opcionalmente, utilizar uma arquitetura diferente nos shards para permitir o processamento em tempo real de transações. Ela consiste no uso de validadores otimistas, que formam um grupo menor e portanto mais rápido para chegar a um consenso. Os blocos otimistas produzidos por tais validadores, são posteriormente verificados pelo núcleo de validadores, que é um grupo muito maior de validadores, este grupo portanto é mais lento para processar as transações, porém mais seguro. Esta abordagem foi chamada de Trust-But-Verify, seu esquemático pode ser visto na figura 2.11.

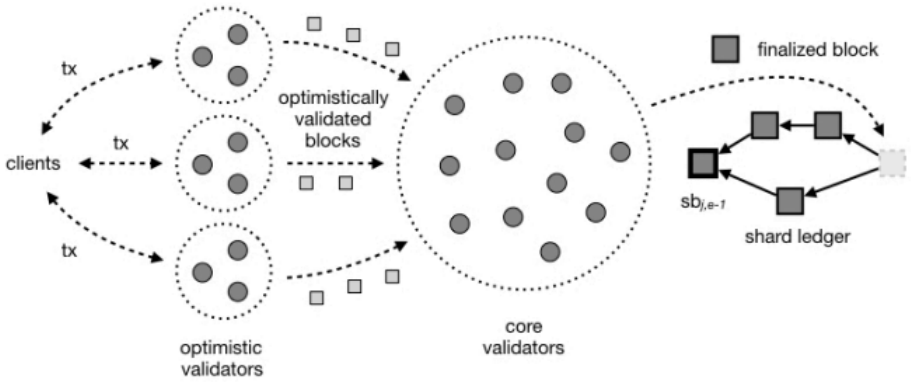


Figure 2.11. Trust But Verify [Kokoris-Kogias et al. 2018]

2.5.2.4. Cross-shard transaction processing

Para processar transações cross-shard, garantindo sua atomicidade, foi desenvolvido o Atomix, que usa o modelo de estado UTXO, onde o balanço de um usuário é armazenado na forma de tokens de valor não gasto (disponível). O protocolo tem três etapas:

1. **Initialização.** Um cliente cria uma transação entre partições que gasta o UTXO de algumas partições de entrada (IS) e cria novos em algumas partições de saída (OS);

2. **Lock.** Cada IS verifica se o UTXO pode ser gasto. Se possível o recurso é travado e a partição divulga proof-of-acceptance, caso contrário a partição divulga proof-of-rejection;

3. **Unlock.** Há duas opções nesta etapa, Unlock to Commit que consome os recursos travados para criar novos no destino da transação e Unlock to Abort, se um IS rejeitou a transação o cliente comunica aos demais para liberar os recursos.

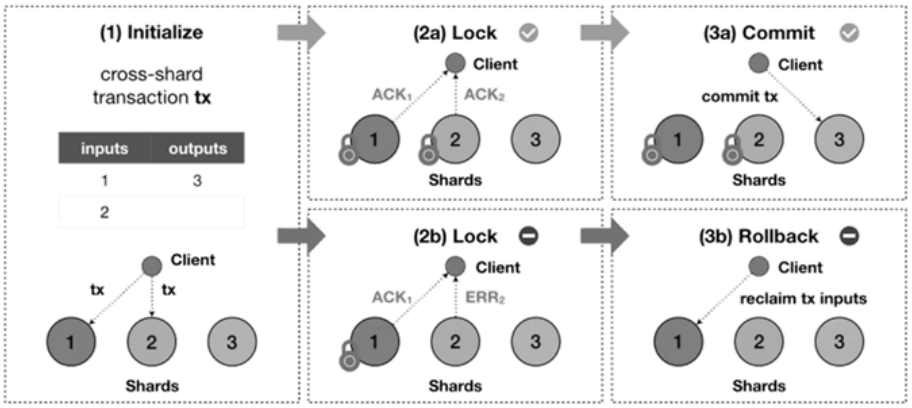


Figure 2.12. Protocolo Atomix [Kokoris-Kogias et al. 2018]

2.5.2.5. Epoch reconfiguration

Para manter a operabilidade durante as fases de transição, o OmniLedger troca os validadores de cada shard gradualmente por epoch. Isso permite que os operadores restantes continuem disponíveis (no cenário honesto) aos clientes enquanto os validadores recém-associados estão inicializando. A fim de alcançar esta operação contínua, podemos trocar no máximo $1/3$ do tamanho da partição, porém quanto maior for o lote, maior é o risco de que o número de validadores honestos remanescente seja insuficiente para chegar a um consenso e mais estresse é gerado pela inicialização de novos validadores a rede.

2.5.3. RapidChain

O RapidChain[Zamani et al. 2018] consiste em três componentes principais: Bootstrap, Consenso e Reconfiguração. Assim como muitos métodos de *sharding*, o RapidChain é executado em períodos de tempo fixos chamados epochs. No primeiro epoch, um protocolo de bootstrapping único é executado, permitindo que os participantes concordem em um comitê. Esse comitê, que chamamos de comitê de referência, é responsável por conduzir eventos de reconfiguração periódicos entre os epochs. Cada epoch consiste em várias iterações de consenso seguidas por uma fase de reconfiguração.

2.5.3.1. Identity establishment and committee formation

As identidades são geradas usando um mecanismo gerador de identidades Sybil-resistant como [Andrychowicz and Dziembowski 2015]. Este mecanismo requer que cada nó resolva um puzzle computacionalmente difícil com as identidades geradas localmente.

O conjunto inicial de participantes inicia o RapidChain executando uma eleição de comitê protocolo, onde todos os nós concordam com um grupo de $O(\sqrt{n})$ nós, aos

quais nos referimos como grupo raiz. O grupo é responsável por gerar e distribuir uma sequência de bits aleatórios que são usados para estabelecer um comitê de referência de tamanho O (log n). Em seguida, o comitê de referência cria k comitês C1, ..., Ck cada um de tamanho O (log n). A fase de bootstrap é executada apenas uma vez no início do RapidChain.

Estabilidade do Bootstrapping

Suponha que haja n nós e uma fração constante, 1/3 desses nós estão corrompidos. No fim do protocolo de bootstrapping do RapidChain, quase todos (percentil 99) os nós não corrompidos concordam com a string aleatória r_0 e, consequentemente, com todos os comitês do *sharding* com probabilidade constante maior que 0.

2.5.3.2. Overlay setup for committees

Na etapa de bootstrapping, os nós do mesmo comitê descobrem um ao outro através de algoritmo peer-discovery. Os nós que são realocados pela regra de cuckoo ou que entram na rede tardiamente baixam fazem download dos dados do seu novo comitê.

Cada transação tx é submetida por um usuário do sistema a um grupo arbitrário de nós do RapidChain. Esses nós roteiam tx, por meio de um protocolo de roteamento entre comitês, a um comitê responsável pelo armazenamento de tx. O protocolo de roteamento inspirado no Kademlia [Maymounkov and Mazieres 2002]. O protocolo Kademlia foi aplicado no RapidChain a nível de comitê. Especificamente cada comitê mantém endereço dos *logn* comitês mais próximos. Assim, o responsável por identificar o comitê que deve armazenar tx, só comunica com um número logaritmo de comitês para encontrá-lo.

2.5.3.3. Intra-committee consensus

Assim que os membros de cada comitê concluem a reconfiguração da epoch atual, eles aguardam para que usuários externos enviem suas transações. Cada usuário envia suas transações para um subconjunto de nós (encontrado através de um protocolo de descoberta P2P) que agrupa e encaminha as transações à comissão responsável pelo seu tratamento. O comitê executa um protocolo de consenso Bizantino dentro do comitê para aprovar a transação e adicioná-la à sua razão.

Estabilidade do consenso intra-comitê

- **Safety:** Com fração de nós corrompidos $f = 1/2$. Provamos safety para um cabeçalho de bloco específico proposto pelo líder na iteração i. Suponha que o nó P é o primeiro nó honesto a aceitar um cabeçalho Hi para i. Em todas as rodadas da iteração i ou todas as iterações após i, nenhum líder pode construir uma proposta segura para um cabeçalho diferente de Hi uma vez que ele não pode obter votos suficientes de nós honestos em um valor que não é seguro. É impossível finalizar uma raiz Merkle de um bloco que está associada a dois blocos diferentes.

- **Liveness:** Em primeiro lugar, observe que todas as mensagens no sistema são mantidas por meio de assinaturas digitais. Todos os nós honestos aceitarão todos os blocos com o valor seguro pendentes, ou eles já os aceitaram, assim que o líder para a altura do bloco atual for honesto. Como líderes são escolhidos aleatoriamente e a aleatoriedade é imparcial, cada comitê terá um líder honesto a cada duas rodadas na expectativa. O líder honesto enviará propostas válidas para todos os blocos pendentes para todos os nós que é seguro propor e tem uma prova válida. Assim, todas as réplicas honestas já aceitaram o esse valor ou irão aceitá-lo uma vez que é seguro.

2.5.3.4. Cross-shard transaction processing

A transação cross-shard no RapidChain baseia-se amplamente no esquema de roteamento inter-comitê. O esquema permite que os usuários e líderes de comitês identifiquem rapidamente a quais comitês eles devem enviar sua transação. Em transações cross-shard em RapidChain, cada transação cria três transações diferentes para criar o mesmo resultado. Todas as três sub-transações são single-shard. Em caso de falha em algumas das sub-transações o RapidChain usa um mecanismo de rollback atômico.

2.5.3.5. Epoch reconfiguration

A reconfiguração permite que novos nós estabeleçam identidades e se juntem aos comitês existentes, e garante que todos os comitês mantenham sua resiliência de 1/2. A reconfiguração do RapidChain se baseia na regra Cuckoo, quando um novo nó se junta a rede ele é designado a um comitê aleatório e alguns nós desse comitê são realocados para os demais. Além disso, o RapidChan separa seus comitês em dois grupos pelo tamanho, sempre que um novo nó se junta, ele é alocado a um comitê do grupo dos maiores e a realocação é feita para o grupo dos comitês menores mantendo o tamanho dos comitês mais estáveis.

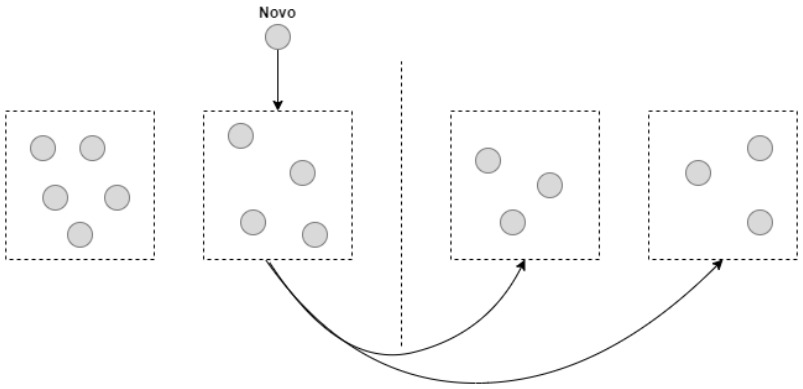


Figure 2.13. Regra de Cuckoo

Estabilidade do Epoch

A função que permite calcular a probabilidade de falha (ter mais nós corrompidos que um dado limite) de um epoch:

$$Pr[X \geq [m/2]] < \sum_{x=[m/2]}^{m} \frac{txn - tm - x}{nm} \tag{2}$$

Onde m é o tamanho do comitê, n é o número de nós da rede e t é o limite superior de nós corrompidos na rede.

Ao contrário da distribuição binomial, a distribuição hipergeométrica depende diretamente do total tamanho da população. Uma vez que n pode mudar ao longo do tempo em uma rede de adesão aberta, a probabilidade de falha pode ser afetada. Para manter a probabilidade de falha desejada, cada comitê em RapidChain executa um consenso em intervalos pré-determinados para concordar com um novo tamanho do comitê.

Para limitar a probabilidade de falha de cada epoch, calculamos o total de comites k = n/m, onde cada um pode falhar com a probabilidade calculada anteriormente. Portanto, a probabilidade de fracasso de cada epoch é:

$$Pepoch < Pbootstrap + k \cdot Pcommittee \tag{3}$$

Estabilidade do Reconfiguration

Assumimos que a qualquer momento durante o protocolo, a fração de nós corrompidos para nós honestos é ε. Nós também assumimos que o protocolo começa a partir de um estado estável com n nós particionados em m comitês que satisfaz as condições de equilíbrio e honestidade. Definimos o conjunto de comitês ativos como os comitês de m/2 com maior número de nós.

Para provar o teorema, é suficiente provar as propriedades de balancing e honesty para qualquer comitê.

- **Balanceamento:** O número máximo de nós em cada comitê é $c\log n + c/2(1+\delta)(3-\frac{t}{n}+\frac{t}{n-t}k)\log n$ e o mínimo é $c/2(1-\delta)\log n$;
- **Honestidade:** Escolhendo k tal que $\frac{t}{n-t} < 1-1/k$ qualquer comitê tem $(1-t/n)(1-\delta)c\log nk/2$ nós honesto e $\frac{t}{n-t}(1+\delta)c\log nk/2$ nós corrompidos com alta probabilidade. Observe que esses valores são calculados para o pior cenário, quando o adversário mirou no comitê de tamanho $(c\log n)k/n$.

2.5.4. Cycledger

O CycLedger [Zhang et al. 2020] é uma proposta mais recente de Sharding, por conta disso, ele pôde fazer uma análise crítica sobre as propostas anteriores. Nesta análise destacou-se três pontos que são considerados deficiências destas propostas, são eles: todos os nós sem falhas devem se conectar bem um com o outro; há uma grande perda de eficiência quando a honestidade dos líderes dos shards é questionável; e não há um incentivo explícito para os nós participarem ativamente do protocolo.

Procurando contornar tais deficiências, o ClycLedger se apresenta como uma solução que: é escalável, pois faz sharding completo; se mantêm robusta mesmo quando os líderes dos shards são desonestos, dado que, cada shard possui um conjunto de nós que supervisionam o líder do shard e podem atuar como suplentes se necessário; e que pretende fornecer incentivos o suficiente para que os nós participem de forma honesta do protocolo, fazendo uso de um sistema de reputação que associa a reputação ao poder computacional do nó, fazendo com que os nós que contribuem mais para a rede sejam devidamente recompensados para se manterem honestos. A proposta assume que não há mais que 1/3 de nós maliciosos, e que mais de 1/2 dos nós de um shard são honestos.

O CycLedger funciona em rounds. Sabendo que cada nó possui um par (chave pública, chave privada) e uma reputação w_r. A cada novo round r é escolhido em $r-1$ um comitê árbitro, que irá gerenciar a identidade dos nós, produzir a próxima aleatoriedade e propor o bloco do round r. Ao mesmo tempo, os outros nós são designados para os m shards, idealmente formando shards de tamanho $c = O(log^2 n)$, dessa forma temos que $n = m * c$. Cada shard possui um líder, λ líderes em potencial(conjunto de sub líderes) e $c - \lambda - 1$ membros. A figura 2.14 demonstra a estrutura discutida. É importante ressaltar que o bloco gerado no round $r-1$ contém também os endereços dos membros do comitê árbitro, dos líderes e sub líderes de cada shard para o round r.

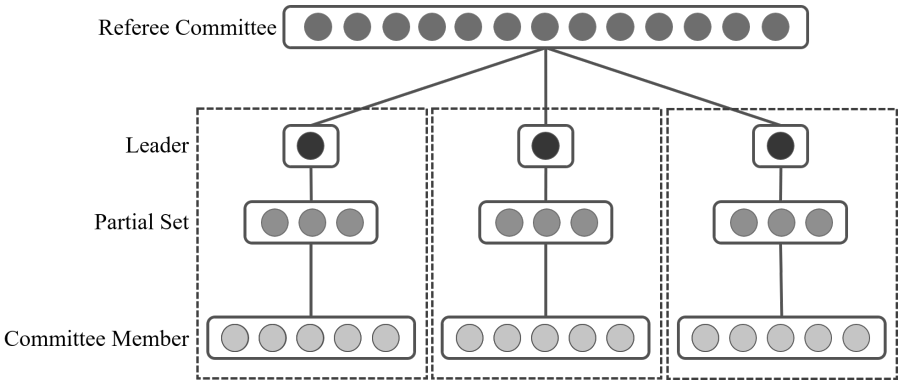


Figure 2.14. Comitês existentes no CycLedger [Zhang et al. 2020]

2.5.4.1. Identity establishment and committee formation

A identidade do nó é estabelecida utilizando um PKI(Public Key Infrastructure) para dar a cada nó um par (chave pública, chave privada). Para que um nó saiba para qual shard será designado é utilizado uma VRF(Verifiable Random Function), uma função que retorna um hash e uma prova, que junto com a chave pública do nó comprova que ele gerou aquela hash.

Papel dos Membros dos Shards

- Determina o id do seu shard usando a VRF;
- Envia sua chave pública, endereço, a hash e a prova pi gerada pela VRF, para o líder e sub líderes do shard;

- Quando o nó i recebe a lista de membros do shard, do líder ou de um sub líder, ele envia os mesmos itens enviados no passo anterior para todos os nós da lista;
- Quando i recebe a chave pública j de um nó j, ele verifica se j está no mesmo shard que ele utilizando a chave e a prova pi.

Papel dos Líderes e Sub Líderes dos Shards

- Mantém uma lista <Chave Pública, Endereço>. Inicialmente ela contém apenas o líder e os sub líderes;
- Quando recebe a chave pública do nó j, verifica se j pertence ao shard utilizando a chave e a prova pi, se confirmado, j é adicionado à lista de nós daquele shard.

2.5.4.2. Overlay setup for committees

No caso do CycLedger, esta etapa é feita em paralelo com a formação dos shards, pois, como vimos, os líderes e sub líderes de cada shard mantém uma lista de membros do shard, que é atualizada conforme novos membros se identificam para os líderes do shard. A partir desta lista os membros estabelecem a comunicação entre si.

Assume-se que há uma boa conexão enquanto os líderes e seus conjuntos são linkados. Supõe-se também que cada líder ou membro do conjunto de sub líderes está conectado com todo o comitê árbitro daquele round, enquanto as soluções anteriores supunham uma boa conexão entre todos os nós honestos. Por último supõe-se que a comunicação dentro do shard é síncrona considerando um delay delta, o que é plausível, dado que os shards possuem apenas centenas de nós. Os líderes e sub líderes de todos os shards também possuem uma comunicação síncrona entre si, porém considera-se um delay maior.

2.5.4.3. Intra-committee consensus

Para realizar o consenso, o CycLedger faz uso de algoritmo que ele chama de semi-commitment exchange, que funciona da seguinte forma: Dada uma transação, primeiramente o líder do shard faz o multicast de <r, M, H(M), sn> com a tag PROPOSE, onde *r* é o número do round, *M* é a mensagem original, *H(M)* é o resumo da mensagem e *sn* é o número de sequência da mensagem, que é único. Quando um nó *i* recebe *M* e *H(M)*, ele verifica a corretude do resumo, o número do round e verifica também se sn é único e então faz o broadcast <r, sn, H(M), i> com a tag ECHO e retransmite o PROPOSE para os outros membros. Caso o nó i recebe de mais da metade dos membros , um ECHO e um PROPOSE idênticos aos que ele produziu e recebeu respectivamente, ele irá enviar um <r, sn, H(M), i> com a tag CONFIRM para o líder do shard. Caso algum nó honesto perceba que o líder enviou mensagens diferentes, então ele informa aos outros membros do shard e o consenso é interrompido, o líder é então expulso e um sub-líder tomará o seu lugar.

O consenso intra-shard é dividido em 5 passos:

1. Após receber transações de usuários externos, cada líder cria uma lista de transações internas;
2. O líder faz o broadcast da lista para todos os membros do shard;
3. Após receber a lista de transações, para cada transação o nó vota, Yes, No ou Unknow, onde Unknow é quando o nó não consegue avaliar a transação. Feito isso ele envia a lista de votos para o líder;
4. O líder então pega as transações que obtiveram maioria de votos Yes, e executa o algoritmo descrito na subseção anterior;
5. Por último, com aprovação de ao menos metade dos membros do shard, as transações são aceitas.

2.5.4.4. Cross-shard transaction processing

O consenso Inter-Shard funciona de maneira análoga ao intra-shard, porém, a lista de transações criada pelo líder possui apenas transações cross-shard. Chamemos então esta lista de $TXList_{i,j}$, nesta lista temos transações entre os shards i e j. Após o shard i chegar a um consenso a respeito das transações em $TXList_{i,j}$, o líder do shard i envia o consenso e a lista de membros do shard i para o líder e sub líderes do shard j. O líder de j então chegará a um consenso sobre a lista recebida e enviará o consenso e sua lista de membros ao líder e sub líderes de i.

Após a conclusão do consenso, intra-shard ou cross-shard, o líder garante a cada membro votante uma pontuação de acordo com a proximidade dos votos daquele membro com o resultado do consenso. Dessa forma cria-se uma relação entre a reputação e o poder computacional. O líder então organiza essa pontuação em uma lista e faz o broadcast dela para os membros do shard, para que a reputação dos membros seja atualizada.

2.5.4.5. Epoch reconfiguration

Na proposta em questão, a Epoch reconfiguration é dividida em duas etapas: A seleção do comitê árbitro, do líder e sub líderes do shard; e da geração e propagação do bloco. Isto acontece pois, o endereço dos nós escolhidos para atuar como parte do comitê árbitro ou como líder/sub líder de um shard no próximo round, deve constar no bloco gerado no round atual, permitindo que esta informação esteja disponível durante a formação dos shards no próximo round.

Seleção do Comitê Árbitro, Líderes e Sub Líderes

Usando uma geração de aleatoriedade distribuída, os membros do comitê árbitro do round r, chamaremos este comitê de C_r, eles irão gerar a seed que será usada no round r + 1. Os

nós que desejam participar do próximo round devem resolver um puzzle PoW, ao resolver, eles devem enviar a sua solução para o C_r, o comitê então irá registrar a identidade do nó. Desta forma, C_r está ciente de todos os participantes do próximo round, ele então escolhe como líderes dos shards os m nós de maior reputação. Fazendo uso da aleatoriedade do round atual, C_r irá definir o comitê árbitro e os sub líderes, de cada shard, do próximo round.

Geração e Propagação do Bloco

O comitê árbitro após receber todas as transações, verifica e encapsula todas as legítimas junto com a aleatoriedade gerada para o próximo round, os participantes do próximo round, a reputação atualizada dos nós, o comitê árbitro escolhido para o próximo round, bem como os líderes e sub líderes. Uma vez que este bloco foi gerado ele é liberado para consulta para todos os nós. Cada nó então chega a um consenso sobre as transações das quais participa após ver o bloco gerado, e o líder envia estas transações para o comitê árbitro do próximo round.

2.5.5. Sharper

SharPer é um sistema de blockchain permissionada. Esse sistema agrupa os nós em clusters e fragmenta os dados e a razão da blockchain. Com isso permite o processamento paralelo de transações. Cada bloco contém uma única transação e é armazenado em cada nó dos clusters envolvidos. Com as transações entre shards a razão é formada como grafo acíclico dirigido. A figura 2.15 mostra um exemplo do grafo formado com as razões do SharPer a partir de um bloco genesis λ

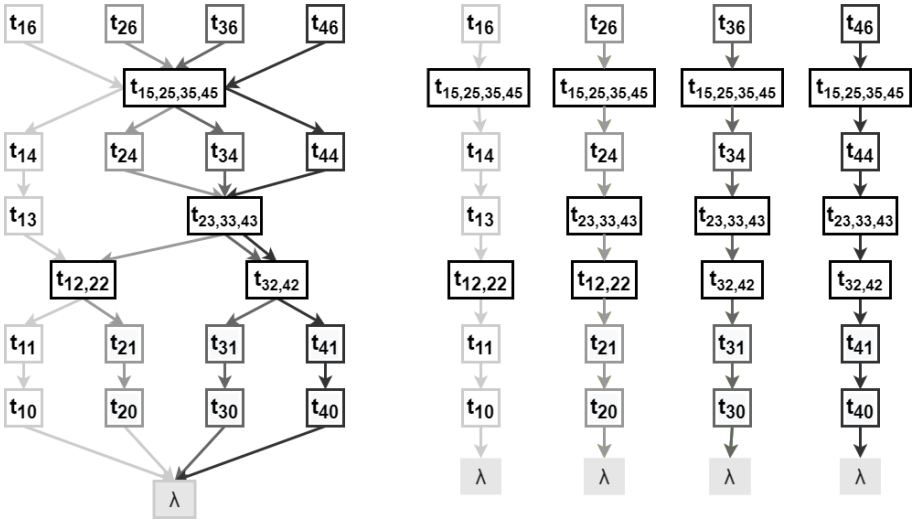


Figure 2.15. Blockchain do SharPer com quatro clusters [Amiri et al. 2019]

2.5.5.1. Identity establishment and committee formation

Em Blockchains não permissionadas, cada cluster consiste em centenas de nós. Este tamanho é necessário para atingir, com boa probabilidade, o valor de 1/3 de nós com

falha. Em outros modelos permissionados o tamanho necessário do cluster é bastante reduzido. O AHL por exemplo garante a segurança com 80 nós. O sharPer fornece um modelo determinístico que fornece segurança com valores próximos ao mínimo para superar a falha. Em um sistema de 16 nós e com falhas bizantinas onde f = 1 e , o tamanho do cluster é 16/4 (3f+1). O último cluster será um pouco maior se a divisão não for exata.

Os protocolos de consenso tolerante a falhas demandam um grande volume de mensagens. Com isso o tamanho reduzido do cluster diminui a latência no sistema. Para diminuir ainda mais a latência, os clusters são distribuídos de acordo com a sua localização geográfica. Nós próximos são agrupados no mesmo cluster.

2.5.5.2. Overlay setup for committees

O SharPer usa comunicação ponta a ponta bidirecional e autenticada. Com isso, um nó malicioso não pode forjar uma mensagem de um nó correto. As mensagens ainda podem conter assinaturas de chave pública e digests para reforçar a integridade. Os nós no SharPer seguem o modelo de falha de parada ou falha bizantina. No modelo de falha de parada, os nós não podem conspirar, mentir ou tentar subverter o protocolo. Com apenas falhas por parada são necessários 2f+1 nós para garantir a segurança com falha de f nós. Enquanto que o modelo resistente a falha bizantina, requer 3f+1 para superar o mesmo número de nós defeituosos.

2.5.5.3. Intra-committee consensus

Crash-Only Nodes

O SharPer usa uma variação do algoritmo Paxos [Lamport et al. 2001] denominada de multi-Paxos. Neste algoritmo um líder estável é pré-escolhido para coordenar o protocolo. Os clientes enviam suas requisições assinadas para o líder. O líder atribui o número de sequência da transação e propaga uma mensagem de propostas para os nós do cluster. O líder aguarda por f nós aceitarem a proposta. Com o líder, f+1 nós aceitaram a proposta garantindo que ao menos 1 nó sem falha aceitou a proposta.

Byzantine Nodes

Para o modelo resistente à falha byzantina, o SharPer usa PBFT [Castro et al. 1999] para garantir a segurança na presença de até f nós maliciosos. Para isso, cada participante do protocolo se comporta como um líder e tenta aprovar a proposta. O protocolo inicia com um cliente solicitando uma transação a um líder que repassa para todos os nós do cluster (pre-prepare). De posse de um proposta válida, cada nó envia uma mensagem de accept aos demais. Os nós do cluster esperam por 2f dessas mensagens, 2f+1 com o próprio, para propagar sua mensagem de commit aos demais. Mais uma vez, se nó receber 2f propostas de commit, ele responde ao cliente. A proposta é aprovada com f+1 respostas ao cliente.

Se um tempo após enviar a proposta ao primeiro líder, o cliente não receber nenhuma resposta ele envia a propostas a todos os nós do protocolo. Neste cenário é provável que o primeiro líder tenha falhado.

2.5.5.4. Cross-shard transaction processing

Crash-Only Nodes

As transações cross shards são processadas de forma similar. O líder envia a proposta para todos os nós dos clusters envolvidos e a aguarda por f+1 nós de cada cluster aceitarem a proposta. Devido a latência na rede, as mensagens para os demais clusters podem ser conflitantes. Um nó pode não ter recebido a última mensagem de seu líder ou mais de um líder divulgar uma proposta no mesmo instante. Para lidar com esse problema o líder que propõe a transação cross shard pode se comunicar com o líder de cada cluster. Então cada cluster processa a mensagem como um transação interna, resolvendo o conflito.

Byzantine Nodes

O protocolo tolerante a falhas bizantinas é semelhante ao caso com apenas falha por parada. acordo de todos os envolvidos. Ainda é necessário que todos os clusters concordem com a proposta, embora o tamanho do quorum passa a ser 2f + 1. Além disso, devido ao possível comportamento malicioso do nó primário, todos os nós, de cada cluster envolvido, enviam as mensagens de accept e commit para cada outro nó.

2.5.5.5. Epoch reconfiguration

O Sharper usa uma distribuição determinística baseada na localização geográfica. Além disso, tem como base uma Blockchain permissionada, onde cada nó possui sua identidade única. Desta forma o modelo descarta a necessidade da reconfiguração e da delimitação de tempo, epochs.

2.6. Outras Soluções

O Pyramid e LightChain buscam atingir objetivos específicos ao modificar a estrutura do *sharding*. O pyramid sobrepõe as partições para otimizar o processamento de transações entre partições. Uma vez que um nó participe de 2 partições ele é capaz de conferir transações entre nós de suas partições. Mesmo que eles estejam, cada um em uma partição.

O LightChain tem por objetivo reduzir a carga de memória gerada pelo crescimento constante da *Blockchain*. A ideia é dividir a cadeia entre as instâncias de execução com auxílio de uma tabela hash distribuída. Esse modelo acaba por fragmentar a comunicação pois as instâncias só enxergam os nós em sua tabela.

2.6.1. Pyramid

O Pyramid [Hong et al. 2021] propõe um novo modelo de *sharding*, o *Layerd sharding*, no qual as partições passam as se sobrepor ao invés de serem completamente isoladas. Desta forma, alguns nós participaram de mais de uma partição. Os nós que fazem parte da sobreposição podem verificar e processar as transações entre partições, envolvendo as

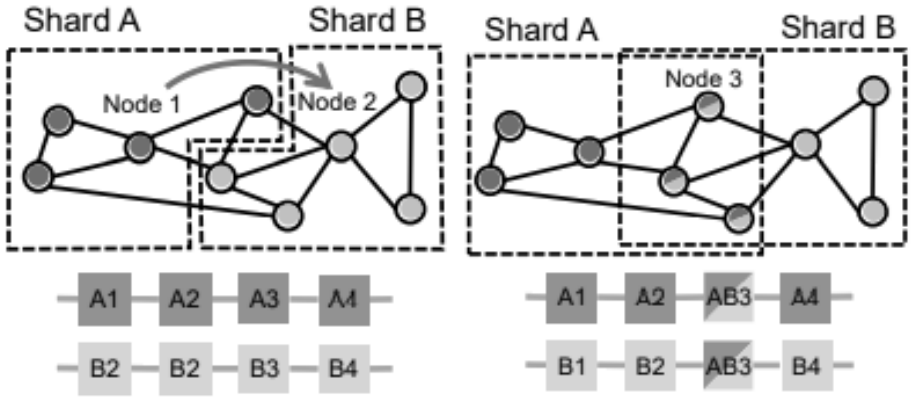


Figure 2.16. Modelo Layered *sharding* [Hong et al. 2021]

diferentes partições que eles participam, de forma direta e eficiente, ao invés de dividir a transação em várias sub-transações, como é feito no *sharding* completo.

As partições do Pyramid podem ser classificadas de duas formas de acordo com seu tipo, i-*shard* ou b-*shard*. Os i-*shards* possuem apenas os nós responsáveis por lidar com transações internas, enquanto os b-*shards* incluem nós que fazem uma ponte entre múltiplos i-*shards*, permitindo que resolvam as transações entre esses i-*shards*. Além desse papel, os nós pertencentes aos b-*shards* também são capazes de lidar com transações internas. No exemplo abaixo o b-*shard* C é responsável por processar as transações entre os i-*shards* A e B.

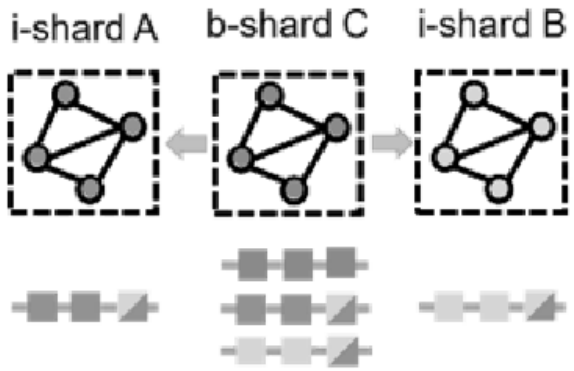


Figure 2.17. Funcionamento dos b-*shards* [Hong et al. 2021]

Embora esse modelo melhore o processamento de transações entre partições ele traz uma complexidade maior para estrutura, pelo fato dos nós terem funções diferentes, o que gera alguns desafios. O primeiro desafio é a construção das partições. Quantas partições são necessárias e quais nós devem ser designados a cada partição.

O segundo desafio é o protocolo de consenso, como gerar e verificar blocos. Os Sistemas que utilizam *sharding* geralmente adotam PoW ou um protocolo BFT. Por conta do *Layered sharding* algumas partições no pyramid são responsáveis por criar blocos para transações entre partições e os blocos gerados precisam ser enviados às partições correspondentes para confirmação, o que torna o design do protocolo de consenso mais desafiador.

2.6.1.1. Identity establishment and committee formation

A formação das partições é realizada em 3 etapas:

1. **Geração de aleatoriedade:** O funcionamento da Pyramid acontece em períodos de tempo fixos chamados de epoch, no começo de cada epoch a aleatoriedade é gerada atravez de um protocolo externo como verifiable random function [Micali et al. 1999] ou verifable delay function [Boneh et al. 2018];

2. **Participação:** Para cada nó, antes de se juntar a um epoch, um novo puzzle PoW é gerado baseado na sua chave pública e na aleatoriedade do epoch. Para participar de um epoch, o nó precisa resolver seu puzzle;

3. **Atribuição:** A cada nó admitido é designado uma ID de partição aleatoriamente, baseado na identidade do nó e na aleatoriedade gerada no epoch. A ID da partição pode representar um i-*shard* ou um b-*shard*, sendo que, os b-*shards* correspondem a um número de i-*shards*;

Na segunda etapa, após resolver o puzzle, o nó precisa colocar a sua solução numa *Blockchain* identidade para registrar sua identidade. Essa *Blockchain* é uma *Blockchain* baseada em PoW cuja função é registrar a identidade dos nós. A dificuldade do puzzle gerado para um nó que quer se juntar a um epoch é ajustada de acordo com a quantidade de nós registrados no epoch anterior.

Uma característica interessante do Pyramid é que podemos definir um parâmetro de escalabilidade w, onde, $0 < w <= 1$, e quanto maior w for, maior será a quantidade de b-*shards* criada. Dessa forma, aumentamos o throughput e diminuímos a latência das transações, porém, aumentamos o uso de armazenamento, dado que o b-*shard* armazena a sua blockchain e todas as quais ele faz a ponte.

2.6.1.2. Overlay setup for committees

O Pyramid considera que todos os nós da rede estão conectados como uma rede ponto-a-ponto parcialmente síncrona. E como os resultados da etapa de atribuição, visto na subseção anterior, são públicos e podem ser verificados usando a aleatoriedade e a Blockchain Identidade, os nós podem trocar mensagens com os outros membros do shard para o qual foi designado.

2.6.1.3. Intra-committee consensus

No pyramid, cada epoch consiste de turnos de consenso. A cada turno de consenso, um líder para a partição será aleatoriamente eleito. Se a partição for uma i-*shard* o bloco proposto inclui as transações internas e um consenso aos moldes das *Blockchain complete sharding* será executado.

2.6.1.4. Cross-shard transaction processing

Uma transação é um pagamento entre duas contas, chamadas remetente e receptor. Se o remetente e o receptor de uma transação estão localizados em diferentes partições, essa transação se trata de uma transação entre partições. A validade de cada transação pode ser dividida em source validity e result validity.

1. **Source Validity:** denota se o estado do remetente satisfaz a condição para a transação, isto é, o remetente possui dinheiro suficiente para realizar a transação;
2. **Result Validity:** denota se o estado do receptor satisfaz o resultado da transação,isto é, o receptor recebe dinheiro adequado;

Nós em um i-*shard* guardam apenas uma *Blockchain* enquanto nós em b-*shards* guardam múltiplas, por conta disso, no pyramid, nós que pertencem aos b-*shards* podem verificar a source validity e a result validity de transações entre partições.

Design de Blocos Cross-Shard

O cabeçalho inclui os hashes dos blocos pais dos i-*shards* relacionados e a raiz da árvore Merkle do corpo. Embora um bloco *cross-shard* válido possa ser proposto por qualquer nó em um b-*shard*, para garantir a consistência dos estados entre o b-*shard* e seus i-*shards* relacionados, o bloco *cross-shard* precisa ser confirmado pelos outros nós no b-*shard* e seus i-*shards* relacionados, através do consenso *layered sharding* que será visto a seguir. A figura 2.18 ilustra um bloco cross-shard relacionado aos i-shards A e B.

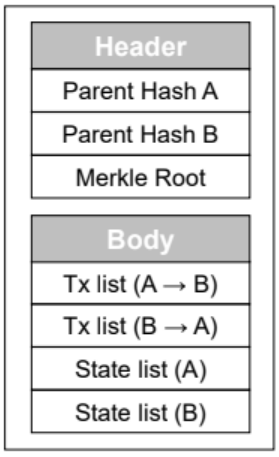


Figure 2.18. Bloco Cross-Shard dos Shards A e B [Hong et al. 2021]

Layered Sharding Consensus

Este consenso se dá em três etapas:

1. **Pré-preparação de bloco:** Nesta etapa o líder propõe o bloco de transação entre shard, e os nós chegam a um consenso através de um protocolo BFT chamado CoSi, que é um protocolo de assinatura coletiva que pode escalar eficientemente. Sendo assim um bloco de transações entre partições com uma assinatura coletiva de uma mais de dois terços de nós atesta que o b-*shard* concorda com isso em um ambiente bizantino;

2. **Preparação de bloco:** Após receber o bloco junto com a prova coletiva do b-*shard*, os nós em cada i-*shard* podem verificar a assinatura coletiva utilizando as chaves públicas salvas na *Blockchain* identidade. Então o i-*shard* também utilizará o protocolo de assinatura coletiva para chegar ao consenso e então enviar uma mensagem de Accept, junto com a hash do Header e a assinatura coletiva para o b-*shard*, caso contrário é enviada uma mensagem de Reject;

3. **commit do Bloco:** Na terceira e última etapa, os nós no b-*shard* inicializam um contador com o número de i-*shards* relacionados. E quando um nó recebe uma mensagem de Accept do i-*shard* associado ele decrementa seu contador e repassa a mensagem para os outros nós do b-*shard*. Quando o contador chega a 0, os nós no b-*shard* podem garantir que o bloco será confirmado. Mas para garantir a confirmação do bloco, uma assinatura coletiva adicional é feita, somente após isso o bloco será confirmado e uma mensagem de commit será enviada aos i-*shards*.

2.6.1.5. Epoch reconfiguration

O Pyramid utiliza reconfiguração total, ou seja, a cada epoch, todos os nós da rede serão designados para uma nova partição aleatoriamente, como apontado na seção Identity establishment and committee formation.

2.6.2. Lightchain

O LightChain [Hassanzadeh-Nazarabadi et al. 2019] replica a ideia de partição usando uma tabela hash distribuída de blocos, transações e nós. Esse modelo fragmenta as três dimensões previstas na partição completa, armazenamento, processamento e comunicação. Toda a cadeia é armazenada de forma distribuída e uniforme entre as instâncias de execução. Além disso, todo bloco ou transação é endereçável dentro da rede, e desta forma qualquer nó da rede pode solicitar um dos blocos ou transações. Na rede um nó que conheça apenas um bloco pode solicitar recursivamente pelo bloco antecessor até alcançar o bloco genesis. Da mesma forma, o LightChain permite solicitar o sucessor até atingir o bloco mais recente da cadeia.

A fragmentação do processamento se dá porque apenas alguns nós aleatórios são acionados para participar de cada processo de consenso ou consulta. Os nós, em momento algum, participam todos ao mesmo tempo de uma mesma operação.

Já a fragmentação da comunicação é consequência da estrutura, uma vez que cada nó possui somente o endereço de seus vizinhos. Entretanto, no LightChain, qualquer nó pode solicitar o estado ou saldo atual de qualquer outro nó da rede, de forma autenticada

e verificável, com complexidade de comunicação de $O(\log n)$. Isso é possível pois um nó não precisa percorrer a *Blockchain* para conseguir essas informações.

O protocolo de consenso proposto, Proof-of-Validation(PoV), possui a mesma fairness do Acordo byzantino. Fairness é a chance de um nó participar do consenso do bloco. Por outro lado, comparado ao acordo byzantino, PoV possui uma complexidade de comunicação bem reduzida. Neste protocolo, a instância que deseja adicionar um bloco na rede recebe uma lista de nós que participaram da decisão. A formação do grupo é feita de forma determinística a partir do estado da rede. Porém não é possível manipular essa escolha.

Apesar do artigo original não fazer menção expressa ao conceito de *shard*, pode-se verificar uma congruência com este conceito no consenso proposto, onde, a lista de nós que participará do consenso se assemelha a um comitê, sendo portanto, uma espécie de comitê ad hoc para cada validação de transação ou formação de bloco.

A Figura 2.19 ilustra a estrutura do LightChain. Nota-se que a cadeia, estrutura base de *Blockchain* se mantém. Porém cada instância armazena uma pequena parte dela, 1 a 2 blocos neste caso, reduzindo muito a necessidade de memória.

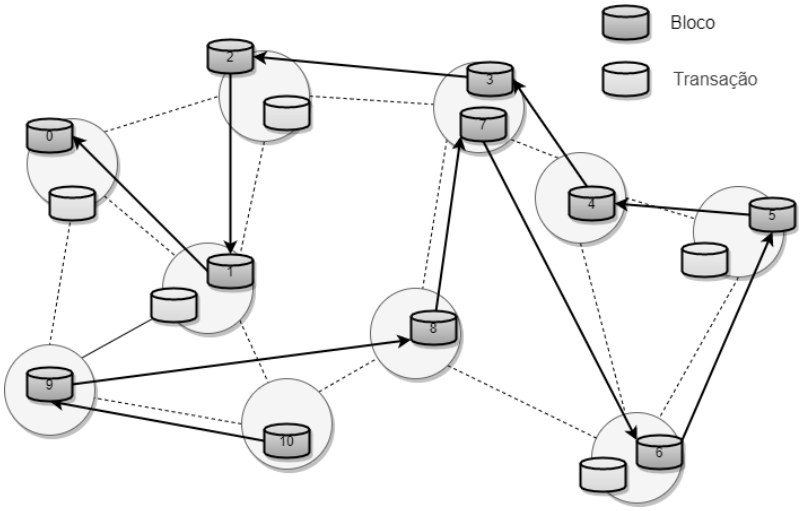


Figure 2.19. LightChain [Hassanzadeh-Nazarabadi et al. 2019]

2.6.3. Tabela Hash Distribuída

A DHT vem para resolver um problema fundamental em aplicações P2P, que é localizar de forma eficiente qual peer possui determinado dado. Consiste de uma tabela onde as informações de localização do objeto de dados são colocadas deterministicamente, nos peers com identificadores correspondentes à chave do objeto de dados. Os nós da rede têm associados a si uma sequência de bits chamado de identificador, e os dados recebem uma sequência chamada chave, oriunda do mesmo conjunto que os identificadores.

2.6.4. Segurança

Normalmente um modelo de partição precisa garantir que: Um bloco seja confirmado ou recusado, ou seja, não fique bloqueado indefinidamente por ação de invasores (vivaci-

dade). E que a distribuição das partições não é manipulável (safety). Porém isso não é necessário para esse modelo pelos motivos a seguir:

Safety: Não existe limite rígido para fração de nós maliciosos. Conceitualmente há segurança se houver ao menos um nó honesto no sistema.

Liveness: Há vivacidade enquanto houver mais de 50% de nós honestos no sistema.

Por fim é necessário provar que é improvável que um invasor consiga aprovar um bloco invalido. Para isso ele precisaria encontrar uma fração de invasores maior que $1/3$ no conjunto de nós do consenso em cada tentativa. Como não há limite para o tamanho do grupo e quanto maior o grupo menor a probabilidade um bloco invalido ser aprovado. É possível determinar um tamanho mínimo para que a chance aprovar tal bloco seja menor que $2^{-\lambda}$.

2.7. Conclusoes, Desafios de Pesquisa em Aberto e Referências Bibliográficas

Nesta seção iremos revisitar as propostas discutidas até então, procurando explicitar as diferenças entre elas observadas por meio das tabelas 2.1 e 2.2. Nesta comparação focamos em quatro aspectos: Protocol Settings, Intra-Committee Consensus, Inter-Committee Consensus e Safety and Performance [Wang et al. 2019].

2.7.1. Protocol Settings

- **formação do comitê**: se refere aos critérios usados para permitir que os nós se juntem ao comitê, que descreve os mecanismos para estabelecer a associação, por exemplo, associação com base em PoW ou PoS. Este é um aspecto importante dos sistemas descentralizados e não permissionados para impedir ataques Sybil. No entanto, para blockchain permissionadas, não precisamos lidar com ataques Sybil, uma vez que os sistemas permissionados operam em um ambiente relativamente confiável, onde os nós participantes são membros do comitê com base nesta política organizacional.

- **consistência**: A consistência mostra a probabilidade de que o sistema chegue a um consenso sobre o valor proposto, ela pode ser classificada como forte ou fraca. Em geral, os protocolos BFT clássicos oferecem consistência forte, mas estão sujeitos ao problema de escalabilidade.

- **modelo de rede**: O modelo de rede mostra a sincronia da rede de comunicação subjacente. Normalmente, as redes de comunicação podem ser categorizadas em três tipos: fortemente síncronas, parcialmente síncronas e assíncronas

2.7.2. Intra-Committee Consensus

- **configuração do comitê**: A configuração do comitê representa como os membros do comitê são atribuídos ao comitê em uma configuração de comitê único, por exemplo, ou os membros servem no comitê permanentemente (estático) ou são alterados em intervalos regulares (rotativos ou de troca) para os protocolos baseados em época.

- **incentivos**: Os incentivos mostram os mecanismos que mantêm os nós participantes motivados a participar do processo de consenso e seguir suas regras. Distinguimos os incentivos em dois aspectos: um é o processo de adesão e o outro é o processo de participação.
- **Líder**: Indica, dentro de um comitê específico, de onde vem o líder. Ele pode ser eleito entre o comitê atual (internamente), externamente ou flexível (por exemplo, por meio dos contratos inteligentes especificados).
- **Complexidade**: A complexidade mostra a complexidade da comunicação dentro de um comitê no nível da mensagem, onde não se refere ao número de nós participantes.

2.7.3. Inter-Committee Consensus

- **configuração inter-committee**: A configuração entre comitês mostra como os membros são designados aos comitês em um ambiente de comitês múltiplos, que pode ser estático ou dinâmico. Uma abordagem dinâmica é normalmente baseada na aleatoriedade gerada na época anterior.
- **Mediado**: indica como mediar as transações cross-shard. Pode ser opcionalmente mediado por um recurso externo, por exemplo, o cliente.
- **Incentivo**: indicam, para os mediadores, se eles receberão alguma recompensa por seus esforços de mediação.

2.7.4. Safety and Performance

Safety

- **TX Censorship Resistance**: O TX Censorship Resistance mostra a resiliência do sistema às transações propostas sendo suprimidas (ou seja, censuradas) por nós maliciosos envolvidos no processo de consenso;
- **resistência DoS**: Representa a resiliência dos nós envolvidos no consenso para ataques de negação de serviço (DoS). Se os participantes do protocolo de consenso forem conhecidos com antecedência, um adversário pode lançar um ataque DoS contra eles.
- **Modelo de Adversário**: Representa a fração de nós maliciosos ou defeituosos que o protocolo de consenso pode tolerar (por exemplo, o protocolo ainda funciona corretamente, apesar da presença de tais nós). Observe que, para diferentes modelos de adversário, ele pode ter diferentes taxas de resistência.

Performance

- **Throughput**: A taxa de transferência é a taxa máxima na qual as transações podem ser acordadas pelo protocolo de consenso;

- **Latência**: Representa o tempo que leva desde o momento em que uma transação é proposta até que se chegue a um consenso sobre ela.
- **Escalabilidade**: Mostra se o sistema tem a capacidade de atingir um maior rendimento quando o consenso envolve um número maior de nós.

Nas tabelas 2.1 e 2.2 considere:

- ✓: possui a propriedade;
- ✗: não possui a propriedade;
- *: possui parcialmente;
- !: não consta a informação;
- -: não se aplica à proposta.

Table 2.1. Tabela Comparativa Elastico, ChainSpace, OmniLedger e RapidChain

		Elastico	**ChainSpace**	**OmniLedger**	**RapidChain**
Protocol Settings	Formação do Comitê	PoW	Flexible	PoW/PoX	Offline PoW
	Consistência	Forte	Forte	Forte	Forte
	Modelo de Rede	Partial Sync.	Async	Partial Sync.	Sync.
Intra-Committee Consensus	Configuração do Comitê	Full Swap	Flexible	Rolling (Subset)	Partical Swap
	Incentivos (Join,Participate)	(✓, ✗)	(✗, ✗)	(✓, ✗)	(✓, ✗)
	Líder	Internal	Internal	Internal	Internal
	Complexidade	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$
Inter-Committee Consensus	Configuração Inter-Committee	Dynamic (Random)	✗	Dynamic (Random)	Dynamic (Random)
	Mediado	!	✗	Client	✗
	Incentivo	!	✗	✗	✗
Safety	TX Censorship Resistance	✗	✓	✓	✓
	Resistência DoS	✓	✓*	✓	✓
	Modelo de Adversário	33%	33%	33%	33%
Performance	Throughput	16 block in $110s$ (1)	350 tx/s (2)	$\approx 10k\ tx/s$ (3)	$\approx$ 7.300 tx/s (4)
	Latência	110s for 16 blocks	<1s	$\approx 1s$	8.7s for 7300tx
	Escalabilidade	✓	✓	✓	✓

(1): 100 nós por *shard* e 16 *shards* no total; (2): 4 nós por *shard* e 15 *shards* no total; (3): 72 nós por *shard*(12.5% adversários) e 25 *shards* no total; (4): 250 nós por *shard* e 4000 nós no total.

2.7.5. Desafios de Pesquisa em Aberto

Para melhor compreensão dos desafios de pesquisa em aberto na área de Blockchain, iremos dividir tais desafios em dois tópicos, Performance e Segurança, como proposto em [Yu et al. 2020] e [Huang et al. 2021].

Table 2.2. Tabela Comparativa CycLedger, Sharper, Pyramid e LightChain

		CycLedger	SharPer	Pyramid	LightChain
Protocol Settings	Formação do Comitê	Cryptographic Sortition	Geographical Distance	PoW	-
	Consistência	Forte	Forte	Forte	Forte
	Modelo de Rede	Sync.	Async.	Partial Sync.	Sync
Intra-Committee Consensus	Configuração do Comitê	Full Swap	Static	Full Swap	-
	Incentivos (Join,Participate)	(✓, ✓)	(-, -)	(✗, ✗)	(-, ✓)
	Líder	External	Internal	Internal	Internal
	Complexidade	$O(n)$	!	!	O(log n)
Inter-Committee Consensus	Configuração Inter-Committee	Dynamic (Random)	Static	Dynamic (Random)	-
	Mediado	✗	Client	✗	-
	Incentivo	✗	✗	✗	-
Safety	TX Censorship Resistance	✓	-/✓	✓	✓
	Resistência DoS	✓*	-/✓	✓	✓
	Modelo de Adversário	33%	50%/33%	33%	33%
Performance	Throughput	!	$\approx 27k\ tx/s$ (5)	$\approx 12k\ tx/s$ (6)	!
	Latência	!	240ms for 27000tx	$\approx 5s$	!
	Escalabilidade	✓	✓	✓	✓

(5):20 nodes, 10% cross-shard transactions ; (6): 20 shards, 4000 nós(12.5% adversários), w(parâmetro de escalabilidade) = 0.5.
Na coluna da proposta SharPer, temos separado por "/" os valores considerando Crash Only Nodes e Byzantine Nodes.

2.7.5.1. Performance

Escalabilidade: A escalabilidade ainda é um grande desafio para a maioria dos sistemas de *Blockchain*. Por exemplo, os protocolos de consenso PBFT emitem um número $O(n^2)$ de mensagens, onde n é o número de participantes. O grande número de mensagens torna a escalabilidade irreal.

Mecanismos Resilientes para Sharding: Os mecanismos resilientes para fragmentar blockchains ainda estão faltando [Huang et al. 2021].

Performace de Transações Cross-Shard: Embora vários protocolos de fragmentação tenham sido propostos, esses protocolos só podem suportar no máximo 1/3 de adversários. Assim, protocolos de acordo bizantino mais robustos precisam ser elaborados. Além disso, todos os protocolos baseados em sharding incorrem em latências e tráfegos cross-shard adicionais devido às transações entre shard. Portanto, o desempenho do cross-shard em termos de taxa de transferência, latência e outras métricas deve ser bem garantido em estudos futuros.

Transações Cross-Chain: Este tópico diz respeito a interoperabilidade entre *Blockchains*, prevendo que, no futuro, teremos uma grande variedade de sistemas *Blockchain* e eles precisarão se comunicar entre si, tal tópico foi pouco explorado e teve o pontapé inicial dado por [Jin et al. 2018].

Soluções de aceleração assistidas por hardware para redes *Blockchain*: Para melhorar o desempenho de *blockchains*, por exemplo, para reduzir a latência de confirmação de transação, algumas tecnologias de rede avançadas, como *Remote Direct Memory Access* (RDMA) e placas de rede de alta velocidade, podem ser exploradas para acelerar o acesso a dados entre as mineradoras em redes *blockchain*.

Otimização de desempenho em diferentes camadas de rede Blockchain: A rede *blockchain* é construída sobre as redes P2P, que incluem várias camadas típicas, como camada mac, camada de roteamento, camada de rede e camada de aplicativo. Os protocolos baseados em BFT funcionam essencialmente para a camada de rede. Portanto, melhorias de desempenho podem ser alcançadas ao propor vários protocolos, algoritmos e modelos teóricos para outras camadas da rede *blockchain*.

Redes de *Big Data* assistidas por *Blockchain*: Embora big data e *blockchain* tenham várias métricas de desempenho que são contrárias entre si. Por exemplo, *big data* é uma tecnologia de gerenciamento centralizado com ênfase na preservação da privacidade orientada para diversos ambientes de computação. Os dados processados pela tecnologia de *big data* devem garantir a não redundância e a arquitetura não estruturada em uma rede de computação em grande escala. Em contraste, a tecnologia *blockchain* se baseia em uma arquitetura descentralizada, transparente e imutável, na qual o tipo de dados é simples, estruturado e altamente redundante. Além disso, o desempenho de *blockchain* requer escalabilidade e o paradigma de computação fora da cadeia. Portanto, como integrar essas duas tecnologias e buscar o benefício mútuo uma da outra é uma questão em aberto que merece estudos aprofundados. Por exemplo, os tópicos de pesquisa em potencial incluem como projetar uma nova arquitetura de *blockchain* adequada para tecnologias de *big data* e como quebrar as ilhas de dados isoladas usando *blockchain*, garantindo os problemas de privacidade de *big data*.

2.7.5.2. Segurança

Preservação de privacidade em *Blockchains*: A maioria dos trabalhos existentes nessa categoria está discutindo a preservação da privacidade e a segurança que a *blockchain* fornece para as aplicações. O fato é que a segurança e a privacidade também são questões críticas da *blockchain* em si. Por exemplo, a privacidade das transações poderia ser hackeada por invasores. Embora estudos sejam escassos, é possível apontar o Monero [Alonso et al. 2020] como solução que se destaca no assunto. Fazendo uso extenso de criptografia, o Monero propõe a preservação da privacidade através do sigilo sistêmico de quatro informações: o endereço público do emissor e do receptor, o montante transferido, e o endereço IP dos envolvidos através de rede sobreposta I2P ou Tor.

Mecanismos anti-criptojacking para mineradores mal-intencionado: Mineradores maliciosos fazendo uso de códigos que confiscam os recursos de hardware, como capacidade computacional e memória, dos usuários da web para utilizá-los em seu

proveito. De acordo com [Tahir et al. 2019] tais ataques ocorrem em navegadores web, portanto, é necessário desenvolver mecanismos e estratégias anti-cryptojacking para proteger os usuários normais da web e manter a rede *blockchain* justa.

Problemas de segurança de *blockchains* de criptomoeda: Poucos esforços foram dedicados à investigações teóricas para as questões de segurança de *blockchains* de criptomoeda. Por exemplo, a exploração de punição e cooperação entre mineradores em várias cadeias é um tópico interessante para *blockchains* de criptomoeda. Portanto, é plausível o surgimento de perspectivas mais amplas de modelagem dos comportamentos de atacantes e contra-atacantes no contexto de ataques de *blockchains* monetárias.

Lei Geral de Proteção de Dados Pessoais - LGPD e *Blockchains*: Uma das principais premissas da LGPD (ou sua equivalente europeia General Data Protection Regulation - GDPR) é o controle do usuário sob seus dados, com prerrogativa de exclusão. Isto vai de encontro à imutabilidade das blockchains. Alguns estudos propõem o desacoplamento do controle e dos dados, transferindo este ultimo para um armazenamento off-chain, enquanto o primeiro permanece na blockchain. Como o armazenamento off-chain não será imutável, alcançar o mesmo nível de descentralização e segurança na parte off-chain é um desafio. [Truong et al. 2020] [Daudén-Esmel et al. 2021]

References

[Al-Bassam et al. 2017] Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., and Danezis, G. (2017). Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778*.

[Alonso et al. 2020] Alonso, K. M. et al. (2020). Zero to monero.

[Amiri et al. 2019] Amiri, M. J., Agrawal, D., and Abbadi, A. E. (2019). Sharper: Sharding permissioned blockchains over network clusters. *arXiv preprint arXiv:1910.00765*.

[Andrychowicz and Dziembowski 2015] Andrychowicz, M. and Dziembowski, S. (2015). Pow-based distributed cryptography with no trusted setup. In *Annual Cryptology Conference*, pages 379–399. Springer.

[bit2me 2020] bit2me (2020). O que é sharding?

[Boneh et al. 2018] Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. (2018). Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer.

[Bootle et al. 2016] Bootle, J., Cerulli, A., Chaidos, P., and Groth, J. (2016). Efficient zero-knowledge proof systems. In *Foundations of security analysis and design VIII*, pages 1–31. Springer.

[CasaDaMoeda 2015] CasaDaMoeda (2015). Origem do dinheiro.

[Castro et al. 1999] Castro, M., Liskov, B., et al. (1999). Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186.

[Centieiro 2021] Centieiro, H. (2021). What's proof of elapsed time.

[Corso 2019] Corso, A. (2019). *Performance analysis of proof-of-elapsed-time (poet) consensus in the sawtooth blockchain framework*. PhD thesis, University of Oregon.

[Danezis et al. 2014] Danezis, G., Fournet, C., Groth, J., and Kohlweiss, M. (2014). Square span programs with applications to succinct nizk arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 532–550. Springer.

[Daudén-Esmel et al. 2021] Daudén-Esmel, C., Castellà-Roca, J., Viejo, A., and Domingo-Ferrer, J. (2021). Lightweight blockchain-based platform for gdpr-compliant personal data management. In *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*, pages 68–73.

[Douceur 2002] Douceur, J. R. (2002). The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer.

[Eichmann 2018] Eichmann, K. (2018). The future client of an energy utility company will be a machine.

[Fernando Wosniak Steler 2017] Fernando Wosniak Steler, A. H. C. (2017). Tudo o que você queria saber sobre blockchain e tinha receio de perguntar.

[Hassanzadeh-Nazarabadi et al. 2019] Hassanzadeh-Nazarabadi, Y., Küpçü, A., and Özkasap, Ö. (2019). Lightchain: A dht-based blockchain for resource constrained environments. *arXiv preprint arXiv:1904.00375*.

[Hong et al. 2021] Hong, Z., Guo, S., Li, P., and Chen, W. (2021). Pyramid: A layered sharding blockchain system. *IEEE INFOCOM*.

[Huang et al. 2021] Huang, H., Kong, W., Zhou, S., Zheng, Z., and Guo, S. (2021). A survey of state-of-the-art on blockchains: Theories, modelings, and tools. *ACM Computing Surveys (CSUR)*, 54(2):1–42.

[IntelSGX 2021] IntelSGX (2021). Intel software guard extensions.

[Jake Frankenfield 2020] Jake Frankenfield, S. G. A. (2020). Proof of elapsed time (poet) (cryptocurrency).

[Jin et al. 2018] Jin, H., Dai, X., and Xiao, J. (2018). Towards a novel architecture for enabling interoperability amongst multiple blockchains. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1203–1211. IEEE.

[King and Nadal 2012] King, S. and Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19:1.

[Kokoris-Kogias et al. 2018] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. (2018). Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598.

[L. 2019] L., K. (2019). The blockchain scalability problem the race for visa-like transaction speed.

[Lamport et al. 2001] Lamport, L. et al. (2001). Paxos made simple. *ACM Sigact News*, 32(4):18–25.

[Luu et al. 2016] Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., and Saxena, P. (2016). A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30.

[Maymounkov and Mazieres 2002] Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer.

[Mearian 2019] Mearian, L. (2019). Sharding: What it is and why many blockchain protocols rely on it.

[Micali et al. 1999] Micali, S., Rabin, M., and Vadhan, S. (1999). Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE.

[Muratov et al. 2018] Muratov, F., Lebedev, A., Iushkevich, N., Nasrulin, B., and Takemiya, M. (2018). Yac: Bft consensus algorithm for blockchain. *arXiv preprint arXiv:1809.00554*.

[Nakamoto 2008] Nakamoto, S. (2008). Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*.

[Pahlajani et al. 2019] Pahlajani, S., Kshirsagar, A., and Pachghare, V. (2019). Survey on private blockchain consensus algorithms. In *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, pages 1–6. IEEE.

[Rebello et al. 2019] Rebello, G., Camilo, G., Silva, L., Souza, L., Guimarães, L., Alchieri, E., Greve, F., and Duarte, O. (2019). Correntes de blocos: Algoritmos de consenso e implementação na plataforma hyperledger fabric. *Sociedade Brasileira de Computação*.

[REVOREDO 2019] REVOREDO, T. (2019). Blockchain: tudo que você precisa saber (potencial e realidade).

[Syta et al. 2016] Syta, E., Jovanovic, P., Kogias, E. K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M. J., and Ford, B. (2016). Scalable bias-resistant distributed randomness. Cryptology ePrint Archive, Report 2016/1067. https://eprint.iacr.org/2016/1067.

[Tahir et al. 2019] Tahir, R., Durrani, S., Ahmed, F., Saeed, H., Zaffar, F., and Ilyas, S. (2019). The browsers strike back: Countering cryptojacking and parasitic miners on the web. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 703–711. IEEE.

[Truong et al. 2020] Truong, N. B., Sun, K., Lee, G. M., and Guo, Y. (2020). Gdpr-compliant personal data management: A blockchain-based solution. *IEEE Transactions on Information Forensics and Security*, 15:1746–1761.

[Wang et al. 2019] Wang, G., Shi, Z. J., Nixon, M., and Han, S. (2019). Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 41–61.

[Wang and Wang 2019] Wang, J. and Wang, H. (2019). Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 95–112.

[Wood et al. 2014] Wood, G. et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32.

[Yu et al. 2020] Yu, G., Wang, X., Yu, K., Ni, W., Zhang, J. A., and Liu, R. P. (2020). Survey: Sharding in blockchains. *IEEE Access*, 8:14155–14181.

[Zamani et al. 2018] Zamani, M., Movahedi, M., and Raykova, M. (2018). Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948.

[Zhang et al. 2020] Zhang, M., Li, J., Chen, Z., Chen, H., and Deng, X. (2020). Cycledger: A scalable and secure parallel protocol for distributed ledger via sharding.

Capítulo 3

Autenticando aplicações nativas da nuvem com identidades SPIFFE[1]

Eduardo Falcão (UFRN), Matteus Silva (UFCG), Clenimar Souza (Scontain), Andrey Brito (UFCG)

Abstract

The purpose of this tutorial is to present how cloud-native applications are authenticated with the zero-trust model and the SPIFFE specification. A distributed application is used as study case and its microservices are attested to receive SPIFFE identities that allow mutual authentication according to the zero-trust principles. To illustrate the advantages of zero-trust model with SPIFFE identities in cloud-native computing environments, we leverage both attestation mechanisms based in Kubernetes as well as mechanisms based in Intel SGX (using the SCONE framework).

Resumo

A proposta deste minicurso é apresentar como aplicações nativas da nuvem são autenticadas utilizando o modelo confiança-zero e o padrão SPIFFE. Uma aplicação distribuída é utilizada como estudo de caso e seus microsserviços são atestados de modo a receber identidades SPIFFE que permitirão autenticação mútua de acordo com os princípios de confiança zero. Para ilustrar as vantagens do modelo de confiança zero com identidades SPIFFE em ambientes de computação nativa da nuvem, usaremos tanto mecanismos de atestação baseados em Kubernetes, como mecanismos baseados em Intel SGX (usando o arcabouço SCONE).

3.1. Introdução

Computação na nuvem é um paradigma bastante conhecido por pessoas da área de Tecnologia da Informação (TI), e até por pessoas que não têm afinidade com a área mas usam

[1]Os códigos e scripts utilizados neste documento, assim como erratas, guias mais detalhados e atualizações podem ser encontrados no repositório do minicurso: `https://github.com/ufcg-lsd/minicurso-sbseg-2021`.

aplicativos e serviços que popularizaram o termo. Surgiu no fim do século XX, e ganhou força no início do século XXI graças à evolução de tecnologias de virtualização e ampla disponibilidade através dos provedores públicos de computação na nuvem. Do ponto de vista da operacionalização de sistemas, sua proposta geral é facilitar e otimizar a alocação de recursos gerando economia financeira e eficiência gerencial de recursos computacionais e humanos [Mell and Grance 2011].

O surgimento da computação na nuvem também impactou a Engenharia e Arquitetura de Software. A princípio, muitos sistemas de propósito geral eram desenvolvidos com arquitetura monolítica e implantados de modo centralizado em servidor único. Ainda antes do surgimento da computação na nuvem já eram construídos sistemas descentralizados, mas em uma proporção menor, pois as tecnologias e técnicas eram complexas e pouco difundidas quando comparadas às disponíveis atualmente. Entretanto, a popularização da computação na nuvem e a evolução de tecnologias de desenvolvimento e implantação facilitaram a criação de sistemas completamente distribuídos. Dentre as arquiteturas de sistemas distribuídos, o padrão arquitetural de microsserviços tem se destacado, sobretudo quando se trata de desenvolvimento de sistemas a serem hospedados na nuvem, pois compartilham alguns princípios semelhantes, como por exemplo, a facilidade e rapidez na alocação e desalocação de recursos e serviços.

Com a adoção desse padrão arquitetural surgiram alguns desafios decorrentes do gerenciamento de diferentes perspectivas de uma aplicação baseada em microsserviços. É evidente que gerenciar a implantação, comunicação, alocação de recursos, e segurança de uma grande quantidade de microsserviços torna-se mais complexo em comparação com aplicações monolíticas. Essa dificuldade levou a comunidade a juntar esforços para a criação de um conjunto de técnicas e ferramentas que viabilizassem a operação de microsserviços: a **computação nativa da nuvem**.

Uma vez que a computação nativa da nuvem facilita a distribuição dos serviços, do ponto de vista de segurança, abordagens tradicionais de proteção baseadas em perímetro têm sido depreciadas pois com vários microsserviços implantados em diferentes plataformas de nuvem, privadas e públicas, não existe mais um perímetro claramente estabelecido como havia antes. A falta de um perímetro claro aumenta a superfície de ataque, problemática esta que aumentou a relevância do **modelo confiança-zero**.

Finalmente, ataques cibernéticos cada vez mais sofisticados têm causado continuamente grandes vazamento de dados em organizações de todo o porte. Por esta razão, além das conhecidas estratégias para proteção de dados em repouso (criptografia) e em trânsito (protocolos de comunicação seguros), a comunidade está discutindo formas de proteger os dados durante seu processamento. Esta última técnica é chamada de **computação confidencial**.

Este trabalho descreve a importância desses três pilares para a criação de aplicações distribuídas que podem ser consideradas seguras diante de um modelo de ameaça rigoroso, no qual até mesmo o provedor de nuvem não é considerado confiável. Em outro trabalho [Brito et al. 2020], nós já descrevemos de forma aprofundada conceitos e práticas relacionados à computação nativa da nuvem e à computação confidencial. Neste trabalho nós revisamos esses conceitos essenciais e focamos em como aplicações nativas da nuvem executando sobre diferentes plataformas e tecnologias podem ser autenticadas

com identidades interoperáveis baseadas no padrão SPIFFE.

Este trabalho está estruturado da seguinte forma. A próxima seção apresenta os conceitos de microsserviços e computação nativa da nuvem, provendo informações básicas sobre a ferramenta mais popular para orquestração de contêineres, o Kubernetes. A seção 3.3 explica o que é o modelo confiança-zero e introduz conceitos básicos sobre identidades de software. Além disso, são apresentados o padrão SPIFFE, a ferramenta SPIRE, e uma demonstração onde serviços são autenticados com identidades SPIFFE. Os conceitos de computação confidencial e o arcabouço SCONE são apresentados na seção 3.4. A seção 3.5 apresenta como estudo de caso uma aplicação distribuída que autentica serviços não-confidenciais e confidenciais com identidades SPIFFE. A seção 3.6 encerra este trabalho apontando os principais desafios e direções de pesquisa.

3.2. Computação nativa da nuvem

A computação nativa da nuvem[2] (ou CNC, do inglês, *cloud-native computing*) é um conjunto de técnicas que permitem a implementação e a implantação de aplicações escaláveis em ambientes dinâmicos, como nuvens privadas, públicas ou híbridas. Aplicações nativas da nuvem apresentam baixo acoplamento, são resilientes, gerenciáveis e observáveis, tendo os estágios do seu ciclo de vida automatizados e integrados sempre que possível. Por estes motivos, estão bem alinhadas à arquitetura de microsserviços, cuja principal forma de distribuição é através de contêineres.

O conceito de computação nativa da nuvem ganhou força com a fundação da *Cloud Native Computing Foundation* (CNCF), em 2015. A CNCF tem como objetivo divulgar e dar suporte à adoção do paradigma de computação nativa da nuvem, e o faz através da organização de eventos e da manutenção de um ecossistema de projetos de código aberto e neutros em relação a fornecedores (*vendor-neutral*). Os projetos apoiados pela CNCF[3] podem ter os seguintes níveis de maturidade: iniciantes, em incubação, ou graduados. Alguns arcabouços bastante conhecidos são projetos considerados graduados pelo CNCF, ou seja, projetos que possuem ampla adoção pela comunidade e em ambientes de produção. Dentre os projetos graduados podem ser mencionados o containerd (execução de contêineres), o Kubernetes (orquestração de contêineres), o Helm (gerenciador de pacotes para Kubernetes), e o Prometheus (monitoramento de recursos). Alguns projetos que estão em incubação pelo CNCF são o gRPC (comunicação via chamadas de procedimento remoto), o SPIFFE (especificação para criação de identidades) e o SPIRE (implementação de referência do SPIFFE).

Os projetos acompanhados pela CNCF são excelentes ferramentas para a construção e operação de aplicações nativas da nuvem, por geralmente promoverem uma utilização eficiente dos recursos da nuvem. Além disso, a utilização de interfaces declarativas e genéricas, que podem ser manipuladas ou integradas por meio de processos automatizados e monitoráveis, faz com que as soluções construídas a partir destes projetos sejam portáveis para diversas infraestruturas e provedores. Neste minicurso construiremos uma

[2]A definição oficial de computação nativa da nuvem pode ser encontrada em `https://github.com/cncf/toc/blob/main/DEFINITION.md`.

[3]A lista dos projetos acompanhados pela CNCF pode ser encontrada em `https://www.cncf.io/projects/`.

aplicação nativa da nuvem que será implantada e executada em contêineres. A aplicação usará Kubernetes para orquestração dos contêineres, e Kafka para comunicação. Finalmente, a aplicação usará identidades SPIFFE para autenticação, e tecnologias de computação confidencial para processamento de dados sensíveis.

3.2.1. Microsserviços

Usar provedores de nuvem para hospedar aplicações e serviços não é novidade. Contudo, nem todas as aplicações conseguem aproveitar a dinamicidade e a elasticidade que provedores de nuvem proporcionam. Aplicações de arquitetura monolítica, por exemplo, bastante comuns antes do surgimento e popularização dos provedores de nuvem, geralmente não permitem uma alocação de recursos mais eficiente e alinhada com as necessidades reais da aplicação. Isto acontece porque a lógica de negócio implementada pela aplicação geralmente tem pontos críticos específicos e que requerem mais recursos do que outros, mas o monólito não oferece a granularidade necessária para alocar mais recursos somente aos pontos críticos, o que resulta em mais custos.

A arquitetura de microsserviços, cuja ideia principal é separar cada ponto específico da lógica de negócio em serviços distintos e com interfaces bem definidas, permite explorar melhor a elasticidade oferecida por provedores de nuvem, já que a aplicação é agora composta por uma série de aplicações menores que se comunicam entre si. Deste modo, a necessidade de alocar mais recursos à aplicação pode ser suprida de forma mais eficiente, já que naturalmente alguns serviços precisam de mais recursos que outros. Além disso, a utilização de microsserviços tende a acelerar o processo de desenvolvimento, onde times menores e mais especializados podem entregar valor mais rapidamente.

Por definição, um microsserviço é um processo coeso, independente, que interage com outros serviços através de mensagens [Dragoni et al. 2017]. Aplicações baseadas em microsserviços, quando bem projetadas, oferecem uma série de benefícios relacionados. Cada microsserviço deve ser configurado com suas dependências de forma autônoma. Isto promove isolamento, fraco acoplamento, e facilita o processo de construção e lançamento de novas versões em produção. Outro benefício do fraco acoplamento é a possibilidade de evolução de um microsserviço, isoladamente, para incorporar novas tecnologias, o que seria impossível para aplicações de arquitetura monolítica.

Idealmente, iniciar e terminar microsserviços deveria ser tão simples e fácil de modo que eles sejam considerados descartáveis. Um fator que contribui para esta característica é o não armazenamento de estado (*stateless*), pois facilita a adequação dos recursos à carga experimentada através da criação e término de instâncias de processamento (escalonamento horizontal). Isto também ajuda na recuperação contra falhas de *software* e *hardware*, pois basta reiniciar os microsserviços com falhas de sistema, ou iniciar uma nova instância do microsserviço em outro servidor que não apresente problemas. Estas e outras recomendações podem ser encontradas em artigos científicos [Khan 2017, Hoffman 2016] ou de forma resumida no manifesto "The Twelve-Factor App" [Wiggins 2017], criado pelos desenvolvedores do Heroku[4].

Aplicações baseadas em microsserviços incorporam os benefícios supramencio-

[4]`https://www.heroku.com/`

nados a um custo de complexidade decorrente da heterogeneidade de tecnologia e plataforma empregadas. Microsserviços coesos são mais fáceis de serem implementados, mas a operação e manutenção de centenas ou até milhares deles não é uma tarefa trivial. A comunicação também pode se tornar difícil de gerenciar. Microsserviços heterogêneos podem se comunicar por diferentes protocolos de troca de mensagens, de forma síncrona ou assíncrona. APIs REST[5] é um exemplo de abordagem síncrona que mitiga a heterogeneidade de comunicação entre microsserviços. O aspecto negativo é que serviços produtores e consumidores ficam acoplados, dificultando a implementação de estratégias de balanceamento de carga, escalonamento de recursos, e recuperação de falhas.

Abordagens assíncronas como, por exemplo, os sistemas publicar-assinar e sistemas de filas de mensagens [Eugster et al. 2003, Dobbelaere and Esmaili 2017], desacoplam produtor de consumidor. Nessas abordagens, mensagens enviadas pelos serviços são agrupadas em *brokers* de mensagens tais como barramentos ou filas. Os serviços interessados (assinantes) podem receber as mensagens seguindo dois modelos de notificação: *pull* e *push*. No modelo *push*, sempre que o *broker* receber uma mensagem ele a encaminhará para os assinantes, enquanto no modelo *pull* os assinantes são notificados de atualizações mas decidem o momento de verificar se o conteúdo novo lhes interessam. Dois exemplos populares de ferramentas de publicar-assinar são o Apache Kafka[6] e o RabbitMQ[7].

Com as mensagens armazenadas nos *brokers*, diferentes políticas de balanceamento de carga podem ser aplicadas, a depender da ferramenta utilizada. No Kafka, por exemplo, a carga pode ser balanceada configurando apropriadamente tópicos e partições. Uma forma simples de lidar com a sobrecarga em um *broker* é monitorar o acúmulo de mensagens e adicionar novas instâncias do microsserviço em momentos de pico. Já para recuperação de falhas, um *broker* assíncrono armazena mensagens por períodos configuráveis (guardando dos últimos minutos ou até últimos dias) e pode reenviá-las para ajudar na restauração de um componente.

Escalabilidade e tolerância a falhas são propriedades que podem ser asseguradas através de diferentes abordagens complementares. No contexto de aplicações CNC, o Kubernetes é uma ferramenta que monitora os estados de contêineres (microsserviços) para decidir se deveria criar ou terminar contêineres para fins de escalabilidade ou recuperação de falhas. A próxima seção apresenta essa ferramenta e seus principais conceitos.

3.2.2. Kubernetes

Kubernetes é uma ferramenta de orquestração de contêineres originalmente desenvolvida pela Google, e fortemente influenciada por sistemas internos de gerenciamento de contêineres da empresa, como o Borg [Verma et al. 2015]. Seu lançamento coincide com um período de crescimento na adoção de contêineres como forma de empacotar, distribuir e implantar aplicações, o que fez surgir a necessidade de soluções para gerenciamento de contêineres em larga escala.

[5] API é abreviação para *Application Programming Interface*, e REST é abreviação para *Representational State Transfer*.

[6] `https://kafka.apache.org`

[7] `https://www.rabbitmq.com`

Kubernetes adota um modelo declarativo, que é muito comum no paradigma de computação nativa da nuvem, onde se definem fatos, ou o estado esperado de partes do sistema, em vez de ações, como é no paradigma imperativo. Os nós de controle de um *cluster* Kubernetes possuem um conjunto de controladores que constantemente observam o estado atual do sistema e, se necessário, atuam sobre ele para atingir o estado esperado. Assim, é possível delegar aos controladores o gerenciamento de aspectos como escalonamento, tolerância a falhas, replicação, comunicação em rede, descoberta de serviços (*service discovery*) e gerenciamento de infraestrutura e recursos, permitindo um foco maior na aplicação em si e em sua evolução. Outro ponto que dá grande flexibilidade às APIs de Kubernetes é o sistema de etiquetas e seletores, que permite atribuir etiquetas a praticamente qualquer tipo de recurso, e implementar filtros, ou seletores, para que um conjunto de fatos seja aplicado somente a um subconjunto dos recursos existentes.

Um *cluster* Kubernetes é composto por dois tipos de nós: os nós de controle, também conhecidos por nós mestres, que abrigam o servidor de API do *cluster* e diversos controladores; e os nós trabalhadores, que de fato executam as cargas e aplicações submetidas pelos usuários. Nós trabalhadores também executam o agente de nó do Kubernetes, ou *kubelet*, que é responsável por reportar o estado do nó para os controladores dos nós de controle. O estado do *cluster* é armazenado no banco de dados chave-valor etcd, também localizado nos nós de controle. Para configurações de alta disponibilidade, é recomendado que se tenha ao menos 3 nós de controle. No presente momento, Kubernetes suporta oficialmente até *5000* nós [8] por *cluster*, o que ilustra a capacidade de gerenciar infraestruturas muito grandes.

3.2.2.1. Tipos de Recurso

Qualquer aplicação submetida a um *cluster* Kubernetes é definida em um arquivo, ou manifesto, escrito na linguagem YAML [9]. Por padrão, a API do *cluster* oferece tipos ou recursos primitivos (*resource types*) que oferecem garantias e comportamentos diferentes. Recursos são sempre criados no contexto de um *namespace*. O *namespace* padrão é chamado de *default*, e serviços de controle do *cluster* rodam no *namespace* "kube-system". Destes tipos oferecidos, o mais básico é o *pod*, que representa um conjunto de um ou mais contêineres que funcionam como uma unidade lógica. Por exemplo, um servidor de banco de dados e um *proxy* reverso: embora sejam dois contêineres distintos, eles podem ser vistos como uma unidade lógica de aplicação. Contêineres no mesmo *pod* compartilham o espaço de rede local e podem compartilhar pedaços de sistema de arquivo definidos em volumes. É importante notar que *pods* são considerados efêmeros e *stateless*, podendo ser criados, movidos ou removidos livremente pelos controladores do *cluster* a fim de atingir o estado esperado do sistema. *Pods* não oferecem qualquer garantia de replicação ou tolerância a falha e, por este motivo, devem estar sempre associados a algum controlador de aplicação. Existem quatro tipos básicos de controladores de aplicação, e cada um possui um comportamento específico, como descrito na Tabela 3.1.

Kubernetes também oferece tipos primitivos para possibilitar a comunicação entre

[8] https://kubernetes.io/docs/setup/best-practices/cluster-large/
[9] https://yaml.org

Nome do recurso	Descrição
Deployment	Um controlador de aplicação que assegura uma quantidade determinada de réplicas para um *pod*. Caso o número de réplicas atual não seja o especificado, o controlador cria ou remove réplicas para atingir o estado esperado. É possível modificar o número desejado de réplicas a qualquer momento. Indicado para aplicações *stateless* que executam indefinidamente.
DaemonSet	Um controlador de aplicação que assegura que uma e somente uma réplica do *pod* será executada em cada nó do *cluster*. É possível filtrar nós através de etiquetas e seletores. Indicado para implementar aplicações como agentes de armazenamento, monitoramento e atestação.
StatefulSet	Um controlador de aplicação que assegura uma quantidade determinada de réplicas para um *pod*, mas também lhes atribui uma identidade permanente, o que habilita aplicações *stateful*. O estado em si é guardado em volumes persistentes (*PersistentVolumes*) que sempre é associado à identidade permanente. Volumes persistentes suportam diversos provedores de armazenamento (como NFS, Ceph, ou serviços específicos de provedores de nuvem, como o Azure File). *StatefulSets* são indicados para aplicações que executam indefinidamente e que possuem estado, como um servidor de banco de dados, por exemplo.
Job	Um controlador de aplicação para *pods* que executam até a completude (*run-to-completion*). Indicado para aplicações em lotes (*batch*), como tarefas de análise de dados ou de aprendizado de máquina. É possível definir execuções paralelas e políticas de tolerância a falha. Apesar de simples, este recurso pode ser complementado por outros sistemas para a construção de *batch schedulers* mais complexos [Sampaio et al. 2019].

Tabela 3.1. Controladores de aplicação básicos de um *cluster* Kubernetes.

pods. Cada *pod* possui um endereço IP único na rede interna do *cluster*. Entretanto, por serem tratados como recursos efêmeros, não se deve utilizar os endereços IP dos *pods* para comunicação entre serviços. A solução está no tipo primitivo *Service*, que expõe um determinado conjunto de *pods* e, com isso, oferece um endereço estável para comunicação via rede. Na prática, *Services* funcionam como balanceadores de carga para o conjunto de *pods* que eles expõem, e as requisições são alternadas entre os *pods* disponíveis. Além disso, cada *Service* é registrado no servidor de DNS interno do *cluster*, o que permite que

outros *pods* consigam acessá-los através de seu nome.

Services têm tipos distintos. O tipo padrão, *ClusterIP*, expõe o conjunto de *pods* apenas na rede interna do *cluster*, isto é, apenas para outros *pods* no mesmo *cluster*. Os tipos *LoadBalancer* e *NodePort*, por sua vez, além de possibilitarem a comunicação intra-*cluster*, também expõem o conjunto de *pods* para tráfego externo. A diferença entre esses tipos vive na forma como o conjunto de *pods* é exposto: no tipo *LoadBalancer*, o controlador do *cluster* especializado em APIs de provedores de nuvem (*Cloud Controller Manager*, ou CCM) provisiona um balanceador de carga gerenciado pelo provedor para expor o conjunto de *pods*; no tipo *NodePort*, por sua vez, os controladores de nó abrem uma porta específica, chamada de *nodePort*, em todos os nós do *cluster*. A porta é escolhida aleatoriamente dentro de um intervalo predefinido, chamado de intervalo de *nodePort*, que compreende as portas de 30000 até 32767. O tráfego pode ser direcionado para o endereço IP do nó na *nodePort* escolhida. Não é necessário utilizar o endereço IP de nós que hospedem os *pods*-alvo, pois o tráfego é redirecionado automaticamente para o *pod* adequado.

Outra forma de expor um conjunto de *pods* para o mundo exterior é através do tipo *Ingress*, que também funciona como um balanceador de carga, mas a nível de aplicação, que expõe *Services* (que atuam na camada de transporte) para o mundo exterior. A vantagem do *Ingress* é a sua flexibilidade, já que é possível implementar comportamentos complexos, como *traffic splitting*, *circuit breakers*, *rate limiters*, *canary deployments*, *virtual host routing*, entre outros. O tipo *Ingress*, contudo, não possui um controlador específico por padrão. Cabe ao usuário escolher e instalar um controlador de *Ingress*. Das diversas opções disponíveis na comunidade, destacam-se os controladores NGINX Ingress Controller (baseado em NGINX), *Voyager* e *HAProxy Ingress Controller* (baseados em HAProxy), *Ambassador* e *Contour* (baseados em Envoy Proxy).

Kubernetes também oferece tipos de recursos específicos para a configuração de aplicações. O tipo *ConfigMap* define um conjunto de arquivos que podem ser referenciados dinamicamente por um ou mais *pods* via sistema de arquivos (volumes) ou variáveis de ambiente. O tipo *Secret* também define um conjunto de arquivos, porém com seu conteúdo codificado pelo método Base64, destinado para a arquivos ou dados sensíveis, como credenciais ou senhas. Vale salientar que, embora seu conteúdo não seja de imediato decifrável, a decodificação de Base64 para texto plano é trivial, logo o tipo *Secret* não fornece nenhuma real garantia de confidencialidade ou integridade.

Por padrão, todos os *pods* do *cluster* possuem uma conta de serviço padrão (descrita pelo tipo primitivo *ServiceAccount*), com permissões limitadas. Ao utilizar uma conta de serviço, cada *pod* recebe um *token* de autorização através de um *Secret* montado automaticamente pelo *pod*, chamado de *Service Account Token* (SAT), que pode ser utilizado para se autenticar com outros serviços do *cluster*. Também é possível utilizar *Projected Service Accounts*, num fluxo bastante similar, mas que permite associar parâmetros adicionais ao *Projected Service Account Token* (PSAT), como tempo de vida e metadados. Desta forma, componentes do *cluster* que recebem estes *tokens* podem verificar a sua validade através de uma API especial do *cluster* chamada de *TokenReview API*. Caso os *pods* precisem de mais permissões, como por exemplo para criar recursos no próprio *cluster* através da API do Kubernetes, é necessário utilizar uma conta de serviço

com as permissões adequadas atribuídas no sistema de autorização baseada em papéis (do inglês, *Role-Based Access Control*, ou RBAC).

Uma vez compreendidas a importância do padrão arquitetural de microsserviços, e da ferramenta de orquestração de contêineres, Kubernetes, na construção de aplicações CNC, o próximo passo é entender o modelo confiança-zero e como o padrão SPIFFE facilita a construção de aplicações CNC em conformidade com esse modelo.

3.3. Confiança Zero

Uma consequência de se segmentar uma grande aplicação em vários microsserviços é que a superfície de ataque aumenta substancialmente. Uma aplicação monolítica é comumente hospedada em um único servidor, talvez suportado por serviços como um banco de dados em separado, mas ainda assim a gestão é simples. Desses serviços, apenas uma pequena fração é exposta ao mundo e, portanto, configurar políticas de segurança para gerenciar a exposição de um número reduzido de serviços é mais simples. Esta abordagem de proteção baseada no estabelecimento de um perímetro seguro para a rede cria a sensação de que as entidades dentro do perímetro estão protegidas e são confiáveis. Entretanto, quando algum atacante consegue penetrar o perímetro de segurança, movimentos laterais dentro da rede são difíceis de detectar e proteger.

Deste problema surgiu a ideia do modelo confiança-zero (do inglês *Zero-Trust*). Nesse modelo, por mais que um serviço esteja dentro do perímetro de segurança, onde teoricamente estaria protegido, o serviço não deveria confiar em nada, nem mesmo em outros serviços em tese confiáveis que estejam dentro do perímetro [Rose et al. 2020]. Nesse caso, espera-se o pior: que o perímetro tenha sido penetrado. Portanto, o modelo confiança-zero propõe que camadas extras de segurança sejam aplicadas, além do estabelecimento do perímetro de segurança (quando aplicável). Desse modo, os mecanismos de autenticação e autorização de todos os microsserviços, incluindo os que não são expostos à Internet, deveriam funcionar rigorosamente a todo momento.

É evidente que o modelo confiança-zero tem alinhamento com aplicações baseadas em microsserviços, e consequentemente aplicações CNC. Por outro lado, esse nível de rigor naturalmente torna a implementação das políticas de segurança mais complexas, principalmente para aplicações de microsserviços, dada a sua heterogeneidade. Pensando nisso, especialistas em segurança criaram o SPIFFE [Feldman et al. 2020] (abreviação do inglês *Secure Production Identity Framework for Everyone*): uma especificação que detalha a criação de identidades padronizadas para sistemas dinâmicos em ambientes heterogêneos, como é o caso de aplicações CNC.

As duas principais vantagens de identidades SPIFFE são indiscutivelmente a segurança e padronização. Identidades padronizadas permitem que microsserviços implementados com diferentes tecnologias sejam interoperáveis do ponto de vista de comunicação. A segurança reside na atestação da aplicação solicitante da identidade e no uso de identidades verificável e com robustez criptográfica. Identidades SPIFFE só são emitidas após atestação da integridade do ambiente no qual a aplicação está executando, além da atestação da integridade da própria aplicação. Em posse de suas identidades SPIFFE, as aplicações podem estabelecer comunicação segura usando conexões TLS (abreviação do inglês *Transport Layer Security*) mutuamente autenticadas. Assim, a aplicação do padrão

SPIFFE força as aplicações ou serviços a usar comunicações mutualmente autenticadas, provendo garantias de confidencialidade e integridade.

Sob a perspectiva de projeto e desenvolvimento de aplicações CNC usando identidades SPIFFE, pouco ou nada precisa ser alterado. Aspectos técnicos de emissão e autenticação de identidades podem ser terceirizados para tecnologias como SPIRE (abreviação do inglês *SPIFFE Runtime Environment*[10]) e *proxies* como o Envoy[11] e Ghostunnel[12]. Os aspectos de segurança podem ser mais facilmente configurados com essas aplicações e isso permite que times de desenvolvimento foquem nos aspectos lógicos da aplicação.

A próxima seção apresenta conceitos gerais sobre identidades de *software* e os motivos que levaram à criação do SPIFFE. Detalhes do padrão SPIFFE são apresentados em seguida. Por fim, são apresentados a arquitetura do arcabouço SPIRE e o fluxo básico das principais operações.

3.3.1. Conceitos Básicos sobre Identidades de Software

Identidades de *software* servem para identificar unicamente serviços e aplicações, assim como pessoas também são identificadas no mundo real. Na posse de um documento de identidade, um *software* ou pessoa pode comprovar para um terceiro quem realmente é. Para isso, o documento de identidade, seja ele de um *software* ou de um humano, é submetido a um processo de verificação de integridade.

Documentos de identidade carregam consigo algum pedaço de informação que permita comprovar sua veracidade e integridade. Um passaporte, por exemplo, possui algum tipo de carimbo ou marca d'água extremamente difícil de ser falsificado. Identidades de *software* possuem informações criptográficas que são impossíveis de serem forjadas. Só as Autoridades Certificadoras (ACs) são capazes de criar novas identidades com as informações necessárias para verificação de integridade. No Brasil temos o Instituto Nacional de Tecnologia da Informação (ITI) como AC raiz, que é encarregada de credenciar e descredenciar outras ACs intermediárias. As ACs intermediárias podem emitir identidades/certificados para serem usados por quaisquer outras organizações. Portanto, no processo de verificação de identidade, é fundamental verificar se a identidade foi emitida por uma AC proveniente de uma cadeia de confiança idônea e de reputação.

A técnica mais conhecida e adotada para realizar este processo de emissão e verificação de identidades emprega uma Infraestrutura de Chave Pública (ICP). A Figura 3.1 ilustra o funcionamento básico de uma ICP para emitir e verificar identidades. Em uma ICP, serviços podem requisitar identidades a uma AC na forma de certificados. Para isto, um serviço precisa enviar uma solicitação de assinatura de certificado (ou CSR, abreviação do inglês *Certificate Signing Request*) à AC. Após verificar que as informações prestadas no CSR são verdadeiras, a AC emitirá o certificado assinado digitalmente utilizando sua chave privada. Sempre que quiser comprovar sua identidade para uma aplicação terceira, o serviço simplesmente enviará seu certificado. Para verificar a integridade do certificado, a aplicação verificará se o certificado foi emitido por alguma AC confiável. Isto é feito usando o pacote de confiança disponibilizado publicamente pelas ACs. Esse

[10] `https://spiffe.io/docs/latest/spire-about/`
[11] `https://www.envoyproxy.io`
[12] `https://github.com/ghostunnel/ghostunnel`

pacote inclui a parte pública do par de chaves da AC, e essa chave é usada para verificar a integridade da assinatura digital no certificado.

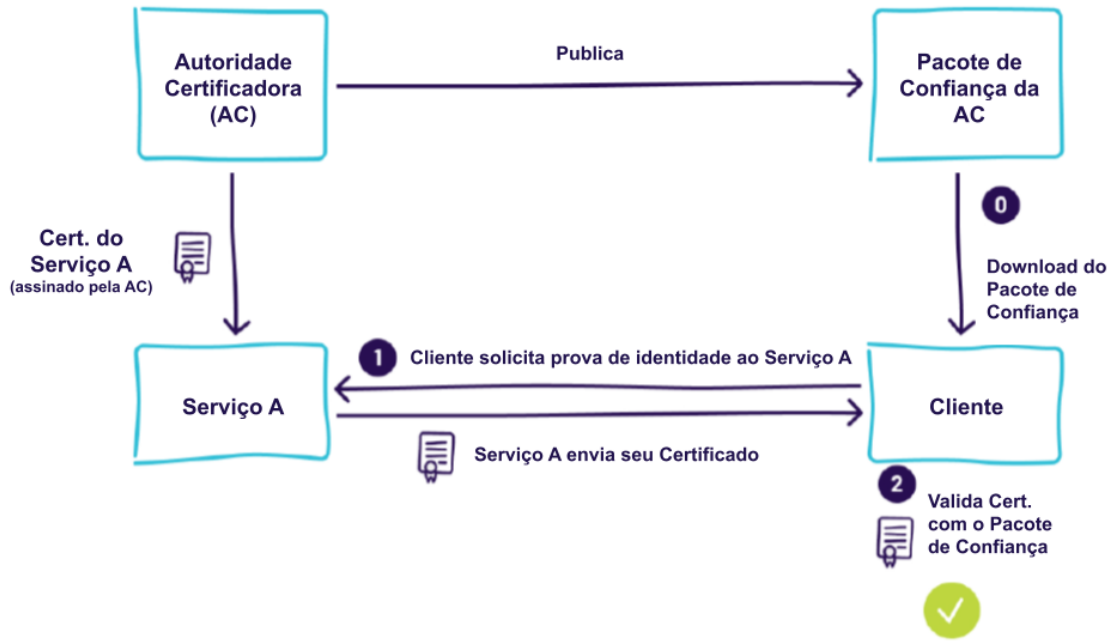


Figura 3.1. Emissão e verificação de certificados em uma ICP. Fonte: traduzido de [Feldman et al. 2020].

Identidades de *software* podem ser usadas para uma série de objetivos. O emprego mais comum de identidades é a autenticação, isto é, identificar-se para um terceiro. Os protocolos mais utilizados para tal são os certificados X.509 e JSON Web Tokens. Uma vez autenticado, um serviço pode utilizar sua identidade para obter autorização para acessar outros serviços com algum nível de restrição. Uma forma simples de implementar autorização é criando uma lista de permissão dos serviços que são autorizados a enviar requisições.

Outra aplicação extremamente importante é a proteção de dados em trânsito. Dois serviços com suas respectivas identidades podem usar o protocolo TLS para criar conexões seguras que permitem a troca de mensagens com confidencialidade e integridade. Cada certificado carrega consigo uma chave pública do serviço que o certificado identifica. Dois serviços que permutam seus certificados conseguem, a partir das chaves públicas, estabelecer uma nova chave secreta e compartilhada entre ambos que será usada para criptografar as mensagens enviadas e descriptografar as mensagens recebidas usando algum algoritmo de criptografia simétrica. No TLS 1.3, o algoritmo para criação dessa chave compartilhada é chamado Diffie-Hellman [Diffie and Hellman 1976]. Para assegurar a integridade das mensagens, cada serviço adiciona ao fim da mensagem um resumo (*digest*) do conteúdo combinado com a chave compartilhada, empregando alguma função *hash*. O algoritmo para criação desse valor de referência para integridade é chamado HMAC [Bellare et al. 1996] (abreviação do inglês *keyed-Hash Message Authentication Code*).

Identidades de *software* têm um ciclo de vida semelhante a identidades de huma-

nos. Depois de criada, toda identidade possui uma data de expiração, e isso leva à necessidade de sua renovação. Além disso, identidades podem ser revogadas. Identidades roubadas, por exemplo, precisam ser revogadas para que outros serviços maliciosos não possam usá-las para se passar pelo serviço dono da identidade. Porém, para que a revogação tome efeito é preciso que a parte verificadora valide se o certificado foi revogado, atualizando a lista de certificados revogados, e às vezes esse processo é negligenciado. Logo, é recomendável que certificados possuam tempo de expiração curto, pois a expiração breve de uma identidade protege contra situações de vazamento/roubo e remove a criticidade de verificadores negligentes que não atualizam com frequência a lista de certificados revogados.

Uma aplicação baseada em microsserviços tipicamente considera uma identidade por microsserviço. No processo de emissão de cada identidade, deveria também haver uma etapa para atestar a integridade do serviço, de modo que as informações declaradas na identidade emitida correspondam de fato ao serviço que as recebem. Considerando que os microsserviços podem ser implementados com diferentes tecnologias, diferentes estratégias de atestação podem ser executadas a depender da plataforma e tecnologia do microsserviço. Outro aspecto relacionado às identidades do microsserviço é o gerenciamento de risco relacionado ao vazamento de identidades. Uma forma de mitigar este problema é ajustando o tempo de validade de uma identidade de acordo com o nível de sensibilidade das informações que cada microsserviço processa. Naturalmente, serviços que processam dados sensíveis precisam renovar suas identidades mais frequentemente. Todo esse contexto apresentado tende a tornar o gerenciamento de identidades mais complexo quando se trata de aplicações baseadas em microsserviços, o que deu origem a uma especificação para produção segura de identidades: o SPIFFE.

3.3.2. Identidades SPIFFE

O SPIFFE é uma especificação [Feldman et al. 2020] que determina como operacionalizar identidades de *software*. Considerando a natureza heterogênea de uma aplicação baseada em microsserviços, o objetivo do SPIFFE é prover interoperabilidade para identidades de *software* de forma agnóstica quanto às plataformas e tecnologias. Para tanto, o SPIFFE define interfaces e documentos necessários para emitir e verificar identidades criptográficas de forma automatizada.

Além dos problemas relacionados à heterogeneidade dos microsserviços, é objetivo do SPIFFE também solucionar problemas relacionados à raiz de confiança. O termo "raiz de confiança" é usado para referir-se a alguma entidade ou componente de *software* no qual pode-se sempre confiar. Portanto, a raiz de confiança é a base de um sistema de autenticação, pois sem ela não seria possível assegurar a veracidade de informações sobre identidades.

A forma mais simples de autenticar um serviço é utilizando algum identificador e senha. No entanto, isso cria a necessidade de gerenciar as senhas dos microsserviços, o que poderia ser realizado através de ferramentas e repositórios de segredos como o *HashiCorp Vault*[13]. Mas para acessar o *HashiCorp Vault* também é necessário que exista alguma autenticação, o que poderia ser realizado por identificador e senha. Logo, é

[13] `https://www.vaultproject.io/`

evidente que a solução de usar senhas para autenticar microsserviços e ferramentas de gerenciamento dessas senhas criará um problema da mesma natureza – sempre existirá uma última senha a ser protegida. O SPIFFE soluciona este problema combinando procedimentos de atestação com a criação de uma cadeia de confiança determinada pelo operador da infra-estrutura.

Para melhor compreender como o SPIFFE soluciona os problemas mencionados é importante conhecer os conceitos básicos da especificação. No vocabulário SPIFFE, cada aplicação ou microsserviço é chamada de carga de trabalho (*workload*), e por esta razão, neste documento utilizaremos esse termo. Note que o conceito de aplicação inclui sistemas de tamanho e complexidade variados, e consequentemente estendemos essa abstração para o conceito de carga de trabalho. Uma carga de trabalho pode ser um servidor web, um banco de dados MySQL, uma aplicação processando itens numa fila, e até mesmo um conjunto de aplicações com propósito único empacotadas em um contêiner ou *pod*. A seguir são apresentados os demais conceitos básicos relacionados ao SPIFFE:

1. SPIFFE ID: forma de representação do nome (ou identidade) da carga de trabalho;
2. Documento de Identidade (ou SVID, abreviação do inglês *SPIFFE Verifiable Identity Document*): um documento que é passível de verificação criptográfica para provar a identidade de uma carga de trabalho a um terceiro;
3. API de Carga de Trabalho: uma API simples, localizada na mesma máquina em que a carga de trabalho executa, usada para emitir identidades sem a necessidade de autenticação;
4. Pacote de Confiança SPIFFE: um formato para representar o conjunto de chaves públicas pertencentes à cadeia de confiança da CA emissora de identidades SPIFFE;
5. Federação SPIFFE: um mecanismo para compartilhar pacotes de confiança SPIFFE e permitir verificação de identidades independente dos limites organizacionais.

Um SPIFFE ID é uma cadeia de caracteres que serve como identificação única para uma carga de trabalho. Ele é formatado como um identificador de recursos uniforme (ou URI, abreviado do inglês *Uniform Resource Identifier*) e possui três partes: o URI, o nome do domínio de confiança, e o nome ou identidade da carga de trabalho. Um exemplo simples de SPIFFE ID poderia ser `spiffe://sbc.org.br/ecos`, onde `spiffe://` é o esquema URI, `sbc.org.br` é o domínio de confiança, e `ecos` seria o nome ou identidade para o serviço de registro em eventos da SBC.

Para qualquer identidade SPIFFE o esquema URI sempre será o mesmo (`spiffe://`). O domínio de confiança corresponde à raiz de confiança do sistema, e poderia representar uma organização ou departamento executando sua própria infraestrutura SPIFFE. Cargas de trabalho que executam no mesmo domínio de confiança recebem documentos de identidade que são verificados com as mesmas chaves raiz do domínio de confiança. No entanto, também é possível usar mais de um domínio de confiança em uma mesma organização, para implementar políticas de segurança mais específicas para emitir identidades em diferentes setores de uma organização. Embora o domínio de confiança de uma

instalação SPIFFE não precise ser ancorada a uma raiz de confiança Web (*e.g.*, o domínio daquela organização na Internet), este vínculo pode existir. Por fim, não existe regra específica para formar nome ou identidade (última parte do SPIFFE ID). As organizações são livres para escolher o formato de nome que faça mais sentido para elas (por exemplo, usando tanto nomes legíveis e descritivos, como códigos opacos).

Uma carga de trabalho é uma aplicação (ou parte de uma aplicação, *e.g.*, microsserviço) responsável por realizar alguma tarefa. No contexto SPIFFE, uma carga de trabalho é tipicamente mais granular do que uma máquina física ou virtual, possuindo, às vezes, a granularidade de um processo. É comum que cargas de trabalho sejam implantadas em contêineres, pois isso ajuda no gerenciamento de dependências e alocação de recursos. Além disso, a conteinerização de cargas de trabalho promove isolamento, o que evita o roubo de identidades por cargas de trabalho maliciosas.

Similarmente como acontece em procedimentos de emissão de documentos de identidades para humanos, a emissão de documentos de identidade SPIFFE (SVID) envolve um procedimento de coleta de informações da carga de trabalho para emitir a identidade com informações íntegras. Este processo soluciona o problema de precisar armazenar senhas quando se emprega alguma autenticação, pois neste caso, a autenticação é substituída pela coleta de informações da carga de trabalho, informações estas suficientes para a emissão do SVID. Cada SVID emitido contém o SPIFFE ID referente à carga de trabalho, e é assinado pela CA que representa o domínio de confiança no qual o serviço está inserido. Por uma questão de facilidade de integração, em vez de criar um novo formato de documento, o SPIFFE adotou formatos já disseminados na comunidade, como certificados X.509 e JWTs. Um SPIFFE ID é embutido em um X509-SVID no campo de extensão "*Subject Alternative Name*". De modo geral, X509-SVIDs são preferíveis a JWT-SVIDs, dado que este último poderia ser interceptado por intermediários (em canais de comunicação inseguros) e usado em ataques de repetição, onde o SVID seria reutilizado por um terceiro para se passar pelo dono da identidade.

Todo domínio de confiança possui um pacote de confiança associado. Esse pacote de confiança é usado por um serviço para verificar a integridade de SVIDs cujo SPIFFE ID pertença ao domínio de confiança relacionado. O pacote de confiança é uma coleção de certificados que contém as chaves públicas das CAs. Como os pacotes de confiança não contém segredos, mas apenas chaves públicas, então eles podem ser compartilhados publicamente. Por outro lado, é necessário distribuí-los de forma segura para evitar modificações não autorizadas, ou seja, é importante assegurar a integridade das chaves.

Existirão situações nas quais cargas de trabalho em diferentes domínios de confiança precisarão se comunicar de forma segura. Isso acontece pois nem sempre é possível agrupar todas elas em um único domínio de confiança. Para ser capaz de estabelecer confiança nas cargas de trabalho é preciso identificá-las, e para identificá-las de modo seguro cada domínio de confiança precisa possuir os pacotes de confiança dos demais domínios. No entanto, por uma questão de segurança, as chaves das próprias CAs são rotacionadas de tempos em tempos, e por isso é importante que haja um mecanismo de compartilhamento automatizado e seguro. O SPIFFE preconiza que os pacotes de confiança sejam permutados através de serviços HTTP protegidos por TLS, chamados de *endpoints* de pacotes. Portanto, operadores que desejem federar sua infraestrutura com outros domínios

de confiança precisam configurar os domínios de segurança terceiros com seus respectivos *endpoints* de pacote, permitindo que os pacotes de confiança sejam recuperados periodicamente.

A API de Carga de Trabalho é uma API local, exposta através de um servidor gRPC, na qual a carga de trabalho usa para obter seu SVID, pacotes de confiança, e a chave privada relacionada ao SVID para assinar dados em nome da carga de trabalho. Para evitar a necessidade de que cargas de trabalho tenham credenciais, essa API não requer autenticação. A API coletará informações do sistema operacional e carga de trabalho em execução para emitir SVIDs com informações de identidade corretas. A API de Carga de Trabalho SPIFFE foi pensada de tal forma que novas soluções de verificação de informações possam ser criadas, provendo interoperabilidade e suporte para novas tecnologias. A comunicação via gRPC é bidirecional. Isto é importante pois pacotes de confiança e SVIDs são rotacionados periodicamente, e quando isto acontece, a carga de trabalho é notificada para atualização. A Figura 3.2 ilustra o funcionamento básico de uma API de Carga de Trabalho.

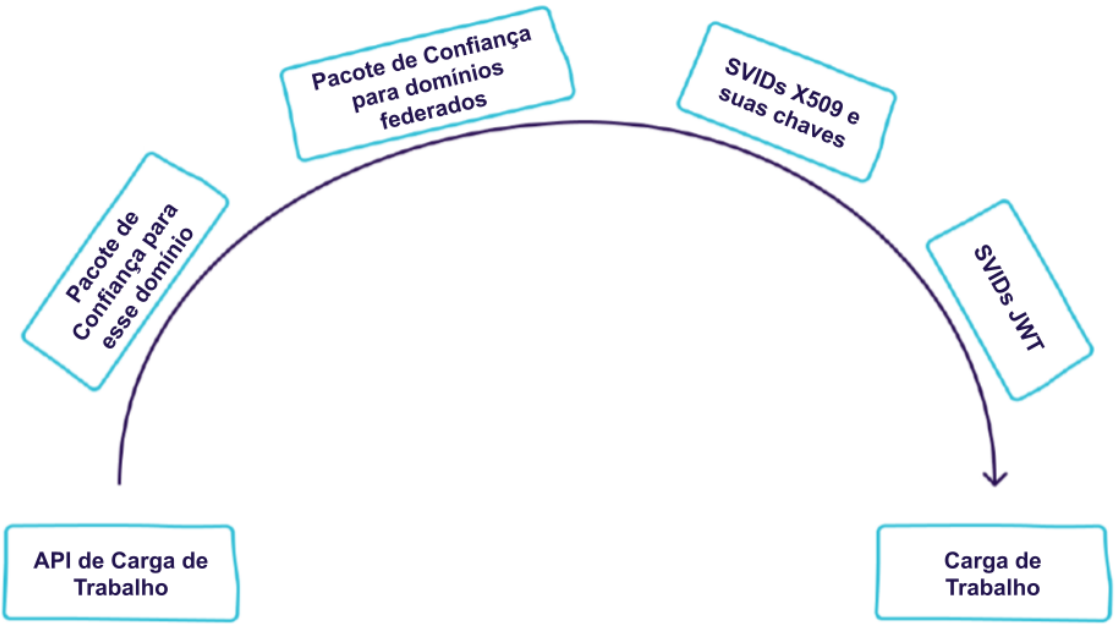


Figura 3.2. API de Carga de Trabalho fornece SVIDs e informações relacionadas. Fonte: traduzido de [Feldman et al. 2020]

3.3.3. SPIRE: *SPIFFE Runtime Environment*

O SPIRE é a implementação de referência do padrão SPIFFE para emissão de SVIDs. Embora existam outras ferramentas que também emitem identidades SPIFFE, tais como Istio, HashiCorp Consul, e Kuma, neste minicurso nós decidimos usar o arcabouço SPIRE pois ele é capaz de emitir SVIDs para serviços executando em variadas tecnologias e plataformas.

O SPIRE tem dois componentes principais: o servidor e o agente. O servidor é responsável por atestar os agentes e entregar-lhes seus respectivos SVIDs. O principal papel do agente é atestar as cargas de trabalho e emitir suas identidades SPIFFE. O SPIRE

é um projeto de código aberto, e ambos os componentes seguem uma arquitetura orientada à *plugins*, o que permite que novos mecanismos de atestação baseados em quaisquer tecnologias possam ser implementados e incorporados ao arcabouço.

Antes de aprofundar-se no funcionamento e características de servidores e agentes é importante compreender a configuração básica de implantação. Uma infraestrutura SPIRE é tipicamente composta por um servidor e múltiplos agentes. Cada agente é implantado em um nó, que geralmente é uma máquina física ou virtual. E cada agente pode servir múltiplas cargas de trabalho, com a restrição de que todas estejam executando no mesmo nó do agente. Um aspecto comum entre servidor e agente é que ambos expõem uma API. A API do servidor, também chamada de API de nó, serve principalmente para que os agentes solicitem suas identidades. Similarmente, a API de Carga de Trabalho serve para que as cargas de trabalho solicitem suas identidades. Uma ilustração dessa configuração básica de uma infraestrutura SPIRE é apresentada na Figura 3.3.

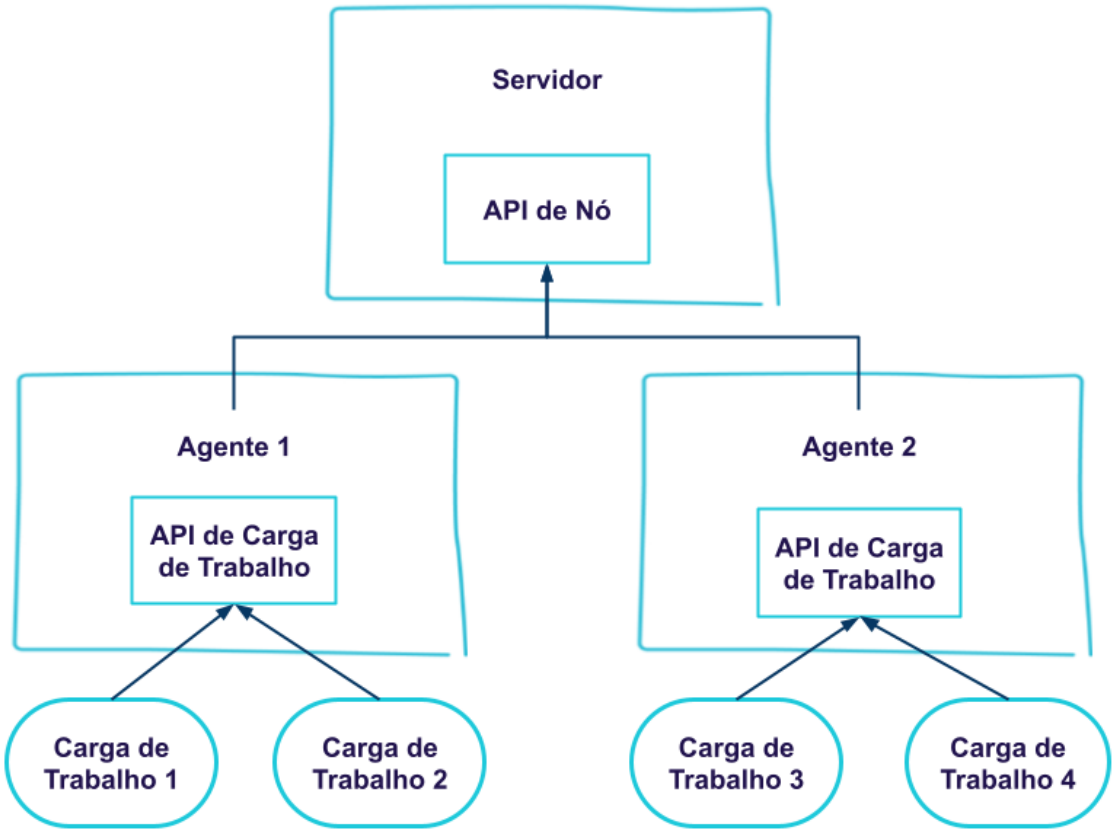


Figura 3.3. Configuração básica de uma infraestrutura SPIRE.

Uma vez compreendidas as noções básicas de uma implantação SPIRE, a seguir são apresentados detalhes de funcionamento do servidor e, posteriormente, são apresentados detalhes de funcionamento dos agentes.

3.3.3.1. Servidor SPIRE

Um servidor SPIRE é responsável por gerenciar e emitir todas as identidades do domínio de confiança no qual foi configurado. Cada SVID emitido é, portanto, assinado pelo servidor SPIRE. Na maioria dos cenários o servidor SPIRE produzirá um certificado auto-assinado para assinar os SVIDs. Esse tipo de configuração é suficiente quando as identidades são usadas dentro da própria organização do domínio de confiança. Em instalações mais amplas e complexas pode ser desejável usar a natureza hierárquica de certificados X.509 para emitir SVIDs assinados por alguma AC publicamente reconhecida. Isso permitiria vários servidores SPIRE serem capazes de emitir SVIDs para um mesmo domínio de confiança. Para esse fim, o SPIRE contém o *plugin* de autoridade *upstream*, que permite obter o certificado de assinatura a partir de outra AC.

O servidor não decide de forma autônoma as identidades a serem emitidas, e para quais nós ou cargas de trabalho elas poderiam ser emitidas. Essas informações precisam ser cadastradas no servidor através de uma interface de linha de comando (CLI, do inglês *Command Line Interface*) ou via API, e são armazenadas em um banco de dados próprio. O conjunto de informações relacionadas a uma identidade é chamado de registro de entrada (Figura 3.4). Como uma identidade pode ser atribuída a um nó ou a uma carga de trabalho, registros de entradas podem ser associados a nós ou cargas de trabalho. Um registro de entrada é composto por: um SPIFFE ID, um conjunto de um ou mais seletores, e o ID do nó (*parent id*). Os seletores são informações coletadas para identificar a carga de trabalho ou nó. O SPIFFE ID é a exata identidade que deve ser emitida para a carga de trabalho ou nó cujo seletor coletado seja igual ao seletor informado no registro. O ID do nó especifica o SPIFFE ID do nó no qual a carga de trabalho precisa estar executando para receber o SVID relacionado àquele registro de entrada. Logo, o ID do nó é aplicável somente para cargas de trabalho.

Figura 3.4. Informações do registro de entrada. Fonte: traduzido de [Feldman et al. 2020].

Documentos de identidade SPIFFE contém informações extremamente importantes acerca da identidade de algum *software*. Logo, é fundamental assegurar a veracidade das informações a serem emitidas no SVID. O SPIFFE preconiza que as informações disponíveis no ambiente de execução (nós e cargas de trabalho) sejam usadas como evidência para comprovação de identidade, e emissão dos SVIDs. Esse processo de coleta

de informações no nó ou carga de trabalho é chamado de atestação.

O servidor é o componente mais crítico da infraestrutura SPIRE, pois ele detém as chaves de assinatura. Um atacante com acesso às chaves de assinatura poderia criar quaisquer novos SVIDs com o SPIFFE ID que desejar. Desse modo o atacante poderia ter acesso a quaisquer serviços e informações dentro do domínio de confiança relacionado àquela chave de assinatura. Por esta razão, o servidor deve ser implantado em uma infraestrutura segura, que comumente é alguma máquina dentro da própria organização, mas podendo ser também uma máquina na nuvem feita confiável com algum mecanismo de proteção de integridade. Esse requisito remove a necessidade de atestação do servidor, pois ele é a raiz de confiança de uma infraestrutura SPIRE.

Pelo fato de ser a raiz de confiança, é o servidor SPIRE que atesta os nós. Ao inicializar, o agente solicita sua identidade ao servidor, que por sua vez inicia a atestação de nó. A atestação envolve algumas trocas de mensagens entre servidor e agente, de modo que o servidor possa obter os seletores do nó para criação do SVID. Essa comunicação é intermediada por um *plugin* do lado do servidor e outro *plugin* no lado do agente, ilustrados na Figura 3.5. Após receber os seletores, o servidor verifica se existe algum registro de entrada que associe algum seletor recebido com algum SPIFFE ID, e em caso positivo o servidor emite um SVID para o agente, juntamente com os registros de entrada associados àquele agente.

Figura 3.5. Arquitetura do *plugin* atestador de nó. Fonte: adaptado e traduzido de [Feldman et al. 2020].

Há três estratégias simples para atestação de nós com sistema operacional Linux e que sejam configurados manualmente: i) *token* de entrada, ii) certificado X.509 e iii) certificado SSH. A primeira estratégia consiste em gerar um *token* no servidor SPIRE e configurar o agente para apresentar o *token* no momento da atestação. Após verificar que o *token* foi de fato gerado pelo servidor, o servidor emite um SVID para o agente. A segunda abordagem emprega certificados X.509. Nesse tipo de atestação os nós são pré-configurados com certificados gerados a partir do certificado raiz ou intermediário, presente no servidor. Sempre que o servidor receber certificados que ele verifique como sendo gerados a partir do certificado raiz, então ele emitirá um SVID para o agente. A terceira opção de atestação de nó consiste em utilizar certificados SSH, que são provisionados automaticamente em alguns nós. O agente pode usar este certificado para receber um SVID com o SPIFFE ID com identidade `spire/agent/sshpop/<hash>`, onde

o *hash* é calculado a partir do próprio certificado. As três estratégias de atestação são ilustradas na Figura 3.6.

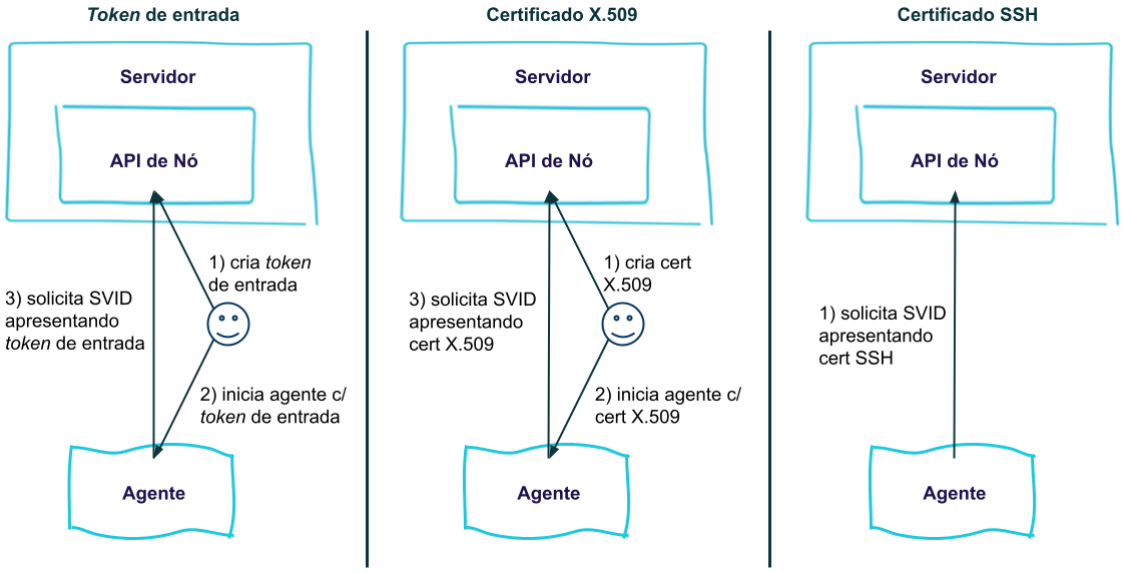


Figura 3.6. Atestação de nós com *token* de entrada, certificado X.509, e certificado SSH.

É interessante perceber que os *plugins* de atestação de nó apresentados na Figura 3.6 não requerem a criação de registros de entrada. O *plugin* que gera um *token* de entrada, por exemplo, usa o próprio *token* gerado como seletor, e o *plugin* de certificados X.509 utilizará o próprio certificado do pacote de confiança para verificar o certificado X.509 recebido pelo nó. Por fim, o *plugin* de certificados SSH sempre irá emitir um SVID para o nó baseado no *hash* do certificado SSH.

É bastante comum que os agentes e suas cargas de trabalho sejam hospedados em nuvens públicas. Muitos provedores de nuvem oferecem APIs que permitem que o nó, que tipicamente é uma VM, possa provar sua identidade. Agentes SPIRE podem usar essas APIs para atestação de nó de modo automático, sem que seja necessário configurar manualmente *tokens* de entrada ou certificados.

Instâncias computacionais EC2 (*Elastic Cloud Compute*) da *Amazon Web Services* (AWS) podem usar o *plugin* de atestação de nó chamado AWS Instance ID (AWS_IID). Com esse *plugin*, um agente é capaz de recuperar um documento de identificação da instância, assinado digitalmente pela AWS, e enviar para o servidor[14], que usará a API da AWS para verificar a integridade do documento. Além disso, o *plugin* AWS_IID é capaz de prover os seguintes seletores: etiqueta (*tag*) da instância, id do grupo de segurança, nome do grupo de segurança, e o papel no sistema de controle de acesso da AWS (*IAM role*). Em posse dos seletores, o servidor verificará se há algum registro de entrada que associe alguma identidade para algum dos seletores recebidos e, em caso positivo, emite um SVID para o agente com SPIFFE ID `agent/aws_iid/<account_id>/<regiao>/<instance_id>`. O fluxo para atestação de instâncias EC2 da AWS

[14]Essa identificação é enviada para o servidor sobre uma conexão TLS autenticada através de um pacote de inicialização no qual o agente é manualmente configurado.

é ilustrado na Figura 3.7. Também existem *plugins* de atestação de nós para instâncias computacionais de outros provedores como o *Google Cloud Platform* e *Microsoft Azure*, todos com a mesma ideia de obter e validar um documento de identificação da instância com o provedor de nuvem.

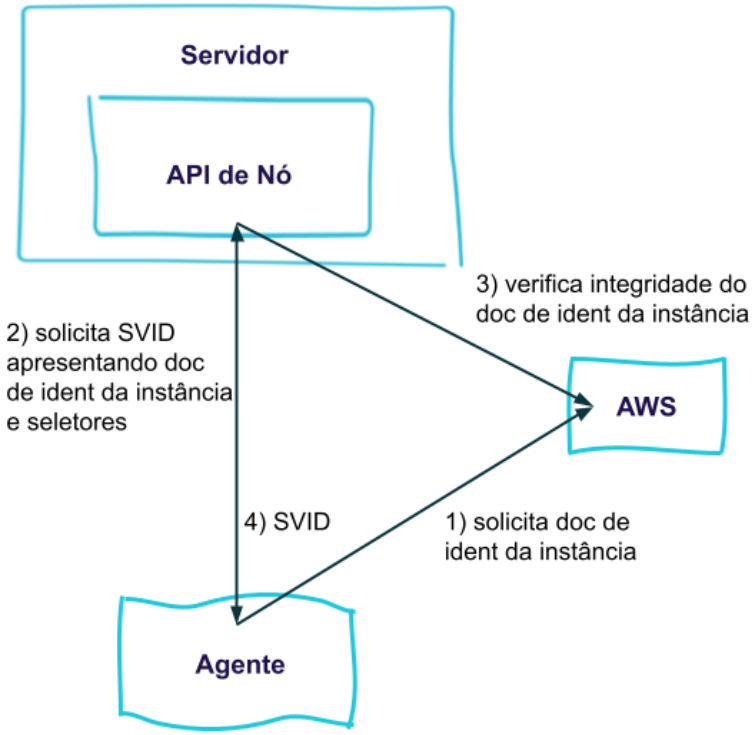


Figura 3.7. Atestação de instâncias EC2 da AWS.

Além da emissão de SVIDs para instâncias EC2 AWS no formato padrão, é possível criar registros de entrada que associem um seletor com um SPIFFE ID específico. Por exemplo, o SPIFFE ID `spiffe://<domínio-de-confiança>/broker` poderia estar em um SVID emitido para um agente SPIRE executando em uma instância EC2 AWS que possuísse uma etiqueta de instância específica, *e.g.*, `tag:app:broker`. Para tal, seria necessário criar um registro de entrada no servidor, utilizando o seguinte comando:

```
$ spire-server entry create
  -node
  -spiffeID spiffe://<domínio-de-confiança>/broker
  -selector tag:app:broker
```

Além dos *plugins* de atestação de nó e autoridade *upstream*, o servidor SPIRE contém *plugins* de banco de dados e de gerenciamento de chaves. O servidor SPIRE necessita de um banco de dados para armazenar, recuperar e atualizar informações como registros de entrada, nós previamente atestados e seus seletores. Esse *plugin* pode ser configurado para utilizar MySQL, SQLite 3 (opção padrão), ou PostgresSQL. O *plugin*

gerenciador de chaves controla como o servidor armazenará as chaves privadas utilizadas na assinatura de SVIDs.

Servidores SPIRE usam um arquivo de configuração para decidir quais *plugins* serão instanciados, além de definir detalhes da API de nó, domínio de confiança e AC. Um exemplo de arquivo de configuração é apresentado na Tabela 3.2. Mais detalhes sobre como configurar o servidor SPIRE e seus *plugins* podem ser encontrados na documentação do SPIRE[15].

```
server {
    # API
    bind_address = "0.0.0.1"
    bind_port = "8081"
    socket_path = "/tmp/spire-server/private/api.sock"
    # Autoridade certificadora e domínio de confiança
    ca_subject {
        country = ["US"]
        organization = ["SPIFFE"]
        common_name = ""
    }
    trust_domain = "example.org"
    data_dir = "./.data"
    log_level = "DEBUG"
}
plugins {
    DataStore "sql" {
        plugin_data {
            database_type = "sqlite3"
            connection_string = "./.data/datastore.sqlite3"
        }
    }
    NodeAttestor "join_token" { plugin_data {} }
    KeyManager "memory" { plugin_data = {} }
    UpstreamAuthority "disk" {
        plugin_data {
            key_file_path = "./conf/server/dummy_upstream_ca.key"
            cert_file_path = "./conf/server/dummy_upstream_ca.crt"
        }
    }
}
```

Tabela 3.2. Arquivo de configuração de um servidor SPIRE.

Por fim, é importante mencionar que a CLI do servidor SPIRE tem uma série de funcionalidades que são muito úteis. Com a CLI é possível: i) criar um *token* de entrada; ii) criar, atualizar, listar, exibir e remover registros de entrada; iii) configurar, listar, exibir e remover pacotes de confiança; iv) banir, listar, exibir e remover agentes atestados;

[15] https://spiffe.io/docs/latest/deploying/spire_server/

v) criar SVIDs no formato X.509 ou JWT; e vi) validar o arquivo de configuração do servidor.

A seguir são detalhados o funcionamento e responsabilidade de agentes SPIRE e como configurá-los.

3.3.3.2. Agente SPIRE

Um agente SPIRE deve executar em cada nó no qual as cargas de trabalho executam. Para receber seu SVID, um agente é submetido a uma atestação de nó, conforme discutido na seção anterior. Depois de atestado, um agente usa seu SVID para se comunicar com o servidor utilizando uma conexão TLS com o objetivo de obter os registros de entradas de carga de trabalho que possuam ID do nó igual ao SPIFFE ID do agente. Em seguida, o agente envia CSRs para o servidor que por sua vez retorna SVIDs para cada registro de entrada de carga de trabalho. Em algum momento as cargas de trabalho solicitarão seus SVIDs através da API de Carga de Trabalho servida pelo agente. A arquitetura do agente é ilustrada na Figura 3.8.

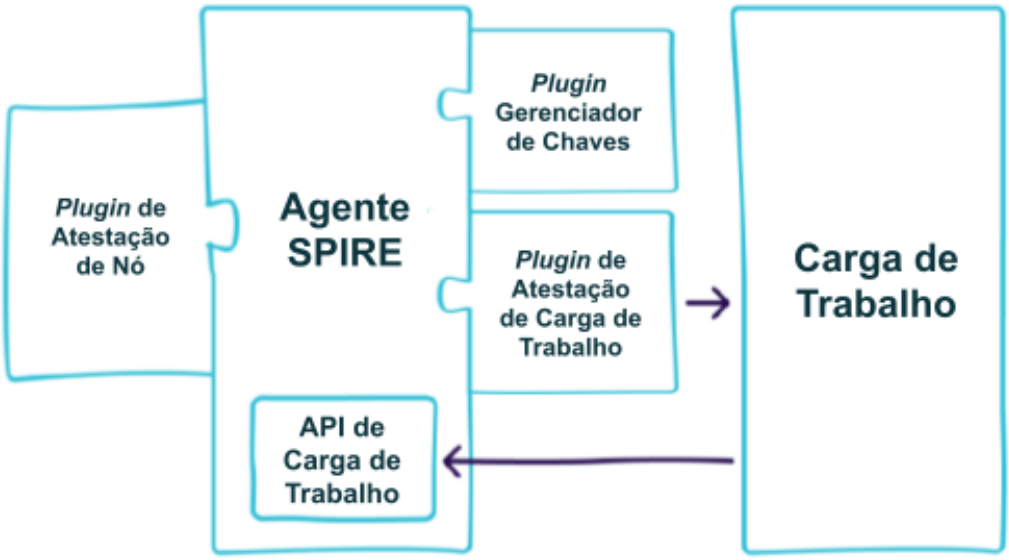


Figura 3.8. Arquitetura do agente SPIRE. Fonte: adaptado e traduzido de [Feldman et al. 2020]

Os principais componentes do agente são o *plugin* de atestação de nó, *plugin* de atestação de carga de trabalho, e *plugin* gerenciador de chaves. O *plugin* de atestação de nó que reside no agente é responsável por coletar as informações que identificam o agente e enviá-las para o servidor. O *plugin* de atestação de carga de trabalho é responsável por coletar as informações que identificam a carga de trabalho e entregá-las para o agente na forma de seletores. Por fim, o *plugin* gerenciador de chaves é encarregado de gerar e usar chaves privadas para os SVIDs X.509 emitidos para as cargas de trabalho.

No momento que um agente recebe uma solicitação de uma carga de trabalho, o agente usará as capacidades do sistema operacional para determinar exatamente qual

processo abriu aquela conexão. No caso de sistemas Linux, o agente executará chamadas de sistema para recuperar o ID do processo, o ID do usuário, e o identificador único global do sistema remoto que está se comunicando via aquele soquete específico. Os metadados do *kernel* são diferentes em sistemas BSD e Windows. Depois de identificar o processo, o agente comunica aos *plugins* de atestação o ID do processo, e a atestação segue coletando informações adicionais daquele processo de acordo com o *plugin* utilizado.

A arquitetura de *plugins* permite que diferentes estratégias e tecnologias possam ser usadas na atestação de carga de trabalho. O exemplo mais simples de *plugin* gera seletores baseados nas informações de *kernel* como usuário e grupo nos quais o processo está executando. Outro exemplo simples é o *plugin* que se comunica com o *daemon* do *Docker* para obter seletores como ID da imagem, rótulos, e variáveis de ambiente do contêiner. Há também outros *plugins* mais complexos como o do *Kubernetes* e *SCONE*, cujos funcionamentos são detalhados na seção seguinte.

Independentemente de qual seja o *plugin* de atestação de carga de trabalho, todos eles seguem um fluxo padrão, ilustrado na Figura 3.9. Primeiramente, a carga de trabalho fará uma chamada à API solicitando um SVID (seta 1). Em seguida, o agente se comunica com o *kernel* do nó para obter os seletores do processo da carga de trabalho (setas 2 e 3). Por fim, o agente confrontará os seletores coletados pelo agente com os registros de entrada providos pelo servidor para decidir se emitirá um SVID para a carga de trabalho e entregará zero ou mais SVIDs à carga (seta 4).

O SVID de cada carga de trabalho é associado a um SPIFFE ID de nó. Quando um registro de entrada de carga de trabalho é criado no servidor SPIRE, além dos seletores e SPIFFE ID da carga de trabalho, também é necessário especificar o SPIFFE ID do nó em que a carga de trabalho deve executar para ser elegível para receber seu SVID. Por exemplo, para o nó com o SPIFFE ID `spiffe://<domínio-de-confiança>/broker` ser capaz de emitir um SVID com `spiffe://<domínio-de-confiança>/queue` para uma carga de trabalho executando neste nó e disparada pelo usuário com ID 1000, seria necessário adicionar um registro de entrada no servidor com o seguinte comando:

```
$ spire-server entry create
  -parentID spiffe://<domínio-de-confiança>/broker
  -spiffeID spiffe://<domínio-de-confiança>/queue
  -selector unix:uid:1000
```

Diferentemente de atestação de nós, o agente SPIRE suporta carregar simultaneamente múltiplos *plugins* de atestação de carga de trabalho. Isso permite combinar seletores para entradas de carga de trabalho. O registro de entrada de carga de trabalho usado como exemplo poderia ser mais robusto se, além do ID do usuário, utilizasse como seletores o caminho para o binário da carga de trabalho e resumo SHA256 do binário da carga de trabalho:

```
$ spire-server entry create
  -parentID spiffe://<domínio-de-confiança>/broker
  -spiffeID spiffe://<domínio-de-confiança>/queue
  -selector unix:uid:1000
```

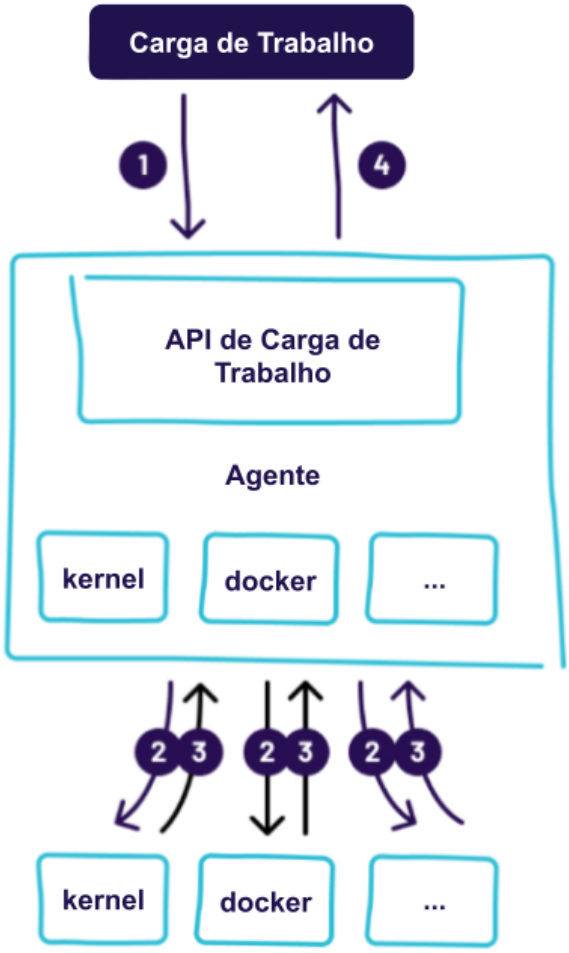


Figura 3.9. Fluxo da atestação de carga de trabalho. Fonte: traduzido de [Feldman et al. 2020]

```
-selector unix:path:/usr/bin/my_daemon
-selector unix:sha256:<SHA256-of-my-daemon>
```

A Tabela 3.3 apresenta os seletores dos *plugins* de atestação de cargas de trabalho *unix* e *docker*. A Tabela 3.4 apresenta o arquivo de configuração para um agente executando com os *plugins* de atestação de cargas de trabalho *unix* e *docker*. Mais detalhes sobre como configurar um agente SPIRE e seus *plugins* podem ser encontrados no guia de configuração dos agentes[16].

Um dos benefícios proporcionados pela infraestrutura SPIFFE é que as identidades possuem tempo de expiração e são renovadas com frequência, com o objetivo de mitigar possíveis roubos de identidades. Agentes SPIRE são responsáveis por gerar novos SVIDs e comunicar essas atualizações às cargas de trabalho interessadas. Os pacotes de confiança também são rotacionados, e os agentes monitoram essas atualizações e notificam as cargas de trabalho. O agente mantém todas essas informações em memória *cache*, de modo que os SVIDs possam ser entregues mesmo que o servidor esteja indisponível. Além disso, essa estratégia diminui os tempos de resposta pois evitam que o

[16] https://spiffe.io/docs/latest/deploying/spire_agent/

Seletor	Descrição
unix:uid	ID do usuário que disparou a carga de trabalho (*e.g.*, **unix:uid:1000**)
unix:user	Nome do usuário que disparou a carga de trabalho (*e.g.*, **unix:user:nginx**)
unix:gid	ID do grupo do usuário que disparou a carga de trabalho (*e.g.*, **unix:gid:1000**)
unix:group	Nome do grupo do usuário que disparou a carga de trabalho (*e.g.*, **unix:group:www-data**)
unix:path	Caminho para binário da carga de trabalho (*e.g.*, **unix:path:/usr/bin/my_daemon**)
unix:sha256	Resumo SHA256 do binário da carga de trabalho (*e.g.*, **unix:sha256:3a6eb0790f39...ac87c94f3856b2d**)
docker:label	Par chave:valor de cada rótulo do contêiner (*e.g.*, **docker:label:com.example.name:foo**)
docker:env	*String* de cada variável de ambiente do contêiner (*e.g.*, **docker:env:DEPLOY_MODE=production**)
docker:image_id	ID da imagem do contêiner (*e.g.*, **docker:image_id:77af4d6b9913**)

Tabela 3.3. Seletores dos *plugins* de atestação de carga de trabalho *unix* e *docker*.

agente precise se comunicar com o servidor para responder a cada solicitação das cargas de trabalho.

3.3.4. Ghostunnel Proxy

Infraestruturas SPIRE são responsáveis por emitir identidades para cargas de trabalho e gerenciar as identidades e pacotes de confiança associados. Uma vez que as cargas de trabalho estejam em posse de suas identidades elas precisam ser capazes de usá-las para comunicação segura via TLS e para autenticação. A maneira mais simples de autenticar serviços e permitir que eles se comuniquem de forma segura é utilizando *proxies* como, por exemplo, o Ghostunnel.

O Ghostunnel é um *proxy* TLS com suporte para autenticação mútua para aplicações que foram concebidas sem suporte à comunicação via TLS. O Ghostunnel pode ser executado em dois modos: cliente e servidor. Um Ghostunnel no modo servidor é implantado na frente de um servidor e aceita conexões TLS. A conexão TLS é então terminada no *proxy* que encaminha no formato plano esperado pelo serviço original (mas visível apenas na máquina local). Esse servidor pode ser uma API aceitando conexões TCP em um domínio e porta específicos, ou um soquete de domínio Unix. No modo cliente, o Ghostunnel aceita uma comunicação insegura de um serviço cliente implantado no mesmo nó e a roteia para o destino final através de uma conexão TLS.

A abordagem mais comum é implantar o Ghostunnel como um *sidecar*[17] na carga

[17]Um *sidecar* é uma espécie de módulo acoplado à carga de trabalho para adicionar funcionalidades.

```
agent {
    data_dir = "./.data"
    log_level = "DEBUG"
    server_address = "127.0.0.1"
    server_port = "8081"
    socket_path ="/tmp/spire-agent/public/api.sock"
    trust_bundle_path = "./conf/agent/root_ca.crt"
    trust_domain = "example.org"
}

plugins {
    NodeAttestor "join_token" { plugin_data {} }
    KeyManager "disk" {
        plugin_data {
            directory = "./.data"
        }
    }
    WorkloadAttestor "unix" {
        plugin_data {
         # Se verdadeiro, o caminho da carga de trabalho será descoberto
         # pelo plugin para prover seletores adicionais.
         discover_workload_path = false

         # Limite do tamanho do binário da carga de trabalho quando for
         # calcular alguns seletores (e.g., sha256).
         # O valor 0 não impõe limite.
         workload_size_limit = 0
        }
    }
    WorkloadAttestor "docker" {
        plugin_data {
         # The location of the docker daemon socket.
         docker_socket_path = ""

         # The API version of the docker daemon.
         docker_version = ""
        }
    }
}
```

Tabela 3.4. Arquivo de configuração de um agente SPIRE.

de trabalho. A ideia de executar o Ghostunnel como um *sidecar* tem como objetivo permitir que ele seja atestado e obtenha uma identidade SPIFFE. Em posse do SVID o Ghostunnel no modo cliente ou no modo servidor será capaz de estabelecer uma conexão TLS com o serviço final. A Figura 3.10 ilustra a implantação do Ghostunnel como *sidecar* para cargas de trabalho em nós diferentes.

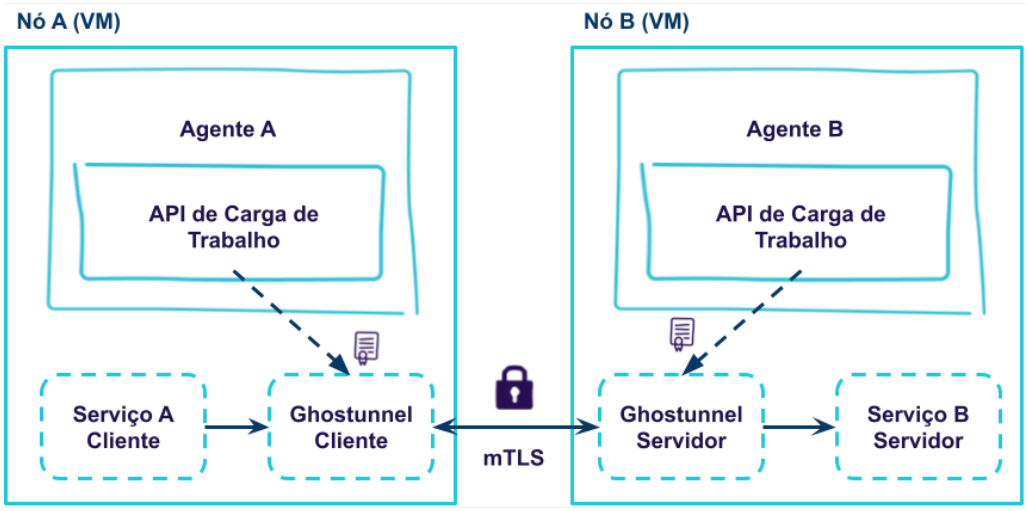


Figura 3.10. *Proxy* Ghostunnel para comunicação segura entre dois serviços via mTLS.

O Ghostunnel no modo servidor vai receber um SVID do agente local, permitindo que clientes remotos confirmem que se comunicam com o servidor correto, e também podem filtrar conexões com origem em cargas com SVIDs com SPIFFE IDs específicos, o que permite limitar a conexão a cargas autorizadas. De modo semelhante, o Ghostunnel no modo cliente recebe um SVID do agente local e usa este SVID para representar o cliente que não possuia um SVID. Além disso, ele também pode filtrar com quais servidores pode se comunicar validando o SVID do servidor.

A Tabela 3.5 lista alguns comandos[18] que permitem reproduzir o caso de uso ilustrado na Figura 3.10. Para fins de simplificação, são usadas cargas de trabalho Unix. Suponha que um usuário com uid 1000 esteja executando uma carga de trabalho Unix. Além de iniciar o serviço A (cliente) o usuário precisaria iniciar um processo do Ghostunnel no modo cliente. Em outro nó é necessário instanciar o serviço B (servidor) e um processo Ghostunnel no modo servidor. Ambos os processos Ghostunnel seriam atestados pelos agentes de seus respectivos nós e receberiam seus SVIDs.

A próxima seção introduz o terceiro pilar para construção de aplicações distribuídas e seguras contra vazamentos de dados: a computação confidencial. Além de apresentar os conceitos básicos de computação confidencial, a seção apresenta o arcabouço SCONE

3.4. Computação confidencial

O crescente número de casos envolvendo o uso indevido de dados pessoais, ou até mesmo o vazamento desses dados, tem aumentado o interesse da comunidade por técnicas que auxiliem a preservar a privacidade dos dados. Para proteger dados em repouso, uma abordagem bastante conhecida e eficiente é criptografar os dados. Os principais cuidados a serem tomados nesses casos são a escolha de um algoritmo de criptografia robusto, uma

[18]Uma descrição completa da demonstração, incluindo detalhes de instalação e configuração de SPIRE e Ghostunnel, pode ser encontrada em `https://github.com/ufcg-lsd/minicurso-sbseg-2021` no diretório `demo-ghostunnel`.

```
# Iniciando o Servidor SPIRE e obtendo tokens de entrada para os agentes
spire-server run -config conf/server/server.conf
spire-server token generate -socketPath ./data/server.sock
    -spiffeID spiffe://example.org/agent1
spire-server token generate -socketPath ./data/server.sock
    -spiffeID spiffe://example.org/agent2

# Iniciando os Agentes em nós diferentes com seus respectivos tokens
spire-agent run -config conf/agent/agent.conf -joinToken "<token>"

# Criando registros de entrada para cargas de trabalho com user id 1000
spire-server entry create -socketPath ./data/server.sock
    -selector unix:uid:1000
    -spiffeID spiffe://example.org/proxy/ghostunnel-client
    -parentID spiffe://example.org/agent1
spire-server entry create -socketPath ./data/server.sock
    -selector unix:uid:1000
    -spiffeID spiffe://example.org/proxy/ghostunnel-server
    -parentID spiffe://example.org/agent2

# Iniciando carga de trabalho nos agentes
# Agente 1: Ghostunnel no modo cliente
ghostunnel-v1.6.0-linux-amd64 client
    --use-workload-api-addr "unix://${PWD}/data/agent.sock"
    --listen=localhost:9001 --target=10.11.19.207:8081

# Agente 2: Ghostunnel no modo servidor
ghostunnel-v1.6.0-linux-amd64 server
    --use-workload-api-addr "unix://${PWD}/data/agent.sock"
    --listen=10.11.19.207:8081 --target=localhost:9001
    --allow-uri spiffe://example.org/proxy/ghostunnel-client

# Demonstração: comunicação mTLS via GHostunnel
# Agente 2
socat TCP-LISTEN:9001,fork STDOUT
# Agente 1
echo "SBSEG2021" | socat STDIN TCP:localhost:9001
```

Tabela 3.5. Comandos para reproduzir demonstração ilustrada na Figura 3.10.

senha forte e o modo de armazenamento dessa senha. Contudo, por mais cuidadosa que seja a aplicação da criptografia, envolvendo todas as etapas de transporte e persistência de dados, os sistemas convencionais vão precisar descriptografar esses dados para o processamento (a realização de consultas, a análise ou transformação dos dados). Essa situação cria uma vulnerabilidade que é a possibilidade de roubo desses dados durante a computação, pois durante a computação é preciso que os dados estejam descriptografados na memória. Deste problema surgiu uma área de estudo chamada computação confidencial,

que trata de propor estratégias seguras para processamento de dados sensíveis.

Há duas principais abordagens para computação confidencial. Uma abordagem utiliza recursos de *hardware*, como por exemplo, a tecnologia Intel Software Guard Extensions (SGX) [Costan and Devadas 2016] que é considerada neste trabalho. A outra é baseada em técnicas algorítmicas, como computação homomórfica [Gentry 2009] (HE, do inglês, *Homomorphic Encryption*) e computação multi-parte [Cramer et al. 2015] (MPC, do inglês, *Multi-Party Computation*).

HE e MPC são técnicas que realizam processamento sobre dados numéricos criptografados. Portanto, o dado só é entregue a entidades não confiáveis após ser criptografado, que neste caso consiste em transformar cada valor numérico em um polinômio. Embora não possam visualizar os dados originais, essas entidades conseguem realizar operações sobre os dados criptografados. Uma das principais diferenças entre HE e MPC é que para HE a computação pode acontecer em um único servidor, enquanto que um aspecto básico do MPC é que múltiplas instâncias de processamento realizam a computação de forma colaborativa. Embora não dependam de *hardware* específico para sua utilização, estas abordagens têm limitações consideráveis, tais como: i) elevado tempo de processamento, pois simples operações aritméticas são convertidas em onerosas operações polinomiais; ii) a complexidade de adaptar uma aplicação para usar essas técnicas [Naehrig et al. 2011, Hayward and Chiang 2015], pois não há ferramentas de uso geral para conversão de código; e, iii) a falta de proteção de integridade para o código, incluindo a falta de um mecanismo de atestação remota.

Assim, as abordagens baseadas em *hardware* têm sido crescentemente utilizadas para resolver o problema de processamento de dados sensíveis. Estas abordagens utilizam tecnologias de computação confiável (TEE, do inglês *Trusted Execution Environments*). Durante a computação, TEEs isolam o código e os dados sensíveis em áreas protegidas da memória, tipicamente chamadas enclaves. Os dados sensíveis e a aplicação que estejam dentro do enclave não são acessíveis por programas não autorizados, ou por usuários com o mais alto nível de privilégio (*e.g.*, usuários administradores), o que permite a computação segura mesmo em ambientes não confiáveis (sejam ambientes remotos, operados por terceiros, ou ambientes locais, mas onde se quer proteger contra ataques internos) [Costan and Devadas 2016]. Do ponto de vista de desempenho, computação confidencial por TEEs é ordens de magnitude mais rápida do que HE e MPC, podendo ter desempenho semelhante ao de aplicações não-confidenciais. Um aspecto negativo é que portar aplicações para serem executadas em TEEs também não é tarefa trivial. A Intel, por exemplo, disponibiliza um kit de desenvolvimento de *software* (SDK, do inglês *Software Development Kit*) para SGX. No entanto, o desenvolvedor precisa entender uma série de aspectos técnicos para incorporar o SDK à sua aplicação. Detalhes de desenvolvimento de aplicações confidenciais usando o Intel SGX SDK podem ser encontrados em [Severinsen 2017].

Uma alternativa que torna mais simples o desenvolvimento de aplicações confidenciais é o uso de abordagens conhecidas como *lift-and-shift*, onde aplicações podem ser executadas em modo confidencial sem a necessidade de modificações. Alguns dos arcabouços *lift-and-shift* mais populares para desenvolvimento de aplicações confidenciais são o GrapheneSGX [Tsai et al. 2017], Fortanix [Leiserson 2018] e SCONE

[Arnautov et al. 2016]. Para este trabalho decidimos utilizar o arcabouço SCONE pois, ao contrário do GrapheneSGX, tem suporte a mecanismos de orquestração de contêineres e suporte ao framework SPIRE. Além disso, ao contrário do Fortanix, possui documentação abrangente[19].

As próximas seções apresentam os principais conceitos relacionados ao arcabouço SCONE.

3.4.1. SCONE

O objetivo do SCONE (abreviação do inglês, *Secure CONtainer Environment*) é facilitar o caminho de um desenvolvedor no processo de migrar aplicações confidenciais para Intel SGX de forma transparente. Nessa abordagem, os desenvolvedores não precisam entender detalhes técnicos de quais partes do código executam dentro do enclave e quais executam fora, pois todo o código do usuário é inserido dentro do enclave e o SCONE injeta código adicional para abstrair operações que não poderiam ser executadas dentro do enclave (como chamadas de sistema).

Além do suporte em termos de *runtime*, são disponibilizados contêineres com as dependências de baixo nível como `musl-libc`, `glibc`, e o próprio compilador `gcc`, já adaptadas para usar o Intel SGX. Portanto, se a linguagem de programação for compilada, desenvolvedores precisam apenas implantar suas aplicações nesses contêineres, usando as bibliotecas adaptadas ou, no pior caso, recompilar as aplicações com o compilador disponibilizado no contêiner. No caso de linguagens interpretadas como Python e Java, imagens estão disponíveis com versões SGX dos interpretadores. Combinando compiladores e interpretadores, SCONE suporta uma variedade de linguagens de programação como C, C++, Java, Python, Go e JavaScript. Durante suas execuções, as aplicações executarão totalmente dentro dos enclaves, protegendo dados e a própria aplicação contra vazamentos ou acessos não autorizados.

Abordagens *lift-and-shift* como o SCONE podem ser usadas para criar microsserviços confidenciais, e isso possibilita que continuemos a explorar as vantagens da utilização de provedores de nuvem sem a preocupação com a confidencialidade da aplicação e dados. Neste trabalho, aplicações confidenciais construídas com SCONE estão sendo chamadas de aplicações de Computação Confidencial Nativa da Nuvem (ou ConfCNC, do inglês *Confidential Cloud Native Computing*). Pelo fato de serem executadas na forma de microsserviços e em contêineres, aplicações ConfCNC são fáceis de serem integradas com uma série de ferramentas populares do ecossistema CNC, incluindo o SPIFFE e SPIRE.

Para entendermos como o SCONE deve ser usado para a migração de aplicações, precisamos entender dois conceitos principais: i) o provisionamento de segredos para aplicações atestadas; e, ii) proteção do sistema de arquivos.

A atestação remota em SCONE é o processo em que uma porção de código da plataforma SCONE garante que a carga de trabalho está íntegra e executando em um processador equipado com Intel SGX. A atestação remota envolve dois componentes: o CAS (do inglês, *Configuration and Attestation Service*) e o LAS (do inglês, *Local Attestation*

[19] `https://sconedocs.github.io`

Service). O CAS é o responsável por atestar uma aplicação SCONE antes de provisionar segredos e configurações de forma segura. O LAS, por sua vez, é o agente responsável pela atestação local da aplicação SGX, gerando um *quote* de atestação que é, então, enviado ao CAS. O *quote* contém informações do enclave a ser atestado, em especial, o MRENCLAVE, um SHA256 da imagem de memória esperada do enclave (incluindo código e as páginas de memória) e o MRSIGNER, que indica o desenvolvedor que assinou aquele código. Além das informações do enclave, o *quote* inclui informações da plataforma, incluindo dados sobre funcionalidades que influenciam a atestação, como a versão do *firmware* do processador e o estado de funcionalidades como o *hyperthreading*. Para gerar o *quote* o LAS precisa executar na mesma máquina que a carga de trabalho. Assim, enquanto um LAS serve um nó, um CAS pode servir um ou mais nós.

O fluxo de atestação funciona da seguinte forma. Ao ser executada, a carga de trabalho aciona código do próprio arcabouço SCONE que foi injetado durante a compilação do código ou do próprio interpretador. Este código injetado aciona o LAS e com o *quote* resultante, entra em contato com o CAS. Cada carga de trabalho deve ser previamente registrada no CAS e esse registro inclui informações do estado esperado do ambiente e informações das configurações esperadas pela aplicação específica. Como parte do estado do ambiente, podemos citar o MRENCLAVE do binário a ser executado e um *hash* da parte do sistema de arquivos relevante para a aplicação (por exemplo, contendo bibliotecas a serem carregadas dinamicamente). Já a parte das informações de configuração de uma aplicação específica podem incluir a definição de variáveis de ambiente, parâmetros de linha de comando ou arquivos secretos, como certificados e chaves.

É importante notar que as informações sensíveis precisam ser provisionadas para a aplicação via CAS e somente depois da atestação, pois só assim se pode garantir que as configurações não foram modificadas e que nenhum segredo foi entregue a uma aplicação que não passaria no teste de integridade.

O cadastro de aplicações no CAS é feito usando as chamadas sessões. Um arquivo de sessão simples é mostrado no Código 3.1. Neste arquivo podemos ver os seguintes componentes:

- O nome do serviço (`alo-mundo`) e o MRENCLAVE do binário que será executado (neste caso, o interpretador Python).
- Os parâmetros a serem passados para o binário (neste caso, o código a ser interpretado).
- Uma variável de ambiente contendo um segredo, variável esta que somente será visível dentro do enclave).
- O caminho para um arquivo que descreve os *hashes* dos arquivos que precisam ser protegidos (`fspf_path`). Neste caso, o próprio código-fonte da aplicação do usuário (`programa.py`) deveria estar protegido, assim como a chave e o *hash* base da parte do sistema de arquivos protegida.

```
1 name: sessao-exemplo
```

```
2  version: 0.3
3  services:
4      - name: alo-mundo
5        mrenclaves: [$MRENCLAVE]
6        command: python3 /app/programa.py
7        pwd: /
8        environment:
9          VARIAVEL_SECRETA: "conteudosecreto"
10       fspf_path: /fspf-file/volume.fspf
11       fspf_key: $SCONE_FSPF_KEY
12       fspf_tag: $SCONE_FSPF_TAG
13
14 security:
15     attestation:
16       tolerate: [debug-mode, hyperthreading, outdated-tcb]
17       ignore_advisories: "*"
```

Código 3.1. Arquivo `sessao.yml` submetido ao CAS descrevendo uma carga de trabalho simples.

A próxima seção ilustra a utilização dos conceitos acima através de um exemplo.

3.4.2. Exemplo de uso do SCONE

Esta seção descreve resumidamente os passos para a execução de uma aplicação SCONE, mais detalhes podem ser encontrados em [Brito et al. 2020]. Assumimos que os componentes básicos como o *driver* SGX e o Docker estão instalados[20].

Para executar um exemplo com o SCONE vamos começar com um código Python (Código 3.2).

```
1 import os
2 print("Alo, mundo!")
3 secret_env = os.environ.get("VARIAVEL_SECRETA")
4 print(f"O segredo é: {secret_env}")
```

Código 3.2. Arquivo `programa.py`, com um programa Python que será executado com SCONE.

Em uma máquina confiável, por exemplo, a máquina local do desenvolvedor, podemos então gerar o sistema de arquivos protegido. O Código 3.3 descreve um *script* para facilitar o trabalho. Ele indica que um diretório será protegido (`app`), mas o diretório raiz não será protegido.

```
1 scone fspf create /fspf/volume.fspf
2 scone fspf addr /fspf/volume.fspf / --not-protected --kernel /
3 scone fspf addr /fspf/volume.fspf /app --encrypted --kernel /app
4 scone fspf addf /fspf/volume.fspf /app /native-files /app
5 scone fspf encrypt /fspf/volume.fspf > /native-files/keytag
```

Código 3.3. Arquivo `fspf.sh`, para proteger o arquivo fonte.

[20]Para versões do Kernel do Linux anteriores à versão 5.11, instruções para a instalação do *driver* estão em `https://sconedocs.github.io/sgxinstall/`. Instruções para o Docker podem ser encontradas em `https://docs.docker.com/engine/install/ubuntu/`.

Em seguida, executamos o *script* gerado usando uma imagem do SCONE (Código 3.4) que inclui alguns utilitários de linha de comando. Ao fim da execução, uma versão criptografada do arquivo `programa.py` será gerada. O *hash* do sistema de arquivos e a chave estarão no diretório do código-fonte, uma vez que devem ficar na máquina do desenvolvedor.

```
# Criamos o diretório para as diferentes versões dos arquivos
# (e para o nosso arquivos de controle, fspf.sh e volume.fspf)
mkdir fspf native-files encrypted-files
cp programa.py native-files/
chmod +x fspf.sh
cp fspf.sh fspf/
export IMAGEM=spire:network-shield-python-alpha3

docker run -it --rm --device /dev/isgx \
 -v $PWD/fspf:/fspf \
 -v $PWD/native-files:/native-files \
 -v $PWD/encrypted-files:/app \
 registry.scontain.com:5050/sconecuratedimages/$IMAGEM \
 bash -c /fspf/fspf.sh
```

Código 3.4. Arquivo `fspf.sh`, para proteger o arquivo fonte.

Para executar esta carga de trabalho dentro de um contêiner Docker, precisamos a seguir de um arquivo de definição de imagem, ou Dockerfile (Código 3.5). Note que apenas a versão criptografada do programa e o arquivo com os *hashes* individuais são copiados. Os dados sensíveis não são copiados (*hash* raiz, chave e código original).

```
ARG IMAGEM=spire:network-shield-python-alpha3
FROM registry.scontain.com:5050/sconecuratedimages/${IMAGEM}
COPY encrypted-files/programa.py /app/programa.py
COPY fspf/volume.fspf /fspf-file/volume.fspf
ENTRYPOINT [ "python3", "/app/programa.py" ]
```

Código 3.5. Arquivo `scone.Dockerfile`, para gerar uma imagem Docker com o exemplo.

Ainda na máquina confiável, o Código 3.6 detalha os passos para a geração da imagem do contêiner e o registro da carga de trabalho no CAS usando a sessão anterior (Código 3.1). Para cadastrar a carga de trabalho precisamos substituir os valores do MRENCLAVE, do *hash* do sistema de arquivos (`fspf_tag`) e da chave do sistema de arquivos (`fspf_key`).

```
# Geração da imagem
docker build . -t sbseg-alo-mundo-scone -f scone.Dockerfile

# Para a primeira execução precisamos descobrir o MRENCLAVE
# Vamos fazer isso dentro do contêiner
docker run -it --rm --device /dev/isgx --entrypoint /bin/bash \
    sbseg-alo-mundo-scone

# Dentro do contêiner, descobrimos o MRENCLAVE do binário com:
```

```
10 SCONE_HASH=1 python3
11
12 # Vamos usar um servidor CAS público
13 # Antes de usar, precisamos estabelecer a confiança nele.
14 scone cas attest 5-4-0.scone-cas.cf -GCS
15     --only_for_testing-trust-any --only_for_testing-ignore-signer \
16     --only_for_testing-debug
17
18 # Finalmente, enviamos a sessão para o CAS público
19 scone session create sessao.yml
```

Código 3.6. Passos para a geração da imagem e registro da carga de trabalho.

O último passo, descrito no Código 3.7 é a execução da aplicação, na imagem do contêiner gerado. Como mencionado anteriormente, para executar uma aplicação SCONE, a máquina precisa ter um componente LAS executando localmente. Assim, o trecho de código mostra como executar um LAS e configura algumas variáveis relevantes: o endereço do LAS (local via Docker), o endereço do CAS (remoto, estamos usando um servidor público) e o nome do serviço a ser executando (o nome do serviço e da sessão onde ele está definido).

```
1  # Configurando o CAS remoto, o LAS local e o nome da sessão
2  export SCONE_CAS_ADDR=5-4-0.scone-cas.cf
3  export SCONE_LAS_ADDR=172.17.0.1
4  export SCONE_CONFIG_ID=sessao-exemplo/alo-mundo
5
6  # A máquina precisa ter um LAS executando localmente, por exemplo:
7  docker run -dt --rm --name las --device /dev/isgx -p 18766:18766 \
8    registry.scontain.com:5050/sconecuratedimages/spire:las-scone5.4.0
9
10 # Execução remota (a imagem deveria ser disponibilizada)
11 docker run -it --rm --device /dev/isgx \
12   -e SCONE_CAS_ADDR=$SCONE_CAS_ADDR \
13   -e SCONE_LAS_ADDR=$SCONE_LAS_ADDR \
14   -e SCONE_CONFIG_ID=$SCONE_CONFIG_ID \
15   sbseg-alo-mundo-scone
```

Código 3.7. Executando a aplicação remotamente.

Como resultado da execução o código de exemplo terá acesso à variável de ambiente com o segredo. Caso o código do interpretador não passe na atestação, o código fonte ficará indisponível, pois o binário não terá acesso à chave de descriptografia.

3.5. Estudo de Caso

Os conceitos e técnicas providos pelo paradigma de computação nativa da nuvem, o que inclui o padrão SPIFFE e ferramentas Kubernetes e SPIRE, tornam mais eficientes a construção e operação de aplicações distribuídas a serem servidas na nuvem. O SPIFFE facilita a implantação do modelo confiança-zero, facilitando a autenticação dos serviços e permitindo comunicação sobre canais seguros através do protocolo TLS. Finalmente,

tecnologias de computação confidencial podem tornar tais aplicações ainda mais seguras visto que protegem o dado durante seu processamento, mesmo que o atacante tenha controle do provedor de infra-estrutura.

Para um aprendizado prático, apresentamos um estudo de caso que considera uma aplicação distribuída construída sobre esses três pilares fundamentais. A aplicação é composta por um microsserviço produtor não-confidencial, e outro microsserviço consumidor confidencial utilizando Intel SGX através do arcabouço SCONE. A ferramenta SPIRE é utilizada para gerenciamento das identidades SPIFFE que são empregadas para autenticação e comunicação segura. Para a comunicação usamos o Apache Kafka, e portanto o serviço produtor (não-confidencial) envia mensagens para o serviço consumidor (confidencial) através de um barramento de mensagens. A Figura 3.11 ilustra os microsserviços, os componentes básicos do SPIRE, e o fluxo que permite a emissão de identidades aos microsserviços

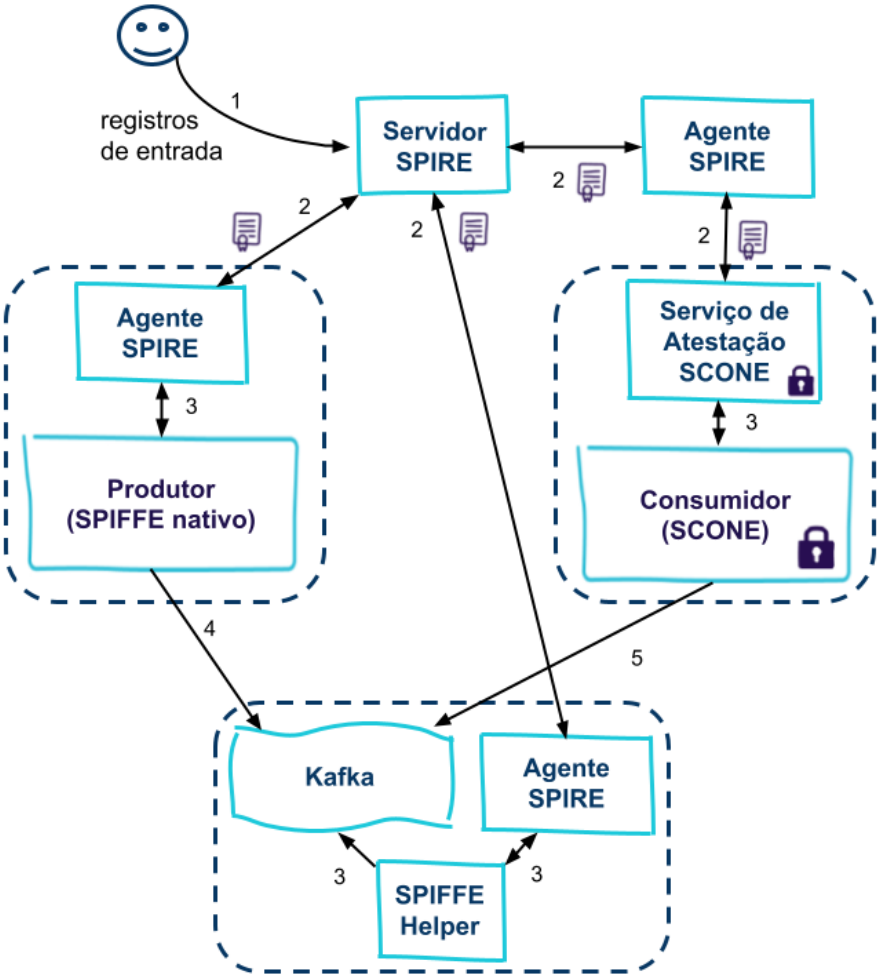


Figura 3.11. Fluxo do estudo de caso.

O propósito da aplicação distribuída foi simplificado para facilitar a compreensão dos aspectos técnicos relacionados às identidades SPIFFE. O serviço produtor gera periodicamente identificadores universalmente únicos (UUID, do inglês *universally unique*

identifier) e os publica no Kafka. O serviço consumidor se inscreve no tópico e recebe os *UUIDs* produzidos. Ambas as aplicações se comunicam com o Kafka e empregam identidades SPIFFE para estabelecimento de um canal protegido por TLS mutuamente autenticado.

O produtor é um serviço escrito em *Golang*, projetado para consultar de forma autônoma a API de Carga de Trabalho para recuperar sua identidade antes de começar a publicar mensagens no Kafka (seta 4 da Figura 3.11). O Código 3.8 apresenta um trecho da implementação do produtor responsável por configurar a obtenção do SVID junto ao SPIRE (seta 3 da Figura 3.11), além de configurar o SVID para comunicação segura com o Kafka.

```
import (
        //outros pacotes
        "github.com/spiffe/go-spiffe/v2/spiffeid"
        "github.com/spiffe/go-spiffe/v2/spiffetls/tlsconfig"
        "github.com/spiffe/go-spiffe/v2/workloadapi"
        kafka "github.com/segmentio/kafka-go"
)

// Configuração da API de Carga de Trabalho
source, err := workloadapi.NewX509Source(ctx,
    workloadapi.WithClientOptions(workloadapi.WithAddr(APISocketPath)))

// SPIFFE ID do Kafka e configuração do TLS
spiffeID := spiffeid.Must(spiffeTrustDomain, kafkaID)
tlsConfig := tlsconfig.MTLSClientConfig(source, source,
    tlsconfig.AuthorizeID(spiffeID))

// Usando o dialer do Kafka com configurações de TLS
dialer := &kafka.Dialer{
        Timeout:   10 * time.Second,
        DualStack: true,
        TLS:       tlsConfig,
}
```

Código 3.8. Serviço produtor: trecho com configuração para identidades SPIFFE.

O Kafka, por ser uma aplicação legada, não possui integração nativa com SPIFFE. Para esses casos de código legado recomenda-se a implementação de um *sidecar*: um módulo implantado próximo à aplicação com a responsabilidade de monitorar a API de Carga de Trabalho para solicitar um SVID e configurá-lo de modo que a aplicação consiga usá-lo de forma transparente. Na seção 3.3.4 foi explicado como utilizar o Ghostunnel como *sidecar*, mas para apresentar outra alternativa, utilizamos o *SPIFFE Helper* neste estudo de caso. O *SPIFFE Helper* é, portanto, uma aplicação *sidecar* que basicamente obtém SVIDs e os disponibiliza para a aplicação legada (setas 2 e 3 da Figura 3.11), atualizando-os quando necessário. O código-fonte do *SPIFFE Helper* é aberto[21]. Como o *SPIFFE Helper* foi pensado para facilitar a integração do SPIFFE com código legado, sua exe-

[21] https://github.com/spiffe/spiffe-helper

cução é simples: `spiffe-helper -config <arquivo_de_configuracao>`. O arquivo de configurações, apresentado no Código 3.9, contém uma série de valores associados à localização e nomes de chaves e certificados, além de um *script* que é executado sempre que o *SPIFFE Helper* obtém as informações do SPIRE. Para o estudo de caso apresentado, o *script* `helper_cmd.sh` (linha 2 do Código 3.9) codifica o certificado em um formato legível pelo Kafka, e notifica o Kafka para que ele atualize seus certificados.

```
1 agentAddress = "$AGENT\_SOCKET"
2 cmd = "/entrypoint/helper_cmd.sh"
3 certDir = "/store"
4 renewSignal = ""
5 svidFileName = "kafka-server-cert.pem"
6 svidKeyFileName = "kafka-server-key.pem"
7 svidBundleFileName = "ca-cert.pem"
8 addIntermediatesToBundle = false
```

Código 3.9. Arquivo de configuração do *SPIFFE Helper*.

Finalmente, o consumidor é um microsserviço confidencial, escrito em *Python*, que executa dentro de um enclave SGX através do arcabouço SCONE. O SPIRE não possuía nenhuma implementação de *plugin* para atestação de cargas de trabalho SGX. Algumas abordagens para atestação de cargas de trabalho SGX foram sugeridas recentemente por Silva et. al (2021), e reutilizamos uma das propostas no estudo de caso apresentado. Em linhas gerais, Silva et. al (2021) propõem que sempre que um Agente SPIRE receba um registro de entrada associado a uma carga de trabalho SCONE ele deve publicar os seletores, chaves e certificados em um serviço de atestação SCONE (seta 2 da Figura 3.11). O serviço de atestação SCONE possui alguns módulos responsáveis por realizar a atestação da carga de trabalho SCONE e apenas conceder acesso aos segredos (SVID e chaves) mediante sucesso da atestação (seta 3 da Figura 3.11).

Para facilitar a implantação e orquestração dos serviços usamos o Kubernetes. Por isso, tanto os serviços produtor como consumidor são atestados usando seletores do Kubernetes. Adicionalmente, o serviço consumidor também é atestado de acordo com seletores próprios de cargas de trabalho SCONE. As próximas seções detalham a atestação de aplicações CNC com Kubernetes e aplicações ConfCNC com SCONE.

3.5.1. Atestação de Cargas Nativas da Nuvem

O SPIRE possui *plugins* que o permitem uma integração com o orquestrador Kubernetes e a atribuição de identidades a partir de atributos desse orquestrador. Atualmente, quatro *plugins* compõem essa integração: dois *plugins* para atestação de nós, um *plugin* para atestação de cargas de trabalho, e um *plugin* para atualização do pacote de confiança.

Há duas formas de atestar agentes implantados em nós Kubernetes. A primeira abordagem, o *plugin k8s_sat*, atesta nós Kubernetes a partir das contas de serviço (*ServiceAccounts* providas pelo Kubernetes aos agentes. O segundo *plugin*, o *k8s_psat*, usa contas serviço projetadas. Para a atestação de cargas de trabalho no Kubernetes, o SPIRE dispõe de um *plugin* de atestação chamado *k8s*. O *plugin k8s* possui uma diversidade de seletores que podem ser utilizados no processo de atestação das cargas de trabalho. Tais

seletores são gerados a partir de consultas ao *kubelet*. A Tabela 3.6 apresenta os seletores e uma breve descrição de cada um. Por fim, o *plugin* para atualização do pacote de confiança (*k8sbundle*) é responsável por atualizar um *ConfigMap* no Kubernetes sempre que o pacote de confiança do servidor for rotacionado.

Seletor	Descrição
k8s:ns	*Namespace* da carga de trabalho
k8s:sa	Conta de serviço da carga de trabalho
k8s:container-image	Etiqueta da imagem de contêiner da carga de trabalho
k8s:container-name	Nome do contêiner da carga de trabalho
k8s:node-name	Nome do nó onde a carga de trabalho executa
k8s:pod-label	Nome de uma etiqueta atribuída ao Pod da carga de trabalho
k8s:pod-owner	Nome do dono do Pod da carga de trabalho
k8s:pod-owner-uid	UID do dono do Pod da carga de trabalho
k8s:pod-uid	UID do Pod da carga de trabalho
k8s:pod-name	Nome do Pod da carga de trabalho
k8s:pod-image	Etiqueta de qualquer imagem de contêiner no Pod da carga de trabalho
k8s:pod-image-count	Número de imagens de contêiner no Pod da carga de trabalho
k8s:pod-init-image	Etiqueta de qualquer imagem de contêiner de inicialização no Pod da carga de trabalho
k8s:pod-init-image-count	Número de imagens de contêineres de inicialização no Pod da carga de trabalho

Tabela 3.6. Seletores dos *plugins* Kubernetes de atestação de carga de trabalho.

Esses seletores podem ser combinados para definir um conjunto de características, dentro do contexto do Kubernetes, que uma carga de trabalho precisa ter para receber seu SVID.

3.5.1.1. Modelo de Ameaça do SPIFFE

O modelo de ameaça do SPIFFE é construído levando em conta três fronteiras de confiança: uma fronteira entre as cargas de trabalho e os agentes, outra fronteira entre os agentes e os servidores, e uma fronteira entre servidores de diferentes domínios de confiança.

A fronteira de confiança entre as cargas de trabalho e os agentes existe pois cargas de trabalho são consideradas, a princípio, não-confiáveis. Dessa maneira, as cargas de trabalho, consideradas como possivelmente comprometidas, só recebem uma identidade após o processo de atestação realizado pelo agente e seus *plugins*.

A fronteira de confiança entre os agentes e servidores existe por uma motivação similar. Na atestação de nó, agentes são considerados não confiáveis e possivelmente comprometidos até que o processo de atestação termine com sucesso. Em outras palavras,

antes da verificação das características de interesse de um nó, o agente não está apto para operar em pleno funcionamento, expondo a API de atestação de cargas de trabalho.

Já a fronteira entre servidores de diferentes domínios de confiança é importante para os cenários onde existe uma federação de servidores. Servidores devem ser capazes de distribuir os pacotes de confiança federados para que as cargas de trabalho consigam verificar identidades de outros domínios de confiança. Contudo, servidores de outros domínios não devem ser capazes de alterar ou gerar identidades pertencentes a outros domínios.

É importante observar que o modelo de ameaça do SPIFFE assume confiança na pilha de *software* e *hardware* utilizada pelos *plugins* (de atestação de cargas de trabalho ou de nós) para a geração dos seletores SPIFFE. Isso significa que diferentes combinações de *plugins* podem gerar diferentes superfícies de ataque.

3.5.2. Atestação de Cargas Confidenciais Nativas da Nuvem

Devido ao modelo de ameaça de aplicações ConfCNC, os *plugins* de atestação de cargas de trabalho disponíveis não são adequados para aplicações SCONE. Uma nova alternativa, que inicialmente foi projetada para cenários onde não é possível implantar agentes SPIRE, como é o caso de aplicações *serverless*, foi utilizada para entrega de identidades para tais aplicações confidenciais. Essa nova abordagem consiste em um novo tipo de *plugin* para agentes SPIRE chamados de *StoreSVID*.

O funcionamento geral de um *plugin StoreSVID* é simples. Quando uma nova identidade é atualizada no agente SPIRE, se essa identidade está marcada como responsabilidade de um *plugin* do tipo *StoreSVID*, esse *plugin* é invocado. Todos os artefatos relacionados à identidade são repassados para o *plugin* que tem a responsabilidade de codificar os artefatos em um formato apropriado e enviá-los para um armazenador seguro, que será a ponta final para entrega da identidade às cargas de trabalho.

Nesse contexto, o Laboratório de Sistemas Distribuídos da UFCG desenvolveu um novo *plugin StoreSVID*, especializado em entrega de identidades para aplicações SCONE, utilizando o SCONE CAS. Dessa maneira, o trabalho de realizar a atestação remota das cargas de trabalho confidenciais é delegado a um CAS. A autorização de acesso às identidades funciona com base em dois seletores SPIFFE: `CAS_SESSION_NAME` e `CAS_SESSION_HASH`. Uma vez que uma carga de trabalho SCONE é atestada, se ela tiver cadastrada na sessão cujo nome e cujo o *hash* correspondam aos seletores, ela receberá a identidade SPIFFE registrada via API de registro do SPIRE.

O fluxo de trabalho do plugin *StoreSVID* para aplicações SCONE é mostrado na Figura 3.12.

No passo 1, o operador registra a carga de trabalho SCONE no CAS, passando as configurações iniciais e parâmetros para a atestação da carga de trabalho no momento da implantação através de uma sessão SCONE. Esse primeiro passo retorna um *hash* para a sessão configurada no CAS. Com esse *hash* em mãos, o operador pode registrar uma entrada para a carga de trabalho no servidor SPIRE, usando os seletores `CAS_SESSION_NAME` e `CAS_SESSION_HASH` (passo 2). Uma vez que uma entrada é registrada no servidor SPIRE, o agente responsável recupera a identidade (passo 3) e

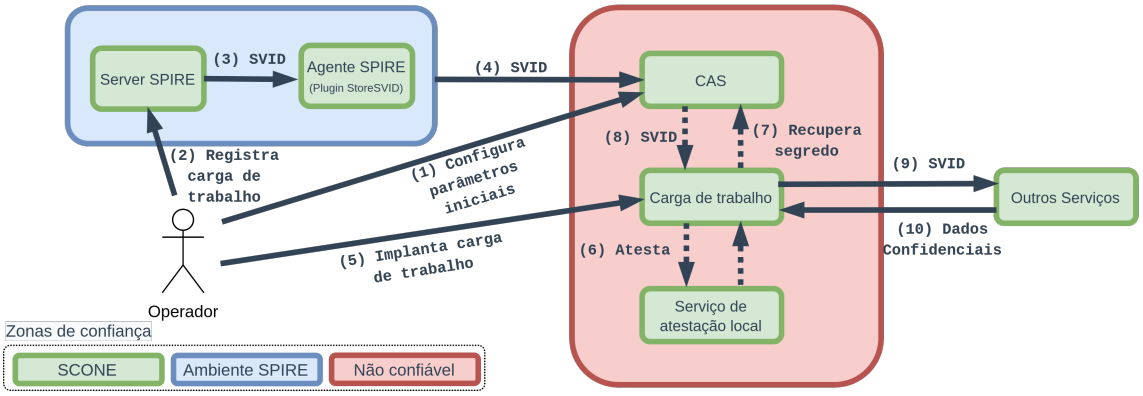


Figura 3.12. Fluxo de trabalho do plugin de atestação SCONE (StoreSVID).

pode então armazená-la no CAS, executando o passo 4.

Após o operador disparar a implantação da carga de trabalho, no passo 5, a carga de trabalho inicia um processo de atestação junto ao serviço de atestação local (passo 6). De posse de um *quote* verificável (como discutido na seção 3.4), a carga de trabalho finaliza o processo de atestação junto ao CAS, no passo 7. No passo 8, mediante a um processo de atestação bem sucedido, o CAS retorna um SVID que corresponde àquela sessão SCONE. A carga pode então utilizar a identidade recebida para se comunicar com outros serviços, receber e enviar dados confidenciais, como ilustrado nos passos 9 e 10.

A próxima seção apresenta os comandos para reproduzir o estudo de caso.

3.5.3. Execução do Estudo de Caso

Para a execução do estudo de caso utilizamos um *cluster* Kubernetes. Informações sobre instalação e configuração do Kubernetes para testes estão disponíveis no repositório de código do minicurso[22].

Os recursos completos, como arquivos de implantação e configuração estão disponíveis no repositório do minicurso, no diretório `demo-produtor-consumidor`. Lá também são encontrados todos os passos para levantar a infraestrutura, incluindo todos os componentes necessários. A seguir são apresentados os passos para registrar e implantar ambos produtor e consumidor.

```
1 git clone https://github.com/ufcg-lsd/minicurso-sbseg-2021
2 cd demo-produtor-consumidor
```

Código 3.10. Baixando o repositório.

Antes de implantar o produtor, é necessário registrar a identidade junto ao servidor SPIRE.

```
1 export AGENTE_ID=spiffe://lsd.ufcg.edu.br/agente-k8s
2 bin/spire-server entry create \
3     -parentID $AGENTE_ID \
4     -spiffeID spiffe://lsd.ufcg.edu.br/produtor \
```

[22]https://github.com/ufcg-lsd/minicurso-sbseg-2021

```
5       -selector k8s:container-name:produtor-kafka
```

Código 3.11. Registrando a identidade do produtor.

Com a identidade registrada, a implantação do produtor é feita utilizando a ferramenta *kubectl*.

```
1 kubectl apply -f produtor/deployment.yaml
2 # verificar o estado da aplicação
3 kubectl get pods
```

Código 3.12. Utilizando kubectl para implantar o produtor.

O comando `kubectl logs <nome-do-pod>` pode ser utilizado para verificar a saída da aplicação.

Para começar o processo de registro do consumidor SCONE é necessário postar a seção SCONE e obter o *hash* correspondente. O registro da seção pode ser feito com o uso do aplicativo de linha do comando do SCONE, o `scone-cli`. Para ter acesso à `scone-cli` basta usar a imagem Docker disponível no repositório do minicurso, como detalhado na seção 3.4. O comando a seguir cria uma seção SCONE a partir do arquivo `session.yaml`.

```
1 # Seção disponível no repositório em consumidor/scone-session.yaml
2 scone session create scone-session.yaml
```

Código 3.13. Utilizando a scone-cli para criar uma seção SCONE.

Após criar uma seção, a `scone-cli` retorna o *hash* dessa seção que pode ser usado para registrar uma identidade no servidor SPIRE. O exemplo a seguir ilustra a criação da identidade do consumidor que executa em um enclave com a ajuda do ambiente SCONE.

```
1  # Registrar entrada SPIRE com um hash e nome de seção
2  export SESSION_HASH=a91ed304958530306f0cab3a2977cbd84e139352ed3c
3                      d2002b6145ee4c4d722f
4  export SESSION_NAME=svid-session
5  export AGENTE_ID=spiffe://lsd.ufcg.edu.br/agente-k8s
6
7  ./bin/spire-server entry create -parentID $AGENTE_ID \
8          -spiffeID spiffe://lsd.ufcg.edu.br/consumidor \
9          -selector svidstore:type:scone_cas_secretsmanager \
10         -selector cas_session_hash:$SESSION_HASH \
11         -selector cas_session_name:$SESSION_NAME
```

Código 3.14. Criação de uma entrada no servidor SPIRE para o consumidor SCONE.

Após criar a entrada, a implantação do consumidor utilizando o comando `kubectl` é similar à implantação do produtor.

```
1 kubectl apply -f consumidor/deployment.yaml
2 # verificar o estado da aplicação
3 kubectl get pods
```

Código 3.15. Utilizando kubectl para implantar o consumidor.

A próxima seção encerra o trabalho apresentando os principais desafios e direções de pesquisa.

3.6. Desafios e Direções de Pesquisa

O padrão SPIFFE juntamente com a ferramenta SPIRE permite que aplicações CNC e ConfCNC se autentiquem de forma transparente, o que facilita a implementação do modelo confiança-zero. Para isto, o SPIFFE resolveu o problema gerado pelo processo de autenticação, que consiste em armazenar informações sensíveis, como um par identificador-senha, substituindo essa abordagem por mecanismos de atestação de nó e carga de trabalho. Com relação à atestação de nós, a maioria dos *plugins* se baseia em evidências que conseguem identificar o nó mas não são robustas para assegurar a integridade do nó, que por sua vez é o ambiente de execução do agente e cargas de trabalho. Além de prover as informações necessárias para emissão do SVID, a atestação poderia ser usada para verificar a integridade e segurança do ambiente de execução. Portanto, futuras pesquisas poderiam envolver a implementação de novos *plugins* de atestação usando diferentes tecnologias, especialmente tecnologias de computação confiável, pois são capazes de coletar informações suficientes para determinar a integridade de um ambiente de execução.

Silva et. al (2021) e Tassyani et. al (2021) conceberam recentemente *plugins* de atestação de carga de trabalho usando tecnologias de computação confiável. Silva et. al (2021) usaram o Intel SGX e arcabouço SCONE para emitir SVIDs para cargas de trabalho com MRENCLAVE específico. Tassyani et. al (2021) criaram um *plugin* de carga de trabalho que usa o chip *Trusted Platform Module* (TPM) como raiz de confiança para medir todos os executáveis e arquivos de configuração importantes carregados no sistema operacional e nos contêineres. As mesmas abordagens também poderiam gerar versões semelhantes dos *plugins* para atestação de nó, possivelmente com pouco esforço. É interessante perceber que identidades geradas a partir de tecnologias de computação confiável carregam consigo uma identificação única da aplicação, que poderia ser utilizada em auditorias. Portanto, estratégia semelhante poderia ser usada para rastrear os dados que nós e cargas de trabalho com identidades dessa natureza consomem, para fins de transparência no uso de dados e aderência à Lei Geral de Proteção de Dados.

Um aspecto do SPIRE que precisa ser aprimorado é a segurança relacionada a diferentes registros de entrada associados ao mesmo SPIFFE ID. Se o administrador (pessoa ou *software*) encarregado por cadastrar esses registros de entrada no servidor for malicioso ele poderá cadastrar um ou mais registros de entrada com o mesmo SPIFFE ID com o objetivo de obter um SVID com um SPIFFE ID específico. Mesmo que originalmente um determinado SVID com SPIFFE ID tivesse associado a um *plugin* que considera seletores rigorosos do ponto de vista de segurança, como por exemplo o *plugin* SCONE usado nesse trabalho, se outro registro de entrada for associado a um *plugin* que entrega SVID para cargas de trabalho atestadas de forma insuficiente, então a mesma identidade também poderia ser emitida para esta última carga de trabalho, mesmo que ela não esteja executando dentro de enclaves SGX. Atualmente, a solução é limitar o acesso à API de registro de identidades. Mas esse é um problema que permite soluções variadas, como permitir que o conjunto de registros de entradas de um servidor SPIRE não contenha duplicatas de SPIFFE ID, ou então cunhar nos SVIDs além do SPIFFE ID o seletor de forma a refletir o *plugin* empregado para coleta de informações no nó ou carga de trabalho.

Outro desafio consiste em portar aplicações legadas para autenticar-se com identidades SPIFFE. A organização e equipe técnica envolvidas precisam ter domínio do padrão SPIFFE e sobretudo do modelo de ameaça SPIRE, de modo que saibam quais são os componentes críticos e possam implantá-los tomando os devidos cuidados. Do ponto de vista técnico, considerando cargas de trabalho que ainda não foram adaptadas para usar a API de Carga de Trabalho do SPIRE, uma alternativa é usar *sidecars* que realizem esta tarefa. Ferramentas como *Ghostunnel* e *Envoy* podem ser usadas para autenticar SVIDs e estabelecer comunicação segura, facilitando portanto a integração de cargas de trabalho já existentes.

Agradecimentos

Este trabalho foi financiado através do projeto ZTPO, uma colaboração entre a Hewlett Packard Enterprise (Brasil), com recursos da Lei de Informática (Lei 8.248 de 23/10/1991), e a unidade CEEI-EMBRAPII na Universidade Federal de Campina Grande.

Referências

[Arnautov et al. 2016] Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P., and Fetzer, C. (2016). SCONE: Secure linux containers with intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 689–703, Savannah, GA. USENIX Association.

[Bellare et al. 1996] Bellare, M., Canetti, R., and Krawczyk, H. (1996). Keying hash functions for message authentication. pages 1–15. Springer-Verlag.

[Brito et al. 2020] Brito, A., Souza, C., Silva, F., Cavalcante, L., and Silva, M. (2020). Processamento confidencial de dados de sensores na nuvem. *Minicursos do XX SBSEG*.

[Costan and Devadas 2016] Costan, V. and Devadas, S. (2016). Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118.

[Cramer et al. 2015] Cramer, R., Damgård, I. B., et al. (2015). *Secure multiparty computation*. Cambridge University Press.

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

[Dobbelaere and Esmaili 2017] Dobbelaere, P. and Esmaili, K. S. (2017). Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper. In *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, DEBS '17, page 227–238, New York, NY, USA. Association for Computing Machinery.

[Dragoni et al. 2017] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham.

[Eugster et al. 2003] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131.

[Feldman et al. 2020] Feldman, D., Fox, E., Gilman, E., Haken, I., Kautz, F., Khan, U., Lambrecht, M., Lum, B., Fayó, A. M., Nesterov, E., Vega, A., and Wardrop, M. (2020). *Solving the Bottom Turtle: a SPIFFE way to establish trust in your infrastructure via universal identity*.

[Gentry 2009] Gentry, C. (2009). *A fully homomorphic encryption scheme*. Stanford university.

[Hayward and Chiang 2015] Hayward, R. and Chiang, C.-C. (2015). Parallelizing fully homomorphic encryption for a cloud environment. *Journal of applied research and technology*, 13(2):245–252.

[Hoffman 2016] Hoffman, K. (2016). *Beyond the Twelve-factor App: Exploring the DNA of Highly Scalable, Resilient Cloud Applications*. O'Reilly Media.

[Khan 2017] Khan, A. (2017). Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Computing*, 4(5):42–48.

[Leiserson 2018] Leiserson, A. (2018). Side channels and runtime encryption solutions with intel® sgx. Whitepaper. Acesso: 27/07/2021.

[Mell and Grance 2011] Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD.

[Naehrig et al. 2011] Naehrig, M., Lauter, K., and Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, page 113–124, New York, NY, USA. Association for Computing Machinery.

[Rose et al. 2020] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero trust architecture.

[Sampaio et al. 2019] Sampaio, L., Souza, C., Vinha, G., and Brito, A. (2019). Asperathos: Running qos-aware sensitive batch applications with intel sgx. In *Anais Estendidos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 89–96, Porto Alegre, RS, Brasil. SBC.

[Severinsen 2017] Severinsen, K. M. (2017). Secure programming with intel sgx and novel applications. Master's thesis.

[Silva et al. 2021] Silva, M. S. L. d. S., Brito, A. E. M., and Brasileiro, F. (2021). Integrating spiffe and scone to enable universal identity support for confidential workloads. Master's thesis.

[Tassyany et al. 2021] Tassyany, M., Sarmento, R., Falcão, E., Gomes, R., and Brito, A. (2021). Um mecanismo de aprovisionamento de identidades para microsserviços baseado na integridade do ambiente de execução. In *Anais do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 714–727, Porto Alegre, RS, Brasil. SBC.

[Tsai et al. 2017] Tsai, C., Porter, D. E., and Vij, M. (2017). Graphene-sgx: A practical library OS for unmodified applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 645–658, Santa Clara, CA. USENIX Association.

[Verma et al. 2015] Verma, A., Pedrosa, L., Korupolu, M. R., Oppenheimer, D., Tune, E., and Wilkes, J. (2015). Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France.

[Wiggins 2017] Wiggins, A. (2017). The twelve-factor app. `https://12factor.net/`. Acesso: 27/07/2021.

Capítulo 4

Segurança em Redes 5G: Oportunidades e Desafios em Detecção de Anomalias e Predição de Tráfego baseadas em Aprendizado de Máquina

Guilherme N. N. Barbosa (UFF), Govinda Mohini G. Bezerra (UFF),
Dianne S. V. de Medeiros (UFF), Martin Andreoni Lopez (TII),
Diogo M. F. Mattos (UFF)

Abstract

This chapter focuses on approaching and contextualizing the security of the fifth-generation (5G) mobile networks, discussing network anomaly detection techniques through hybrid tools. Classical techniques for prediction, such as time series regression analysis and the Hidden Markov Model, are revisited. New anomaly detection and traffic prediction techniques based on deep learning are presented, such as recurrent neural networks, neural networks with long short-term memory, and convolutional neural networks. Finally, the challenges and new paradigms of the next-generation networks (6G) are presented. We also present a case study with a practical exercise to develop an example of anomaly detection and traffic prediction through open source and free tools.

Resumo

Este capítulo aborda e contextualiza a segurança das redes móveis de quinta geração (5G), discutindo técnicas de predição de tráfego e detecção de anomalias em redes através de ferramentas híbridas. Revisitam-se técnicas clássicas para predição de tráfego, como análise de regressão de séries temporais e Modelo Oculto de Markov. Novas técnicas de detecção de anomalia e predição de tráfego baseadas em aprendizado profundo são apresentadas, tais como redes neurais recorrentes, redes neurais com memória longa de curto prazo e redes neurais convolucionais. Por fim, são apresentados os desafios da próxima geração de rede móveis (6G), novos paradigmas e um estudo de caso com exercício prático de desenvolvimento de um exemplo de detecção de anomalia e predição de tráfego através de ferramentas livres de código aberto.

Este capítulo foi realizado com recursos do CNPq, CAPES, RNP, FAPERJ, FAPESP (2018/23062-5) e Prefeitura de Niterói/FEC/UFF (Edital PDPA 2020).

4.1. Introdução

A quinta geração (5G) de sistemas de comunicações moveis é mais do que uma nova geração de tecnologias, mas denota uma nova era em que a conectividade se tornará cada vez mais fluida e flexível. Em 2025, as redes 5G provavelmente cobrirão um terço da população mundial[2], interconectando pessoas, máquinas e dispositivos inteligentes, com mais de 41 bilhões de dispositivos interconectados [Wasicek, 2020]. As redes 5G se adaptam aos aplicativos e o seu desempenho se ajusta precisamente às necessidades do usuário, permitindo a execução de diferentes aplicações como carros autônomos, enxames de drones [Andreoni Lopez et al., 2021], cirurgias remotas, comunicação máquina-a-máquina (*Machine-to-Machine* - M2M), entre outros. Além disso, a rede 5G visa melhorar o desempenho atual das redes móveis, aprimorando a experiencia dos usuários com picos de vazão de 10 Gb/s e latências menores que 1 milissegundo [Wazid et al., 2020]. O maior desempenho e a flexibilidade das redes 5G são devido à implementação do paradigma das redes sem fio definidas por software (*Wireless Software Defined Networs* - WSDN). Algumas soluções implementam as WSDN como o SoftAir, CRAN e CONTENT. Com as WSDNs, é possível realizar fatiamento (*slicing*) das redes [Popovski et al., 2018, Cunha et al., 2019]. Assim, a rede 5G é definida como uma rede orientadas a serviços, que permitem a implantação de novas aplicações com suporte a diferentes requisitos de desempenho. Atualmente, as redes móveis de quinta geração já são uma realidade para 176 redes comerciais no mundo e são foco de investimento de mais de 461 operadoras em 137 países[3]. Para usar essas redes já em operação, são catalogados mais de 600 dispositivos comercialmente disponíveis, mostrando que a tecnologia 5G é uma realidade comercializada e em rápido crescimento. Contudo, a tecnologia 5G impõe desafios para a garantia de privacidade e segurança. Assim, soluções de privacidade e segurança devem ser implantadas em vários níveis, incluindo dispositivos, equipamentos de interface aérea, infraestrutura de rede de acesso de rádio na nuvem (*Cloud - Radio Access Network* - C-RAN), instalações de *backhaul* móveis, entre outros. Para garantir a adequação do nível correto de segurança e privacidade, o 3GPP define a Especificação #: 33.501 para os requisitos de segurança dos sistemas 5G[4]. A organização foca a segurança 5G em autenticação de assinatura, autorização do equipamento do usuário, autorização de acesso e serviço de rede, mas também inclui o usuário e a integridade dos dados de sinalização para garantir a uniformidade e a interoperabilidade entre os elementos da rede.

Os maiores desafios de segurança 5G surgem na camada de aplicação, devido à multiplicidade de aplicações suportadas e à flexibilidade da tecnologia para acomodar novos aplicações. Com largura de banda substancialmente maior e latência ultrabaixa, a rede 5G oferece suporte a muitas aplicações e serviços novos e aprimorados, como realidade virtual não vinculada e telepresença a qualquer hora e em qualquer lugar. Aplicações disruptivos ou tradicionais, como Voz sobre 5G (*Voice over 5G*), são susceptíveis a problemas de segurança comuns, como confidencialidade e privacidade de dados, e a alguns problemas completamente novos, como roubo de identidade virtual ou a extrapolação do consentimento do usuário através da realização de aprendizado sobre seus dados.

[2]Disponível em https://www.gsma.com/futurenetworks/ip_services/understanding-5g/5g-innovation/.

[3]Disponível em https://gsacom.com/paper/5g-market-update-executive-summary-august-2021/.

[4]Disponível em https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169.

4.1.1. As Gerações de Redes Móveis

A primeira geração de comunicações móveis (1G), introduzida em 1979, tinha transmissões com picos de 2,4 Kb/s. A segunda geração (2G), já na década de 1990, permitia o envio de mensagens curtas através da tecnologia *General Packet Radio Service* (GPRS), com velocidades entre 50 Kb/s e 1 Mb/s. Tanto a primeira como a segunda geração careciam de segurança no projeto. A terceira geração (3G), introduzida em 1998, fornecia serviços como internet móvel, navegação web, descarga de imagens com taxas de navegação de até 2 Mb/s. Nas redes 3G, foi introduzida a característica de autenticação mútua, ou de duas vias, para evitar a conexão com estações bases falsas. As redes referenciadas como 3.5G consistem em uma evolução da terceira geração, com a inserção da tecnologia de pacotes de acesso de alta velocidade (*High Speed Packet Access* - HSPA+) com velocidades teóricas de até 22 Mb/s. Em relação às redes 3G, a tecnologia da rede 3.5G provia uma nova rede de acesso, pois o núcleo da rede 3G já contava com comunicação sobre IP. As redes de quarta geração (4G) fornecem maiores velocidades e segurança introduzindo novos serviços sobre IP, como telefonia e TV. Além disso, a inserção de novas tecnologias, como múltiplas entradas e múltiplas saídas (*Multiple-input and multiple-output* - MIMO) para aumentar a capacidade de transmissão, e a multiplexação por divisão de frequências ortogonais (*Orthogonal Frequency Division Multiplexing* - OFDM) para manter altas taxas de transferências, fazem as redes 4G atingirem velocidades de transmissão de dados de até 1 Gb/s. As redes 4G usam protocolos criptográficos avançados para autenticação do usuário e oferecem proteção contra ataques físicos, como adulteração física de estações base, que podem ser instaladas em instalações públicas ou do usuário. No entanto, todas as gerações mencionadas carecem de suporte à manutenção da conexão confiável em mobilidade. A infraestrutura das redes 4G adicionam a banda larga móvel aprimorada (*enhanced Mobile BroadBand* - eMBB), a qual suporta conexões estáveis com taxas de dados de pico altas, bem como taxas moderadas para usuários de celular servindo como entrada para as redes 5G. As principais categorias de casos de uso das redes 5G são a comunicação massiva de tipo de máquina (*massive Machine Type Communication* - mMTC) [Bockelmann et al., 2016], com aplicações tais como o monitoramento de ambientes com grande quantidade de dispositivos e baixas ta-

Tabela 4.1. Resumo das ameaças nas diferentes tecnologias de comunicações móveis da primeira a quarta geração. Adaptado de [Ahmad et al., 2019].

Geração	Mecanismos de Segurança	Desafios de Segurança
1G	Sem medidas explícitas de segurança e privacidade.	Bisbilhotamento, interceptação de chamadas e nenhum mecanismo de privacidade.
2G	Proteção baseada em autenticação, anonimato e criptografia.	Estação base falsa, segurança de link de rádio, autenticação unilateral e spamming.
3G	Adotou a segurança 2G, acesso seguro à rede, Autenticação e Acordo de Chave (AKA) e autenticação bidirecional.	Vulnerabilidades de segurança de tráfego IP, segurança de chaves de criptografia, segurança de roaming.
4G	Nova criptografia (EPS-AKA) e mecanismos de confiança, segurança de chaves de criptografia, segurança de acesso do 3GPP e proteção de integridade.	Maior segurança induzida por tráfego de IP, integridade de dados, segurança de Base Transceiver Stations (BTS) e interceptação de chaves de longo prazo.

xas de transmissão, e as comunicações ultraconfiáveis de baixa latência (*Ultra-Reliable Low-Latency Communications* - URLLCs) [Popovski et al., 2018] que suportam transmissões de baixa latência, como a direção de veículos autônomos que precisam de alta confiabilidade e reação imediata.

4.1.2. As Ameaças de Segurança às Redes 5G

As redes 5G foram projetadas para solucionar muitas das falhas que suas predecessoras tinham, tais como limitações na autenticação de dispositivos, falhas à privacidade do usuário e a vulnerabilidade da interface de rádio. Como muitos operadores estendem a infraestrutura das redes 4G, quase todas as ameaças e requisitos de segurança relacionados às gerações móveis pré-5G ainda são aplicáveis no 5G. Além disso, o 5G terá um novo conjunto de desafios de segurança devido principalmente aos seguintes fatores: maior número de usuários, heterogeneidade de dispositivos conectados, novos serviços de rede, questões de privacidade do usuário e suporte a dispositivos da Internet das Coisas (*Internet of Things* – IoT) e a aplicativos de missão crítica. O software de rede e a utilização de novas tecnologias como redes definidas por software (*Software Defined Networking* – SDN) [Andreoni Lopez et al., 2016], virtualização de funções de rede (*Network Function Virtualization* – NFV) [Andreoni Lopez et al., 2019], computação de borda móvel (*Mobile Edge Computing* - MEC) e fatiamento da rede (*Network Slicing*), apresentarão desafios adicionais a segurança e privacidade [Khan et al., 2019].

Alguns desafios de segurança identificados na literatura [Ahmad et al., 2017a, Ahmad et al., 2018, Wazid et al., 2020] são: o **tráfego de rede repentino** (*flash crowd*), em que existe um grande número de dispositivos de usuário final acessando a infraestrutura ao mesmo tempo; **segurança de interfaces de rádio**, na qual as chaves de criptografia de interface de rádio devem ser enviadas por canais não seguros; **integridade do plano do usuário**, relacionada à proteção de dados do usuário para evitar o vazamento de informações confidencias; **segurança obrigatória na rede**, segurança fim-a-fim de todos os serviços na rede; **segurança de roaming**, pois os parâmetros de segurança do usuário não são atualizados com roaming de uma rede de operadora para outra, levando a compromissos de segurança em redes visitantes; **ataques de negação de serviço (DoS) na infraestrutura**, controle de pacotes para evitar o descontinuidade dos serviços; **tempestades de sinalização**, relacionadas a sistemas de controle distribuído que requerem coordenação, por exemplo o protocolo de *Non-Access Stratum* - (NAS) das redes 3G e 4G ; **Ataques DoS em dispositivos do usuário final**, já que não há medidas de segurança para sistemas operacionais, aplicativos e configuração de dispositivo de usuário dados.

Nas redes 5G, em virtude da conexão à internet dos objetos, ameaças tradicionais também podem ser executadas [Wazid et al., 2020]. **Bisbilhotamento** é uma ameaça passiva que ocorre quando o atacante escuta as mensagens trocadas entre participantes da rede. Outro ataque passivo é a **análise de tráfego**, no qual o atacante intercepta e examina o tráfego da rede para determinar o seu comportamento. O **ataque de repetição** ou *replay attack* é uma forma de ataque em que uma transmissão é repetida de forma maliciosa por um atacante que a interceptou. Esse ataque é semelhante ao **ataque do homen-no-meio**, no qual depois de extrair as mensagens como no ataque de repetição, o atacante modifica as mensagens antes de enviá-las ao destinatário. O ataque do homen-no-meio normalmente é associado ao **ataque de falsificação de identidade**, na qual o atacante modifica

algumas características como endereços IP ou MAC para fingir ser outro membro da rede e assim evitar o princípio de não repúdio. O **ataque de negação de serviço** (*Denial of Service* - DoS) é ainda uma das piores ameaças das rede. No ataque de DoS, o adversário realiza algumas tarefas, como explorar vulnerabilidades de protocolos, para impedir que partes legítimas acessem os recursos da rede ou sistema. Além disso, existe uma variante distribuída (*Distributed Denial of Service* - DDoS), na qual múltiplos atacantes atuam simultaneamente. Esse ataque pode ser realizado em aplicações, como a inundação por HTTP, TCP SYN e UDP, com o objetivo de consumir a maior largura de banda possível, ou na infraestrutura, como os ataques de negação de serviço nos rádios definidos por software [Li et al., 2011]. **Ataques às bases de dados** também são uma ameaça às rede 5G, pois na arquitetura das redes 5G existem diferentes servidores, seja na nuvem, na névoa ou na borda da rede. Múltiplos ataques podem ser executados contra as bases de dados com o objetivo de o atacante extrair informações. Ataques dentro desse grupo são a injeção de SQL, *Cross-Site Scripting* (XSS) e *Cross-Site Request Forgery* (CSRF). **Ataques de malware** permanecem como uma ameaça às novas redes, pois o atacante executa um Software Malicioso (*Malicious Software – Malware*) em um sistema remoto para realizar atividades não autorizadas, como roubo, exclusão, atualização e criptografia de informações importantes. Alguns tipos de *malware* são os cavalos de troia, software espião, *ransomware*, *keylogger* e botnets. As botnets são normalmente utilizadas para o espalhamento do *malware*. Botnets populares que ainda estão em funcionamento são Mirai, Reaper, Echobot e Necurs.

Considerando especificamente as redes 5G, é possível identificar desafios em relação aos casos de uso [Zhang et al., 2019b]. Para os milhares de dispositivos *mMTC* de baixo custo, como sensores, é essencial o uso de algoritmos criptográficos leves e protocolos de gerenciamento de chaves de baixo consumo, de forma a otimizar o uso das baterias de baixa capacidade desse tipo de dispositivo. Os serviços *URLCC*, que requerem muito baixa latência, requerem protocolos rápidos e leves de autenticação forte, assim como algoritmos criptográficos de alta velocidade para atender os requerimentos de baixa latência e alta confiabilidade. Contudo, a restrição de processamento desses dispositivos limita o nível de segurança alcançado pelos protocolos usados.

Uma vez que a redes 5G utilizam tecnologias como a Virtualização de Função de Rede (*Network Function Virtualization* - NFV) e Redes Definidas por Software *Software Defined Network* - SDN, as redes 5G herdam ameaças dessas tecnologias como mostra a Figura 4.1.2. A funções de rede virtualizadas (*Virtual Network Functions* - VNF) são introduzidas no 5G para consolidar várias funções de rede em dispositivos de software, que são executados em uma variedade de hardware padrão da indústria. A dissociação do software e do hardware permite a redução de despesas de capital e operacionais, aumentando a escalabilidade e a resiliência do serviço de rede. Contudo, desafios de segurança do NFV podem vir da infraestrutura de implantação da virtualização (*Network Function Virtualization Infrastructure* - NFVI), do gerenciamento e orquestração do ambiente NFV (*Network Function Virtualization Orchestration* - NFVO) e das interfaces entre o as funções de rede virtualizadas [Lal et al., 2017]. Como ameaças na infraestrutura, o atacante pode criar máquinas virtuais que contêm *malware* para ganhar acesso a outras VMs ou mesmo ao *hypervisor*. Mesmo o *hypervisor* pode ser atacado através da injeção de código malicioso que pode comprometer o controle, assim como as VMs hóspedes. As funções

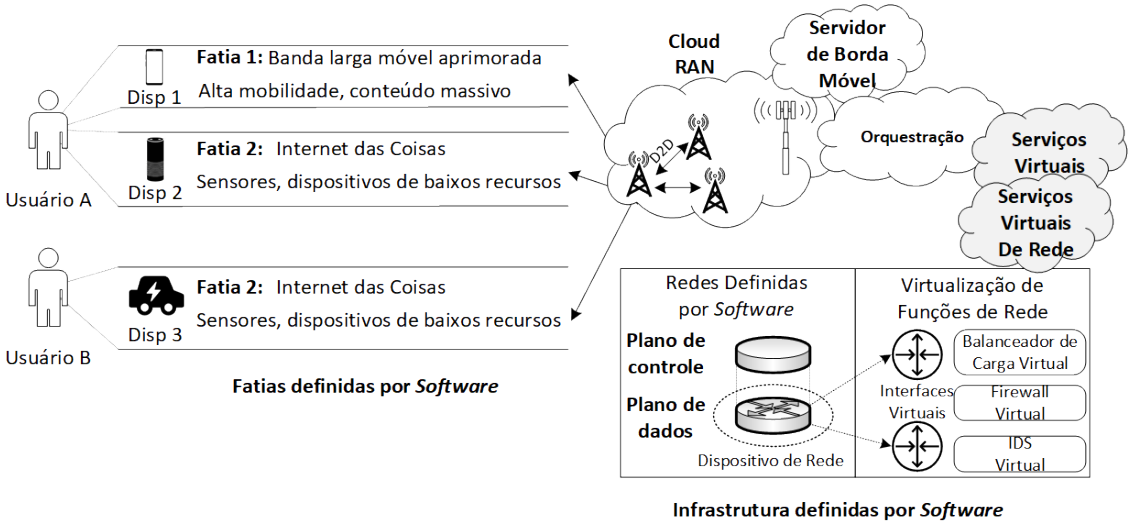


Figura 4.1. Exemplo de arquitetura da rede 5G. Cada usuário tem diferentes equipamentos (*User Equipment* – EU). Para cada UE é possível obter uma fatia de rede com os requerimentos e qualidade de serviço específico. As estações bases são controladas pela rede de acesso de rádio na nuvem (*Cloud Radio Access Network*. Entre os dispositivos pode existir comunicação direta sem passar pelas estações bases. O servidor de borda móvel trará a nuvem para mais perto da rede dos dispositivos. O gerenciamento é realizado por redes definidas por *software* que permitem a criação de serviços virtuais e funções de rede virtualizadas. Adaptado de [Nieto et al., 2019].

de rede virtualizadas (VNFs) são vulneráveis a todo tipo de ameaças de software. Além disso, o atacante pode executar um ataque de negação de serviço contra a VNF para inundar a comunicação. O gerenciamento e a orquestração (NFVO) são um alvo importante já que pode ser considerado como um ponto único de falha. Todas as regras de orquestração devem ser revisadas para ter uma consistência no sistema. As interfaces do NFV devem ser protegidas para evitar a execução de código malicioso, eliminação de portas dos fundos (*backdoors*), prevenção de vazamento de informação. O conceito do fatiamento da rede é introduzido nas redes 5G para fornecer serviços personalizados, permitindo o compartilhamento de recursos entre múltiplos inquilinos sobre uma mesma infraestrutura. No entanto, o isolamento é fundamental entre as fatias de rede [Cunha et al., 2019]. O atacante pode abusar da elasticidade de uma fatia para consumir os recursos de outra fatia.

As redes definidas por *software* (*Software Defined Network* - SDN) simplificam o gerenciamento das redes fornecendo programabilidade por meio do desacoplamento das funções de controle do plano de encaminhamento de dados. O plano de controle é logicamente centralizado para criar políticas de encaminhamento de dados e o plano de dados é distribuído para lidar com o tráfego com base nas políticas de encaminhamento. A centralização lógica das SDN apresenta múltiplas vulnerabilidades [Yao et al., 2019]. As interfaces entre os planos, conhecidas como *northbound*, entre o controlador e as aplicações de rede, e *southbound*, entre os elementos comutadores e o controlador, podem ser utilizadas para atacar outro plano. O plano de controle é especialmente atraente para ataques como negação de serviço devido à sua característica de centralidade como ponto único de falhas. Além disso, outras possíveis ameaças ressaltadas na literatura [Scott-Hayward et al., 2015] são: *acesso não autorizado*, seja ao controlador da

redes ou às aplicações; *vazamento da informação*, como a descoberta de regras de fluxos ou políticas de encaminhamento, credenciais como chaves ou certificados para cada rede lógica; *modificação de dados*, alterar regras de fluxos para modificar pacotes, ataque homem-no-meio; *aplicações maliciosas*, execução de aplicações que permitem a inserção de regras fraudulentas; *negação de serviço*, seja através da inundação de comunicação do controlador-comutadores ou inundação da tabela de fluxo em cada comutador; *segurança do sistema SDN*, os comutadores OpenFlow podem operar no modo a prova de falhas ou falha autônoma, quando o comutador é desconectado do controlador, o atacante pode usar esses modos para atacar o controlador.

Paralelamente, a comunicação dispositivo-a-dispositivo *Device-to-Device*(D2D) apresenta desafios de privacidade em relação a localização. A comunicação D2D exige uma proximidade relativa entre os nós. Isso permite que usuários em conluio executem técnicas para localizar nós móveis próximos [De Ree et al., 2019]. A privacidade do local pode ser garantida usando técnicas de preservação de identidade de autenticação mútua anônima. A introdução de pequenas células móveis define os dispositivos móveis e a infraestrutura de rede. O principal problema das pequenas células móveis reside na falta de uma entidade segura e confiável para estabelecer a segurança durante a implantação da rede. Essa falta de uma entidade confiável apresenta problemas quando se trata de gerenciamento de chaves. Os esquemas de gerenciamento de chaves determinam como as chaves criptográficas são geradas, distribuídas aos nós da rede, autenticadas, atualizadas e revogadas.

4.1.3. Organização do Capítulo

Este capítulo foca nos desafios de segurança relacionados ao grande volume de dados que as redes 5G propiciam. O capítulo aborda técnicas para a previsão de tráfego e detecção de anomalias nas redes. São elencadas técnicas baseadas em inferência estatísticas e técnicas baseadas em aprendizado de máquina. Essas técnicas são essenciais para garantir a segurança das redes 5G, pois com o grande aumento previsto para as comunicações nessa nova geração de rede, o controle e a orquestração das infraestruturas de rede deverão ser mais ágeis e precisos.

O restante do capítulo está organizado da seguinte forma. A Seção 4.2 discute a privacidade dos dados pessoais nas redes 5G. As ferramentas para a predição de tráfego baseadas em modelos estatísticos são apresentadas na Seção 4.3. Por sua vez, as ferramentas baseadas em algoritmos de aprendizado de máquinas são elencadas na Seção 4.4. Os desafios da detecção de anomalias e predição de tráfego para as redes 5G são abordados na Seção 4.5. A Seção 4.6 discorre sobre os desafios futuros para a próxima geração de redes móveis, as redes 6G. Um exemplo prático para detecção de anomalias utilizando redes neurais é mostrado na Seção 4.7. As considerações finais e perspectivas estão na Seção 4.8.

4.2. A Privacidade dos Dados nas Redes 5G

Na era das redes 5G, o desempenho das aplicações está intimamente ligado com a exploração das capacidades dessas redes. O uso ótimo dos recursos disponíveis é alcançado garantindo os requisitos estritos de qualidade de serviço (*Quality of Service* –

QoS), como altas taxas de transmissão, latência ultra baixa e mínima variação de atraso *jitter*. Para tanto é necessária a criação de perfis verticais acurados em termos de uso de recursos, eficiência elástica e capacidade de se adaptar dinamicamente às condições da rede. O uso dos perfis é fundamental para automatizar os processos de produção e desenvolvimento de *software* de automação sobre uma infraestrutura 5G [Zafeiropoulos et al., 2020]. Novas tecnologias precisam ser testadas e combinadas com validações práticas. Zafeiropoulos *et al.* propõem uma metodologia integrada de *benchmarking* e de criação de perfis para aplicações industriais em 5G de forma a facilitar a extração de informações relevantes sobre o sistema testado para, assim, realizar o dimensionamento adequado das aplicações e determinar políticas de operação eficientes [Zafeiropoulos et al., 2020].

As redes 5G possuem grandes dimensões e incluem diversas partes interessadas, como os usuários finais, operadoras, provedores de serviços verticais, empresas e novas tecnologias em conjunto com novos modelos de negócios. Os serviços oferecidos nessas redes contêm informações primárias sobre seus usuários, como identidade, localização ou posição, e outros dados privados. É comum o uso de computação em nuvem por uma parte dos interessados para armazenar, usar e processar informações pessoais dos usuários finais. Os dados pessoais dos usuários finais são processados e compartilhados por diferentes partes interessadas de acordo com os objetivos de cada parte. Como essas informações são armazenadas e podem estar disponíveis às partes interessadas, as redes 5G evocam problemas significativos no vazamento de dados privados, podendo ser uma fonte crítica de violações de privacidade [Khan et al., 2019]. Atender às questões de privacidade para cada parte interessada é uma tarefa complexa devido à natureza paradoxal da tarefa nesse cenário, uma vez que as partes possuem interesses particulares contrastantes envolvidos. A análise do perfil de tráfego de aplicações que executam sobre as redes 5G pode conter diversas informações pessoais sobre seus usuários. A proposta FLOWR (*Flow Recognition*) é um sistema de autoaprendizado que requer um treinamento supervisionado mínimo e detecta automaticamente novas assinaturas de aplicativos contidos no fluxo de rede [Xu et al., 2015]. Para tanto, a proposta foca em aplicações *Web* e define como características do aplicativo a concatenação do nome do serviço *Web* e uma chave-valor do cabeçalho HTTP. Como assinatura do aplicativo, extrai características que identificam o aplicativo com uma boa margem de confiança. A proposta tem como premissa que fluxos em intervalos de tempo próximos e com uma alta probabilidade de ocorrer concomitantemente são oriundos de um mesmo aplicativo. Assim, uma característica que ocorre em um fluxo com um intervalo de tempo T, junto à assinatura de um aplicativo, e que possui uma probabilidade de ocorrência concomitante com a assinatura maior que um limiar p, é promovida a assinatura daquele aplicativo. Para o funcionamento do sistema, é necessário um conhecimento inicial que é extraído de um conjunto de dados.

Soluções de Internet das Coisas (*Internet of Things* - IoT) são uma das principais aplicações das redes 5G. Contudo, os operadores desses ambientes inteligentes não têm o completo conhecimento dos seu inventário de dispositivos IoT. Sivanathan *et al.* focam em identificar dispositivos IoT em uma rede, através da assinatura de cada dispositivo e desenvolvem um arcabouço para classificação de dispositivos IoT usando características de tráfego obtidas a nível de rede [Sivanathan et al., 2019]. Para tanto, são utilizados 28 dispositivos IoT, englobando câmeras, luzes, tomadas, sensores de movimento, eletrodomésticos e monitores de saúde. Os traços (*traces*) de tráfego de todos os dispositivos são

coletados por um período de seis meses. A análise utiliza características estatísticas, aplicando as seguintes métricas para a caracterização dos dispositivos: volume, duração e taxa média do fluxo, tempo de hibernação, número das portas, endereços de consultas DNS, intervalo das consultas NTP (*Network Time Protocol*) e conjuntos de cifras do *handshaking* TLS (*Transport Layer Secutrity*). É realizada uma medida de custo de obtenção de cada uma das variáveis, de acordo com necessidade de processamento dessas medidas. O custo é, então, classificado em custo baixo, médio ou alto. O trabalho propõe também uma métrica de mérito dos atributos, ou seja, o impacto de cada variável no resultado da classificação. Dessa forma, é possível realizar uma escolha de quais variáveis utilizar, de forma a otimizar a implementação em linha (*online*), sem comprometer o desempenho da classificação. A proposta IoTArgos implementa um sistema de monitoramento de segurança multi-camadas, que coleta, analisa e caracteriza dados de comunicação de dispositivos IoT heterogêneos através de roteadores domésticos programáveis [Wan et al., 2020]. O sistema IoTArgos executa em 22 redes domésticas de três países diferentes, constituídas de 20 dispositivos IoT com diversas aplicações. Esses dispositivos coletam dados continuamente por seis meses utilizando roteadores domésticos programáveis que executam o *software* OpenWrt, e usando *dongles* USB para coleta de pacotes ZigBee e Bluetooth. Ao total, são coletados 6 milhões de fluxos considerados normais. Em relação aos dados de ataques, são simulados 19 diferentes ataques a dispositivos IoT em diferentes camadas, gerando 300 mil fluxos atacantes. O sistema extrai dois tipos de características multi-camadas. O primeiro contém informações consideradas com características brutas: endereço IP, nome do domínio do terminal de destino, tempo de chegada entre pacotes, tamanho do pacote, duração do fluxo, portas utilizadas e portas. O segundo é considerado como características avançadas: quantidade de terminais remotos e quantidade de aplicações dominantes. A classificação dos ataques e a detecção de intrusão são realizadas utilizando um modelo de aprendizado de máquina com múltiplos estágios. O primeiro estágio consiste na utilização de algoritmos clássicos de aprendizado de máquinas supervisionado, como K-Vizinhos Mais Próximos (*K-Nearest Neighbors* - KNN), Regressão Logística, Naïve Bayes, Floresta Aleatória (*Random Forest* - RF) e Máquina de Vetor de Suporte (*Support Vector Machine* - SVM), para classificar ataques conhecidos. Os fluxos considerados normais pelo primeiro estágio, ou seja, fluxos legítimos, são inseridos no segundo estágio que utiliza algoritmos de aprendizado não supervisionado para descobrir comportamentos suspeitos ou não usuais, sendo capaz de evitar, assim, ataques desconhecidos e de dia zero (*zero-day attacks*). Os autores propõem o conceito de um módulo de defesa em tempo real, que consiste em um módulo seguinte ao de detecção de intrusão com o propósito de alertar os usuários sobre o ataque detectado, além de desabilitar ou desconectar os dispositivos comprometidos e o seu respectivo concentrador (*hub*), caso necessário. A avaliação experimental demostra que o sistema IoTArgos é capaz de detectar atividades anômalas que visam dispositivos IoT em casas inteligentes com alta precisão.

Li *et al.* demonstram que o aumento da abrangência das câmeras e a integração na vida cotidiana podem resultar em padrões de comportamento e problemas de privacidade [Li et al., 2020]. Os autores realizaram um estudo detalhado, utilizando um grande provedor de câmeras de segurança domésticas (*Home Security Camera* - HSC), cobrindo 15,4 milhões de fluxos e 211 mil usuários. As análises são realizadas através de duas

abordagens: comportamento por usuário e comprometimento de privacidade. Os serviços oferecidos pelos provedores de HSC possuem basicamente dois modos: o modo ao vivo, em que o usuário assiste as imagens captadas pelas câmeras em tempo real, e o modo de reprodução, no qual são realizadas gravações das imagens no servidor a partir de uma detecção de movimento e então os usuários podem assistir as gravações posteriormente. O modo ao vivo está disponível a todos os usuários gratuitamente enquanto o modo de reprodução somente usuários que pagam uma assinatura. Do conjunto total de usuários, 59% dos usuários pagam pela assinatura e correspondem a 95% do total de tráfego, cuja predominância está no tráfego de *upload* de reprodução. Segundo a análise, 60% desse tráfego não é assistido, o que caracteriza um desperdício de recursos de rede e de armazenamento. É ressaltado que os usuários tendem a assistir os vídeos das câmeras em uma ou duas localidades que, normalmente, são diferentes da localização da câmera. Foram identificados três riscos à privacidade dos usuários. O primeiro é o risco de pico de tráfego devido a um aumento vertiginoso no tráfego da câmera que indica que o usuário começou a assistir ao vídeo ao vivo ou houve alguma detecção de movimento que iniciou a gravação das imagens. O trabalho mostra que a aplicação de um classificador simples é capaz de distinguir entre esses dois estados com 100% de acurácia, o que leva ao conhecimento da presença ou não dos usuários nas casas. O segundo risco está relacionado à regularidade de tráfego. O padrão de tráfego das câmeras pode representar o padrão de comportamento dos usuários, revelando as suas rotinas. Os usuários mais suscetíveis a este tipo de ataque são os usuários que pagam pela assinatura e têm altas taxas de *upload*. O terceiro risco está relacionado à mudança da taxa de tráfego, pois indica mudanças nas atividades realizadas pelos usuários. Experimentos com diversas atividades identificaram as diferenças de taxas de tráfego de acordo com as mudanças das atividades.

4.3. As Ferramentas baseadas em Modelos Estatísticos

Nas redes móveis de quinta geração (5G), um dos maiores desafios será gerenciamento de rede devido à sua complexidade. Assim, ITU (*International Telecommunication*) trabalha na atualização das recomendações relativas à qualidade de serviço (QoS) e qualidade de experiência dos usuários (QoE). A recomendação ITU-T Y.3172 prevê a introdução de mecanismos de aprendizado de máquina para o gerenciamento e a orquestração funcionalidades nas próximas gerações de rede[5]. Nesse sentido, modelos estatísticos baseados em séries temporais são os métodos clássicos com bom desempenho sempre escolhidos para realizar predições [Boukerche et al., 2020], sobretudo em fluxos de rede, por possuírem uma boa capacidade analítica e uma implementação com baixo custo computacional. A previsão de séries é um campo essencial do aprendizado de máquina aplicado a redes 5G [Chakraborty et al., 2020]. A modelagem de séries temporais é uma ampla área de pesquisa e vários modelos de previsões de séries temporais evoluíram ao longo do tempo. Nesta seção, são abordados os principais modelos estatísticos para análise de séries temporais utilizando técnicas de regressão como ARIMA (*Auto-Regressive Integrated Moving Average*) e SARIMA (*Seasonal Auto-Regressive Integrated Moving Average*), e o Modelo Oculto de Markov (*Hidden Markov Model* - HMM) que têm como característica a análise de probabilidades entre eventos. As séries temporais são representações matemáticas de fenômenos que ocorrem continuamente durante um intervalo

[5]Disponível em https://www.itu.int/rec/T-REC-Y.3172-201906-I/en.

de tempo. Podem ser divididas em três componentes: tendência, sazonalidade e irregularidade. A tendência relaciona-se a uma perspectiva de longo prazo, a sazonalidade diz respeito a eventos sistemáticos associados ao calendário e, por último, as irregularidades são flutuações não sistemáticas em curta duração [Medeiros et al., 2019]. Existem objetivos basilares para realizar a análise de uma série, sendo eles a cognição do mecanismo gerador da série e a predição de pontos futuros. No que diz respeito ao mecanismo gerador da série, é fundamental identificar o comportamento, isto é, descrever se existem ciclos, tendências ou sazonalidade e pontos de periodicidade relevantes. Com base nisso, a predição do comportamento é possível. Cabe ressaltar que a escolha do melhor método e seus respectivos parâmetros para uma dada série, tem como objetivo reduzir os erros de predição, pois estimar o futuro envolve incertezas.

4.3.1. *Auto-Regressive Integrated Moving Average* – ARIMA

O modelo ARIMA foi inicialmente proposto por George Box e Gwilym Jenkins, sendo também conhecido como método Box-Jenkins. Existem ainda variações do modelo como o VARIMA(*Vector Auto-Regressive Integrated Moving Average*), que é utilizado para múltiplas séries temporais, e o SARIMA (*Seasonal Auto-Regressive Integrated Moving Average*), empregado em casos em que existe uma possível sazonalidade nos pontos da série. Todos esses modelos possuem um ótimo desempenho para análises de curto prazo, enquanto o modelo SARIMA é o que possui melhor capacidade para análises a longo prazo. A estrutura do ARIMA é composta por três coeficientes sendo o primeiro denominado auto-regressivo p, seguido do coeficiente de diferenciação d e por último o coeficiente de médias móveis da série q. O modelo ARIMA [Yang et al., 2021] é dado por:

$$y'_t = \alpha_0 + \sum_{i=1}^{p} \alpha_i y'_{t-i} + \varepsilon_t + \sum_{i=1}^{q} \beta_i \varepsilon_{t-i}, \tag{1}$$

em que o coeficiente α_i refere-se ao termo auto-regressivo da série, β_i é relacionado à média móvel e ε_t diz respeito à parte residual do modelo. As principais etapas para utilização do modelo ARIMA podem ser realizadas em três etapas [Yang et al., 2021]:

1. **Pré-processamento na série**. O pré-processamento pode ser feito através do teste *Augmented Dickey–Fuller* (ADF) para identificar se a série é estacionária. Em caso negativo, são realizadas diferenciações da série, quantas vezes forem necessárias, até obter uma série estacionária. O número de diferenciações é caracterizado através do parâmetro d;

2. **Cálculo dos valores da função de autocorrelação amostral (ACF) e autocorrelação parcial (PACF)**. O cálculo dos valores das funções ACF e PACF é feito para a série estacionária obtida, determinando os parâmetros p e q respectivamente. Para fins de desempenho, esses parâmetros podem ser obtidos através da análise da métrica *Akaike Information Criterion* (AIC), que tem como objetivo mensurar a qualidade relativa de um modelo estatístico;

3. **Teste do modelo e realizar predições**. Por fim, são realizados testes no modelo que apresenta o melhor desempenho e as predições da série são realizadas.

4.3.2. *Seasonal Auto-Regressive Integrated Moving Average* – SARIMA

Uma das principais variações do ARIMA é o modelo SARIMA. Esse modelo tem como objetivo realizar uma análise mais profunda em séries com características predominantes de sazonalidade e periodicidade, podendo ser útil para predições de tráfego de redes sem fio [Sone et al., 2020] e detecção de anomalias em redes [Kromkowski et al., 2019]. Por ser uma variação do modelo ARIMA, o SARIMA pode ser representado por SARIMA$(p,d,q)(P,D,Q)_s$. A primeira parte do modelo, representada pelos parâmetros p, d e q é não sazonal, enquanto a segunda parte é sazonal e constitui o fator de sazonalidade. Os parâmetros P, D e Q representam respectivamente o número dos termos de sazonalidade da parte auto-regressiva, o número de diferenciações sazonais e a parte sazonal de médias móveis. O fator de sazonalidade contribui para analisar características como uso de banda, que tendem a ter comportamento cíclico. Hanbanchong e Piromsopa utilizam o modelo SARIMA para detectar anomalias predizendo o uso de banda através da sazonalidade existente [Hanbanchong e Piromsopa, 2012]. Em diversos outros campos de estudo, em que a série temporal é utilizada como fonte de análise para estabelecer pontos futuros, o modelo SARIMA é amplamente utilizado. Análise de condições climáticas, predição de carga energética e propagação de doenças infecciosas são temas de estudo que frequentemente usam esse modelo.

4.3.3. Modelo Oculto de Markov (*Hidden Markov Model* – HMM)

Processos estocásticos são definidos através de variáveis aleatórias, as quais representam características determinísticas em um intervalo de tempo t. Os processos estocásticos são utilizados para analisar o comportamento de sistemas em que o grau de incerteza é consideravelmente alto. Esses processos podem ser classificados em dois cenários, em relação ao estado e ao tempo, com característica discretas e contínuas para cada um dos cenários. Um processo estocástico é considerado *markoviano* se a probabilidade condicional de um estado futuro depende apenas do estado presente e não dos estados anteriores. Essa probabilidade pode ser descrita como probabilidade de transição e é expressa matematicamente por

$$P(q_{t+i} = S_j | q_t = S_i). \tag{2}$$

A equação 2 representa a probabilidade do estado q_{t+1} ser S_j no momento $t+1$ dado que o estado q_t é igual a S_i no instante t.

Um processo markoviano é classificado como **Cadeia de Markov** se as variáveis aleatórias são definidas em um espaço de estados discreto. A Cadeia de Markov representa sistemas que podem a qualquer instante de tempo t estar em um dado estado S. A mudança entre um estado e outro ocorre através de uma matriz de transição, que descreve as probabilidades de o sistema mudar do estado S_0 para o estado S_n. A Figura 4.3.3 mostra uma Cadeia de Markov com 3 estados e as probabilidades de transição entre esses estados, representadas por $\alpha_{i,j}$.

O conjunto de probabilidades da matriz de transição de uma Cadeia de Markov é caracterizado pela Equação 3 e deve obedecer às propriedades das Equações 4 e 5.

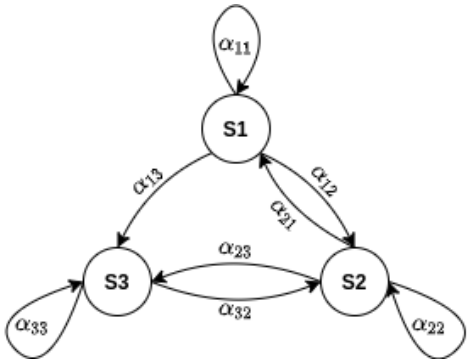


Figura 4.2. Exemplo de representação de uma cadeia de Markov com probabilidades de transição entre três estados. Os estados são representados pelas variáveis s_i e as probabilidades de transição entre estados, pelas variáveis α_{ij}.

$$a_{ij} = P(s_{t+i} = S_j | s_t = S_i) \qquad (3)$$

$$a_{ij} \geq 0 \qquad (4) \qquad \sum_{i=1}^{N} a_{ij} = 1 \qquad (5)$$

O **Modelo Oculto de Markov** (*Hidden Markov Model* - HMM), em sua essência, é a variação de um processo estocástico Markoviano. O HMM é caracterizado por duas componentes, uma não observável e outra observável. A primeira componente representa o estado de um sistema previamente modelado, enquanto a segunda representa as observações já realizadas. Os processos não observáveis representam um conjunto de estados interligados através da matriz de probabilidades, enquanto os processos observáveis representam as saídas de cada estado. Como exemplo, alertas de um sistema de detecção de intrusão (*Intrusion Detection System* - IDS) [Chadza et al., 2020], podem ser caracterizados como um processo estocástico observável em um modelo oculto de Markov, no qual a sequência de observações representam os alertas e a sequência de estados ocultos representam o estado do evento de segurança [Zhan et al., 2020]. O modelo é representado de forma reduzida através de uma tupla com três elementos, (A, B, π), em que A representa a matriz de transição, B a distribuição de probabilidades das observações e π o vetor de probabilidade inicial. Di Bernardino e Brogi mostram que o modelo também pode ser representado com parâmetros adicionais [Di Bernardino e Brogi, 2019], da seguinte forma:

1. Sendo N o número de estados do sistema, o conjunto de estados descritos individualmente é dado por
$$S = \{S_1, S_2, ..., S_N\}; \qquad (6)$$
2. Existe um número M de observações realizadas, cujo conjunto é dado por
$$O = \{O_1, O_2, ..., O_M\}; \qquad (7)$$
3. A transição entre estados é dada pela matriz de transição de probabilidades A que possui dimensão NxN e é definida por $A = [a_{ij}]$, cujos elementos são dados por
$$a_{ij} = P(s_{t+i} = S_j | s_t = S_i), \qquad 1 \leq i, \quad j \leq N; \qquad (8)$$

4. A matriz de probabilidades de observações $B = [b_{ij}]$ possui dimensão NxM e os elementos são descritos através de

$$b_{ij} = P(o_t = O_j | s_t = Si), \qquad 1 \leq i \leq N, \quad 1 \leq j \leq M; \tag{9}$$

5. O vetor de probabilidade inicial é definido por

$$\pi_i = P(s_1 = S_i), \qquad 1 \leq i \leq N. \tag{10}$$

No HMM existem dois tipos principais de estruturas, classificadas como ergóticas, ou sem restrições, e esquerda-direita (*left-right*). No modelo com estrutura ergótica, cada estado pode transitar entre quaisquer outros, sendo esse modelo completamente conectado. O modelo com estrutura esquerda-direita não permite transições entre um estado e estados anteriores [Chadza et al., 2020], sendo mais relevante para detecção de ataques, principalmente os que possuem diversas etapas antes de atingirem o objetivo.

4.3.4. Classificador Bayesiano

A classificação Bayesiana é fundamentada no Teorema de Bayes, no qual as probabilidades de um dado evento estão condicionadas à probabilidade de hipótese com resultados já conhecidos. Seja um conjunto de dados $X = (x_1, y_1)$,...,(X_N, y_n) com x amostras e y classes correlatas para um problema de classificação, sendo $x \in \mathbb{R}$ e $y \in [1, K]$ [Medeiros et al., 2019], o Teorema de Bayes é descrito por:

$$P(y = i|x) = \frac{P(i) * P(x|i)}{P(x)}, \tag{11}$$

em que $p(i)$ a probabilidade de uma hipótese ser verdadeira a partir da amostra de uma classe e $p(y|x)$ a distribuição de probabilidades desconhecidas no espaço amostral x.

O classificador Naïve Bayes tem sua origem no **Teorema de Bayes** e tem como premissa desconsiderar a correlação entre variáveis. É comumente utilizado para dados com alta dimensionalidade [Kumar Dwivedi et al., 2018]. A probabilidade condicional é utilizada para prever ataques e tráfegos regulares, podendo ser utilizada na detecção de ataques em redes definidas por software [Ahmad et al., 2020], por exemplo. Em redes *Ad-Hoc*, o classificador Naïve Bayes pode ser usado como parte de um arcabouço para detecção de ataques de Negação de Serviço Distribuído (*Distributed Denial-of-Service* - DDoS) [Reddy e Thilagam, 2020]. Nesse caso, Reddy e Thilagam utilizam o classificador para dividir o tráfego de rede em dois padrões, normal e ataques DDoS. Para isso, consideram cinco características do tráfego para determinar a qual padrão um fluxo pertence, sendo eles o tamanho do pacote, a porta, o IP de origem, o IP de destino e a variação do atraso (*jitter*). O classificador é utilizado amplamente para classificação de textos, sendo possível detectar cargas úteis anômalas no tráfego de rede [Swarnkar e Hubballi, 2016]. Essa detecção é importante porque uma das principais formas de ataque HTTP ocorre através da modificação da carga útil do pacote.

A tabela 4.2 descreve os modelos estatísticos descritos nesta seção, apresentando as principais características, finalidades, pontos positivos e negativos.

Tabela 4.2. Comparativo entre os principais modelos estatísticos usados para a predição de tráfego em redes 5G.

Modelo	Finalidade	Aplicação	Prós	Contras
ARIMA	Análise de séries temporais	Predição de Tráfego	Análise de curto prazo	Custo computacional
SARIMA	Análise de séries temporais com sazonalidade	Predição de Tráfego	Captura dependência entre dados consecutivos	Custo computacional
HMM	Predição de estados	Detecção de anomalias	Capacidade de capturar dependência temporal	Tempo de treinamento
Classificador Bayesiano	Classificador	Detecção de anomalias	Eficaz para múltiplas classes	Presume independência das características

4.4. As Ferramentas baseadas em Algoritmos de Aprendizado de Máquinas

A rede 5G demanda a implantação de uma infraestrutura mais automatizada e independente da ação humana. Técnicas de aprendizado de máquina têm se tornado amplamente utilizadas em predições de tráfego e detecção de anomalias, principalmente pelo alto desempenho, que compensa o custo computacional exigido por alguns algoritmos. As técnicas de aprendizado recebem dados com o objetivo de obter modelos que descrevam as observações realizadas, permitindo a descoberta de comportamentos até então então desconhecidos. A partir dos modelos, podem ser tomadas decisões sobre tarefas específicas de forma acurada [Zhu et al., 2019]. O núcleo da rede 5G deve ser escalável e para isso implementam-se novas instâncias sob demanda utilizando redes definidas por software (*Software-Defined Networking* - SDN) e virtualização de funções de rede (*Network Function Virtualization* - NFV). Para que isso seja realizado de forma automatizada e utilizando os recursos de rede de forma eficiente, é necessário prever a carga de uso da rede a todo instante. Dessa forma, torna-se possível provisionar corretamente os recursos. Nesse sentido, a utilização de técnicas de aprendizado de máquina é fundamental [Alawe et al., 2018]. Por outro lado, com o aumento de tráfego e da quantidade de dispositivos, a segurança também exige esforços para detectar anomalias e ameaças antecipadamente na rede 5G, principalmente em ambientes com utilização de canal compartilhado e computação na borda, uma vez que brechas de segurança são decorrentes da comunicação com redes abertas, facilitadas pela virtualização de funções de rede [Sedjelmaci, 2021]. A detecção de ataques provenientes de *botnets*, por exemplo, pode ser realizada através de aprendizado de máquina utilizando um sistema de detecção de intrusão (*Intrusion Detection System* - IDS) baseado em redes neurais profundas (*Deep Neural Network* - DNN) [Fernandez Maimo et al., 2018, Lobato et al., 2021].

O aprendizado de máquina pode ser classificado em diversas categorias, sendo as principais as listadas a seguir [Medeiros et al., 2019]:

- **Aprendizado supervisionado**, em que as observações são fornecidas como pares de entrada-saída e o objetivo do algoritmo é identificar uma função que relacione as entradas com as saídas. O algoritmo realiza predições baseadas em amostras com padrões previamente rotulados. O treinamento é mantido até que se encontre um modelo ótimo de precisão e acurácia. Como exemplo, podem-se citar os modelos de máquina de vetor de suporte (*Support Vector Machine* - SVM), Redes Neurais, árvores de decisão, entre outros;
- **Aprendizado não supervisionado**, em que as observações fornecidas para os algoritmos são referentes somente aos dados de entrada e sem rotulação, sendo o objetivo do algoritmo agrupar as entradas em grupos similares denominados agrupamentos (*clusters*). K-Médias (*K-Means*), Floresta de Isolamento (*Isolation Forest*) e Rede de Crença Profunda (*Deep Belief Network*) são exemplos de algoritmos de aprendizado não supervisionado;
- **Aprendizado semi-supervisionado** caracteriza-se como uma interseção entre os modelos supervisionado e não-supervisionado. Os algoritmos são capazes de aprender a partir de um conjunto de dados parcialmente rotulado e generalizam o aprendizado para os demais dados, não rotulados;
- **Aprendizado por reforço**, em que os algoritmos se baseiam em um modelo de recompensas e punições que são oferecidas a partir da interação do modelo com o ambiente. Não há mapeamento direto entre entradas e saídas e os resultados são obtidos a partir da retroalimentação (*feedback loop*) entre o sistema de aprendizado e o ambiente. A cada iteração, as ações disponíveis são apresentadas ao modelo no seu estado atual e, após a mudança de estado, recebe um sinal de reforço que tem o objetivo de instigar um comportamento desejado, ou seja, ações que maximizam a recompensa a longo prazo [Kaelbling et al., 1996]. Exemplos de algoritmos dessa categoria são *Q-Learning*, *Q-Learning* Profundo (*Deep Q-Learning* - DQL) e Estado-Ação-Recompensa-Estado-Ação (*State–Action–Reward–State–Action* - SARSA).

Nas redes 5G, o aumento de tráfego é consequência do número de dispositivos conectados, aumentando a complexidade das redes e gerando uma quantidade significativa de informações para análise em tempo real. Tal efeito induz uma maior dificuldade no processamento e detecção de anomalias. Nesse contexto, o aprendizado profundo (*deep learning*) torna-se uma ferramenta valiosa. O aprendizado profundo é uma área do aprendizado de máquina que tem o objetivo de reconhecer padrões complexos em estrutura de dados a partir de de representações mais simples dessas estruturas. As soluções mais simples são organizadas em uma hierarquia cujos diferentes níveis se complementam para a composição de informações complexas [Goodfellow et al., 2016]. Trinh *et al.*, por exemplo, utilizam um modelo semi-supervisionado com uso do algoritmo memória longa de curto prazo (*Long Short-Term Memory* - LSTM) para detectar atividades legítimas [Trinh et al., 2019]. A detecção é importante sobretudo em áreas metropolitanas, onde podem ocorrer anomalias causadas por aglomerações inesperadas e degradação no serviço de maneira involuntária.

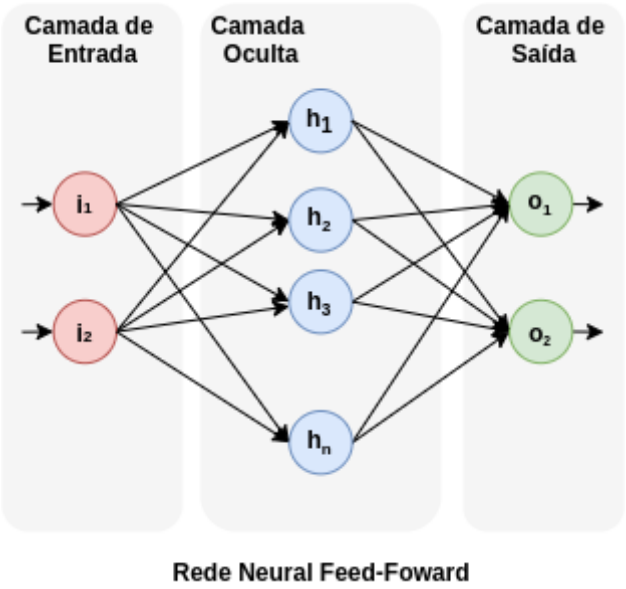


Figura 4.3. Camadas de uma Rede Neural *Feed-Forward* e interação entre os neurônios. A rede neural é formada por três camadas, entrada, oculta e saída. O fluxo de informação percorre a rede da entrada para a saída. Não há retroalimentação entre neurônios.

4.4.1. Redes Neurais Recorrentes (*Recurrent Neural Network* – RNN)

As Redes Neurais *Feed-Forward*, também conhecidas apenas como redes neurais, fazem parte dos primeiros algoritmos que impulsionaram a inteligência artificial. Simulando o comportamento do cérebro humano, tais redes foram utilizadas primordialmente para problemas de classificação e, com o avanço do poder computacional, tornaram-se capazes de trabalhar em diversos campos da ciência. As Redes Neurais *Feed-Forward* possuem a característica de processamento de dados em sequência. A estrutura dessas redes é composta por três camadas, pelas quais o fluxo de informações transita de maneira unidirecional. A informação se move da camada de entrada para a camada oculta e termina nas camadas de saída, conforme ilustrado na Figura 4.4.1. A camada de entrada recebe as informações, representadas na figura por i_1 e i_2, repassando-as para a camada oculta, que aplica uma função matemática específica nos dados da camada anterior para produzir uma saída. Essa função é conhecida como função de transferência, representada na figura por h_k. Por fim, a camada de saída representa o resultado do treinamento da rede neural, representado na figura pelas saídas o_1 e o_2. Por considerarem apenas a entrada atual, essas redes não possuem memória, não sendo viáveis para realizar predições de séries temporais.

As redes neurais recorrentes (*Recurrent Neural Networks* - RNNs) são uma variação de rede neural *feed-forward* que adiciona a capacidade de memorizar os estados passados para processar as próximas sequências de dados, possuindo grande potencial para realizar predições em séries temporais [Jiang e Schotten, 2019]. A Figura 4.4.1 mostra um exemplo genérico de RNN. Observa-se que, diferentemente das redes neurais *feed-forward*, há um relaxamento no sentido de fluxo da informação, que passa a poder fluir através de conexões cíclicas, representadas na figura pelas setas tracejadas. Essas

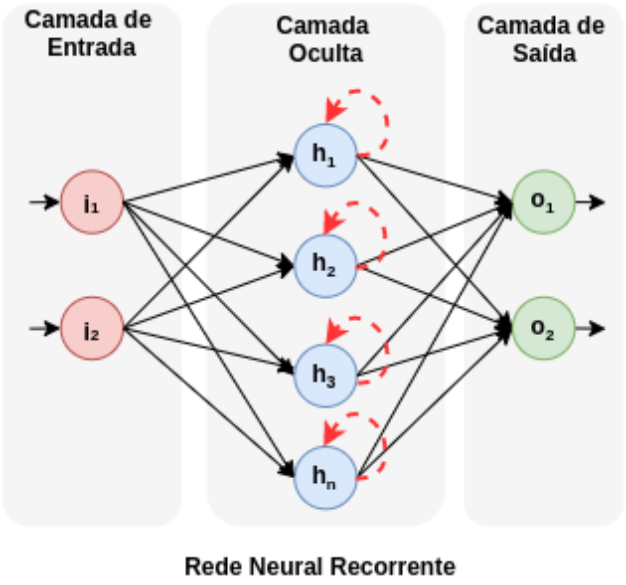


Figura 4.4. Interação entre os neurônios de uma Rede Neural Recorrente. A rede apresenta a capacidade memorizar estados passados e usá-los no processamento dos próximos dados. Há a retroalimentação de informação nos neurônios da camada oculta.

conexões permitem o acesso a estados anteriores, agregando a capacidade de memória à rede. Diversos estudos utilizam as redes neurais recorrentes para predição de tráfego, em virtude de capturar comportamentos mais complexos e não lineares, comparadas aos modelos estatísticos tradicionais, podendo exibir dependências de longo prazo [Ramakrishnan e Soni, 2018]. Ramakrishnan e Soni comparam alguns modelos de redes neurais recorrentes, como o modelo de memória longa de curto prazo (*Long Short-Term Memory* - LSTM) e unidades recorrentes fechada (*Gated Recurrent Units* - GRU) com modelos estatísticos tradicionais, para mensurar o desempenho de cada um na predição do volume de tráfego, de pacotes por protocolo e distribuição de pacotes. A análise mostra que a LSTM possui o melhor desempenho dentre os modelos. As redes móveis 5G produzem dados sequenciais em larga escala [Zhang et al., 2019a], tais como fluxos de tráfego de dados e latência de aplicativos. Dessa forma, é interessante utilizar a RNN para aprimorar a análise de dados de séries temporais em redes móveis.

4.4.2. *Long Short-Term Memory* – LSTM

A rede neural LSTM é uma variação de RNN, porém os nós da rede possuem um estado interno de memória, que pode ser utilizado para armazenar e recuperar informações durante várias iterações. Esse modelo vem sendo amplamente utilizado para modelagem de dados contínuos como processamento de linguagem e predição de séries temporais através de reconhecimento de padrões. Uma célula básica do modelo LSTM é apresentada na Figura 4.5. A estrutura da célula é composta por três portas lógicas denominadas Portão de Esquecimento (*Forget Gate*, Portão de Entrada (*Input Gate*) e Portão de Saída (*Output Gate*). O ***Forget Gate*** é responsável por remover os valores que não são mais elementares no estado da célula. Possui como entrada, dois valores sendo o primeiro h_{t-1} que diz respeito ao valor da célula anterior e x_t, que representa uma entrada em um

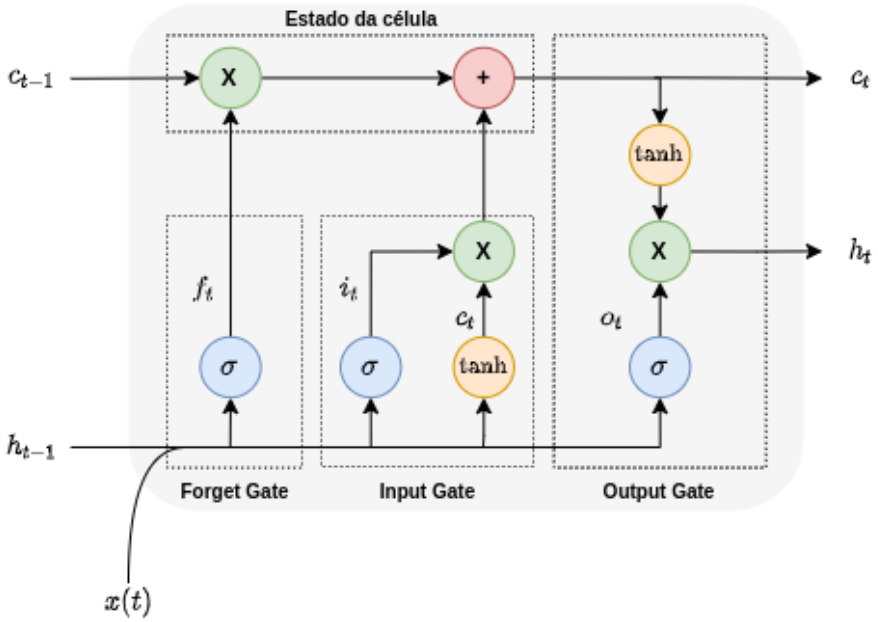


Figura 4.5. Célula básica de uma rede neural de memória longa de curto prazo (LSTM). Os portões de entrada (*input gate*), esquecimento (*forget gate*) e saída (*output gate*) controlam o esquecimento ou o aproveitamento de estados anteriores. A memória interna e novos dados são ativados por funções *sigmoides* e de tangente hiperbólica.

dado instante. Ambos os valores são inseridos em uma função de ativação denominada **sigmoide**, fornecendo uma saída binária. Em seguida, o valor é multiplicado por uma matriz de peso (W_f), e adiciona-se o enviesamento (*bias*) b_f. O enviesamento é um fator de correção para ajustar o modelo. O modelo é representado pela Equação 12.

$$f^{(t)} = \sigma(W_f[h^{(t-1)}, x^{(t)}] + b_f). \tag{12}$$

O ***input gate*** tem como objetivo inserir novas informações relevantes para atualizar o estado da célula. Inicialmente, o estado x_t e o estado oculto h_{t-1} são passados pela função *sigmoide*, tendo como valores de saída 0 e 1, sendo que 0 representa uma informação não relevante, enquanto 1 possui relevância para a memória. Em seguida, x_t e h_{t-1} passam por uma segunda função, a tangente hiperbólica $tanh$, que irá criar um vetor fornecendo valores -1 a 1. Em seguida, o resultado de cada uma das funções é inserido em uma função de multiplicação. O ***output gate*** tem como finalidade definir qual será o próximo estado oculto h_t, responsável por realizar as predições. Inicialmente, os valores h_{t-1} e x_t são passados por uma terceira função *sigmoide* resultando em um valor o_t. Então, o estado da célula é passado por uma função $tanh$. Por fim, são multiplicados ambos os valores para se obter o novo estado oculto h_t, determinando assim os próximos valores com relevância que devem ser transmitidos para a próxima etapa.

Dada a sua capacidade de memória, as redes neurais LSTM são utilizadas para predizer séries temporais. Soluções híbridas com outras redes neurais são utilizadas para otimizar o processamento e predição de tráfego de redes. Huang *et al.* utilizam redes

neurais convolucionais e LSTM para extrair, respectivamente, características geográficas e temporais do tráfego de rede [Huang et al., 2017]. Em redes sem fio, o conceito de Informação do Estado do Canal (*Channel State Information* - CSI) é fundamental para garantir qualidade nas transmissões, em função de características do meio, como interferências e perdas no espaço livre. Como a rede 5G propicia um aumento de dispositivos sem fio, torna-se fundamental realizar predições de tráfego voltadas para a otimização do uso dos canais. Luo *et al.* utilizam redes neurais convolucionais e LSTM para extrair as relações espaciais e temporais para predizer o CSI [Luo et al., 2020].

4.4.3. Redes Neurais Convolucionais

Dentro do contexto de inteligência artificial e aprendizado de máquina, as redes neurais convolucionais (*Convolutional Neural Networks* - CNNs) são um tipo de rede neural profunda (*Deep Neural Network* - DNN) utilizadas de forma mais eficiente com dados de entrada com características multidimensionais, por exemplo, imagens. As CNNs podem ser utilizadas para classificar ou agrupar os dados de saída, de acordo com um grau de similaridade atribuído pelo algoritmo. A influência para criação da rede neural convolucional é originária da estrutura do córtex visual do cérebro humano, que tem como objetivo processar informações visuais. O termo visão computacional sintetiza essas características biológica através de processos e modelagens utilizando, sobretudo, algoritmos capazes de analisar imagens e classificá-las, semelhante ao cérebro humano. A estrutura clássica de uma CNN é composta por cinco camadas, conforme mostra a Figura 4.4.3. As principais camadas são as camadas convolucionais, camadas de agrupamento e camadas densas [Andreoni Lopez e Mattos, 2021]. A **camada convolucional** contém filtros de tamanhos específicos, responsáveis por realizar a operação de convolução dos dados originados na camada de entrada, sendo imagens ou mapa de características, resultando em um novo mapa de características que irá alimentar a próxima camada. Matematicamente, uma imagem ou mapa de características é representado por uma matriz, tendo como componentes n_A, n_L e n_C representando altura, largura e número de canais respectivamente. Para o caso de uma imagem RGB, considera-se $n_C = 3$. Por convenção, considera-se que o filtro K é quadrado com dimensão ímpar, denominado por f, permitindo que cada pixel da imagem seja centralizado no filtro e, assim, considere todos os elementos em sua vizinhança. O produto convolucional entre a imagem e o filtro é uma matriz bidimensional, resultado de uma operação de multiplicação elementar entre o filtro e uma parte da imagem, conforme mostra a Figura 4.4.3 e expressa como:

$$conv(I,K)_{x,y} = \sum_{i=1}^{n_A} \sum_{j=1}^{n_L} \sum_{k=1}^{n_C} K_{i,j,k} I_{x+i-1,y+j-1,k}, \tag{13}$$

em que K é a matriz representando o filtro aplicado à matriz de entrada I para a operação de convolução *conv*.

A **camada de agrupamento** é utilizada após a camada de convolução, com o objetivo de reduzir as amostras das características extraídas da camada de entrada, sem impacto no número de canais, reduzindo dessa forma a redundância de dados [Andreoni Lopez e Mattos, 2021]. Por fim, a **camada densa** tem como finalidade descrever de forma mais detalhada as características extraídas da camada anterior. Uma função de ativação é utilizada para resultar na probabilidade de cada amostra na camada de saída.

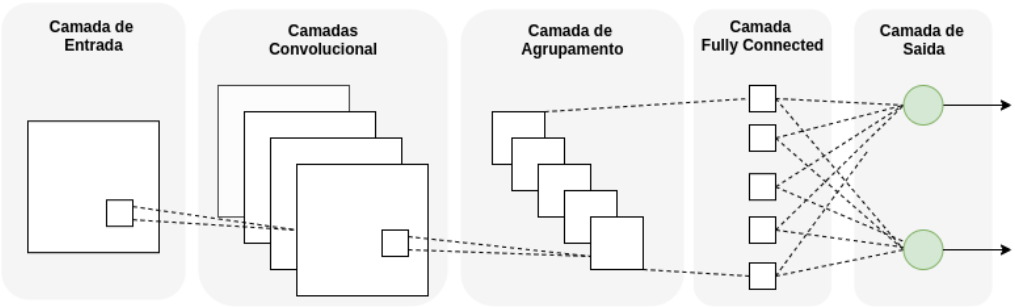


Figura 4.6. Estrutura simplificada de uma rede neural convolucional (*Convolutional Neural Network* - CNN). O aprendizado profundo com CNN é caracterizado pela repetição de camadas convolucional e de agrupamento. A cada par de camadas de convolução e agrupamento são extraídas características de mais alto nível. A camada densa (*Fully Connected*) realiza a classificação através das características extraídas. A consolidação do resultado ocorre camada de saída.

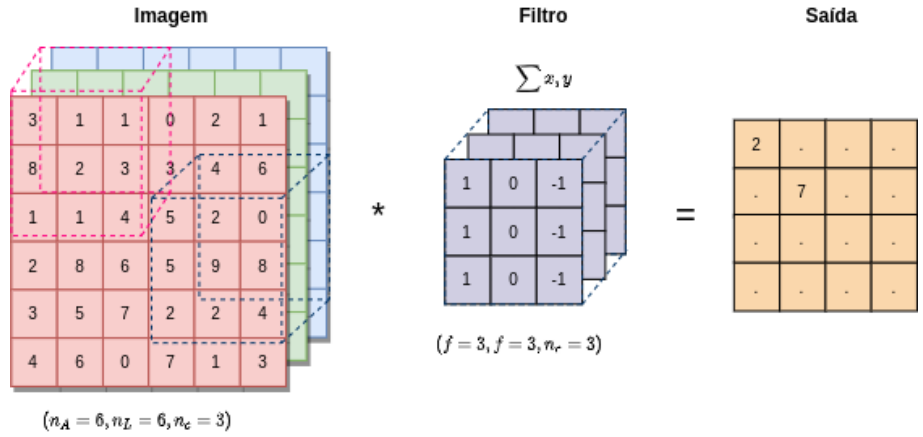


Figura 4.7. Operação convolucional entre imagem e filtro. O resultado da convolução é proveniente da multiplicação matricial entre segmentos da imagem de entrada e os filtros usados.

4.4.4. Codificadores Automáticos (*Autoencoders*)

Os codificadores automáticos, *autoencoders*, são um tipo de algoritmo de aprendizado de máquina não supervisionado utilizado para identificar a codificação dos dados de entrada. Na maioria dos casos, é utilizado como um pré-processamento de outras redes neurais com objetivo de reduzir a dimensionalidade dos dados e consequentemente ignorar ruídos existentes no sinal, podendo aprimorar a entrada de dados de algoritmos supervisionados, como a CNN por exemplo. A estrutura do codificador automático é composta por três camadas: uma camada de entrada, uma camada oculta e uma cada de saída, sendo a camada oculta utilizada como **codificador** e a camada de saída como **decodificador** [Bochie et al., 2020]. O codificador é representado por uma função $f(x)$ que transforma os dados de entrada em uma função h. O decodificador é uma função $g(x)$ que transforma a representação h para um valor reconstruído $\bar{x}$. Os codificadores automáticos são prioritariamente ferramentas para a comprimir dados. Atualmente, duas aplicações

práticas comuns dos codificadores automáticos são a eliminação de ruído de dados, já que levam os dados a uma versão mais compacta, codificada com perdas, e a redução de dimensionalidade para visualização de dados. Com as restrições de dimensionalidade e esparsidade apropriadas, os codificadores automáticos podem aprender projeções de dados que são mais interessantes do que a análise de componentes principais (*Principal Component Analysis* – PCA) ou outras técnicas simples.

Wu, Nekovee e Wang propõem um método de inferência da interferência dinâmica em um canal gaussiano multiusuário baseado em aprendizado profundo e codificadores automáticos [Wu et al., 2020]. A proposta é um mecanismo de codificador automático adaptativo. A intensidade da interferência é prevista por meio de um processo de aprendizado profundo, com a aprendizagem em linha (*online*) em tempo real do conhecimento do nível de interferência. Os resultados mostram que o codificador automático proposto funciona de forma mais robusta em um canal de interferência para todos os níveis de interferências. A melhoria é mais notável para os cenários de interferência forte e muito forte. A proposta estabelece uma base para permitir uma constelação adaptável para sistemas de comunicação 5G, nos quais condições de rede heterogêneas são consideradas.

4.4.5. Aprendizado Federado

O aprendizado federado (*Federated Learning* - FL) é um paradigma de aprendizado que tem como objetivo permitir que dispositivos móveis treinem de maneira colaborativa modelos preditivos compartilhados, mantendo os dados de treinamento localmente [Cunha Neto et al., 2020]. Isso garante principalmente segurança com relação aos dados, pois não é necessário o envio de informações sensíveis, como dados pessoais, para um servidor centralizado, sendo uma das principais técnicas que garantem a privacidade dos usuários. Com o aumento de dispositivos conectados na borda da rede e o aumento significativo de poder computacional por parte dos dispositivos de borda, o aprendizado federado é uma técnica promissora para utilização na rede 5G. O objetivo desse modelo é que cada dispositivo receba o modelo atual de um servidor central e, em seguida, utilize os próprios dados locais para realizar o treinamento local. Como cada cliente realiza um treinamento com dados distintos, são geradas pequenas atualizações locais que são enviadas para o servidor central. Por sua vez, o servidor central garante a agregação das atualizações originárias de todos os clientes, sendo calculada a média entre todos os participantes para melhorar o modelo compartilhado, através do algoritmo de média federada (*Federated Average* - FedAvg) [Cunha Neto et al., 2020]. Após a geração no modelo mais atual, novamente o servidor central envia para os clientes a última atualização. Assim, o aprendizado federado permite a geração de modelos mais eficientes com menor latência, pois é possível a utilização imediata do modelo no próprio dispositivo. O aprendizado federado é fortemente baseado em mecanismos de aprendizado de máquina treinados e otimizados através do método do gradiente descendente estocástico (*Stochastic Gradient Descent* - SGD). As principais implementações do aprendizado federado atuais dependem da ponderação da contribuição local dos diferentes clientes para definir a direção de otimização do modelo global através do algoritmo FedAvg.

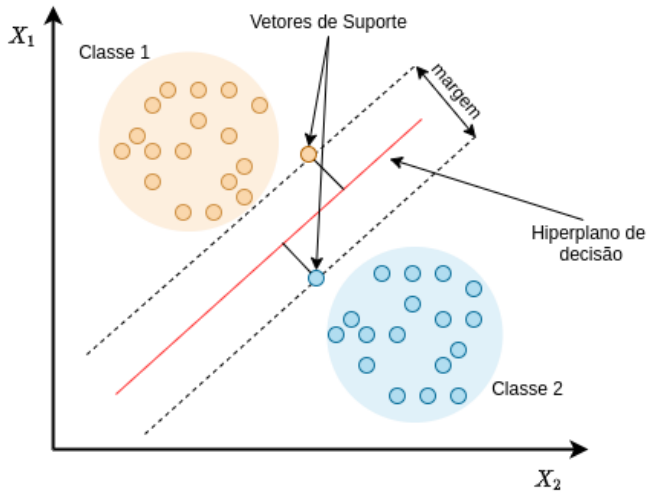


Figura 4.8. Estrutura geral do algoritmo máquina de vetor de suporte (*Support Vector Machine* - SVM). O algoritmo busca amostras de dados, vetores de suporte, que definem o hiperplano de separação entre classes. A seleção dos vetores de suporte visa maximizar a margem de separação entre classes.

4.4.6. Máquina de Vetor de Suporte de Classe Única (*One-Class Support Vector Machine* - OCSVM)

A Máquina de Vetor de Suporte (*Support Vector Machine* – SVM) é uma técnica de aprendizado de máquina supervisionada, podendo ser aplicada para classificação binária ou de múltiplas classes. Nesse último caso, aplica-se uma SVM para cada classe. A técnica é baseada na teoria de aprendizagem estatística e seu principal objetivo é classificar um conjunto de dados através de um espaço multidimensional, definindo um hiperplano de separação entre as classes de tal modo que os dados com as mesmas características estejam agrupados do mesmo lado do hiperplano. A proposta do algoritmo é encontrar os pontos mais próximos das linhas de cada uma das classes, conforme mostra a Figura 4.4.6, sendo esses pontos denominados **vetores de suporte**. Em seguida, calcula-se a distância da **margem**. O SVM tem como objetivo maximizar essa distância, pois assim será encontrado o hiperplano ideal do modelo, de forma a criar um limite de decisão entre as classes. É fundamental que a margem tenha a maior amplitude possível para garantir a maximização da distância entre as classes. O SVM utiliza uma família de funções denominada ***kernel***, que possui diversos tipos, como linear, polinomial, *sigmoide*, dentre outras. O objetivo da função *kernel* é transformar o conjunto de dados, de modo que uma superfície de decisão não linear possa ser transformada em um plano de dimensão superior, fazendo com que a separação entre as classes seja tratada de linearmente.

Uma variação do modelo tradicional é o modelo de classe única da máquina de vetor de suporte (*One-Class Support Vector Machine* - OCSVM), utilizado amplamente para detecção de anomalias. Para aplicação do modelo OCSVM utilizam-se no treinamento dados de uma única classe, ditos normais, ou dados contendo uma pequena fração de amostras anômalas. Com isso, o OCSVM é capaz de detectar amostras fora da classe alvo e amostras com novas características. As amostras que não pertencem à classe alvo estão distantes do hiperplano de decisão e são classificadas como pontos discrepantes

(*outliers*), que nesse caso representam as anomalias. Diversas aplicações utilizam esse modelo, por exemplo, o OCSVM é utilizado para aprender o comportamento normal dos sensores de veículos conectados e automatizados [Wang et al., 2021]. Para detectar variações de anomalias, os autores realizam um processamento nos dados originários dos sensores para mitigar a influência de ruídos utilizando um filtro de Kalman estendido.

A Tabela 4.3 apresenta de forma sintetizada as principais características de cada modelo apresentado nessa seção. Destaca-se que para cada modelo é necessário realizar uma ponderação entre custo computacional e desempenho.

Tabela 4.3. Comparativo entre os principais modelos de aprendizado de máquinas usados para a detecção de anomalias e predição de tráfego em redes 5G.

Modelo	Finalidade	Aplicação	Vantagens	Desvantagens
Codificadores automáticos (*Autoencoders*)	Aprendizado de representações e compactação	Detecção de anomalias	Capacidade em tratar dados sequenciais	Alto custo para grande volume de dados
CNN	Modelagem de dados espaciais	Detecção de anomalias e análise de dados espaciais	Precisão no reconhecimento de padrões	Alto custo computacional e dificuldade na definição de parâmetros
LSTM	Análise de dados sequenciais	Predição de tráfego	Modelagem de dependências de longo prazo	Maior consumo de memória
One-Class SVM	Classificador	Detecção de anomalias	Eficaz em espaços de alta dimensionalidade	Tempo de treinamento superior para grande conjunto de dados
RNN	Análise de dados sequenciais	Predição de tráfego	Capacidade de capturar dependência temporal	Necessidade de um grande conjunto de dados para treinamento

4.5. Os Principais Desafios de Gerenciamento, Segurança e Previsão de Tráfego em Redes 5G

O aumento na comunicação através da rede 5G representa uma disruptura na segurança e privacidade de dados transmitidos, principalmente no que diz respeito à criptografia. Métodos para garantir a segurança da informação, como esteganografia e criptografia caótica são modelos apontados como adequados para aplicações em tempo real [Kakkar, 2020], que serão utilizadas amplamente com a chegada da rede 5G. Wang *et. al.* propõem técnicas de aprendizado profundo para modelar a relação espaço-tempo na predição de redes móveis [Wang et al., 2017]. Para isso, utilizam uma arquitetura baseada em codificadores automáticos (*autoenconder*) e LSTM para avaliar a dependência espaço-tempo, em função da distribuição do tráfego na rede. Os autores utilizam o codificador automático para modelagem e extração das características espaciais e LSTM para modelagem

temporal. Alawe *et al.* investigam a escalabilidade dos recursos do núcleo da rede 5G, composto principalmente por SDN e NFV. O plano de controle no núcleo da rede 5G implementa a Função de Acessibilidade e Mobilidade (*Access and Mobility Function* - AMF) que atua diretamente nas requisições de conexão dos usuários. Assim, a AMF representa um gargalo no plano de controle de redes móveis [Alawe et al., 2018]. São comparados dois modelos de redes neurais, DNN e LSTM, tendo o último um desempenho superior. Nie *et al.* realizam predição de tráfego em redes de malha utilizando Rede de Crença Profunda (*Deep Belief Network* - DBN) [Nie et al., 2017]. Inicialmente, os autores utilizam a transformada discreta *wavelet* para extração das componentes de baixa frequência do tráfego de rede, que representam as dependências de longo prazo, aplicando em seguida a DBN para realizar a predição dessa componente. Já as componentes de altas frequências, representam flutuações irregulares no tráfego e os autores utilizam um modelo gaussiano para caracterizá-las, estimando os parâmetros através da Máxima Verossimilhança.

Uma das principais características da rede 5G é a escalabilidade, seja pela ótica de novos serviços ofertados pelos provedores, quanto por uma maior utilização por parte de consumidores, sensores, dispositivos inteligentes, entre outros. Dessa forma, estender a capacidade da rede torna-se fundamental para garantir essa escalabilidade. Nesse sentido, técnicas como fatiamento da rede são comumente tema de estudos para alcançar a escalabilidade, fornecendo uma maior flexibilidade na administração dos recursos. A utilização de recursos virtualizados e automatizados, fundamentais na rede 5G, são agravantes no ponto de vista de segurança, pois os provedores devem garantir que o fatiamento seja eficaz, evitando que agentes externos possam interromper o serviço e garantindo que o plano de dados continue íntegro. Nesse contexto, Benslimen *et al.* propõem um arcabouço utilizando o modelo ARIMA para predizer ataques, e o modelo LSTM para predizer anomalias e falhas [Benslimen et al., 2021].

Com o avanço das tecnologias, os meios de comunicação sem fio tornaram-se extremamente populares, fazendo com que a tecnologia associada evolua contínua e rapidamente para suportar a comunicação de dados em tempo real com qualidade, como a realização de vídeo chamadas. No entanto, na rede 5G, diversos sensores e dispositivos também são parte fundamental das comunicações, sendo imprescindível estabelecer uma robusta proteção do ponto de vista de infraestrutura, privacidade de usuários e, sobretudo, *software* para esses dispositivos [Zhang et al., 2019a]. Lopez-Martin *et al.* propõem a utilização de codificadores automáticos variacionais condicionais (*Conditional Variational Autoencoders*) para integrar os rótulos de intrusão dentro das camadas de decodificação, permitindo ser utilizado para predição de ataques e reconstrução de informações faltantes [Lopez-Martin et al., 2017]. Por ter apenas uma única etapa de treinamento, o modelo torna-se útil no que tange a otimização de recursos computacionais. Pela sua complexidade, a rede 5G exige o desenvolvimento de arquiteturas e soluções com alta resiliência. Ahmad *et al.* categorizam os principais desafios como [Ahmad et al., 2018]:

- ***flash network traffic***, que representa um aumento significativo de aparelhos e dispositivos conectados à rede, podendo ser contornado através da melhoria dos recursos existentes ou da adição de mais recursos conforme a demanda aumenta, usando redes definidas por *software* ou funções de rede virtualizadas;

- **integridade no plano de usuários**, que também é de vital importância, sendo fundamental a utilização de criptografia ponta a ponta. Aplicações específicas podem demandar a utilização de outras camadas de segurança neste plano, além da comunicação criptografada.

A comunicação móvel possibilita grandes avanços científicos ao longo de suas gerações, pois flexibiliza e universaliza a troca de informações em tempo real, tornando os usuários e dispositivos ubíquos, através de computação móvel, redes de sensores, entre outros. Entretanto, essas características também contribuem para que os ataques aumentem de forma exponencial, porque a superfície de ataque aumenta conforme mais dispositivos são conectados e há possibilidade desses mesmos dispositivos serem vetores de ataques distribuídos, causando prejuízos financeiros, roubo de informações e até mesmo uma guerra eletrônica. O tráfego dentro de uma célula, geralmente apresenta flutuações recorrentes e possui rajadas a qualquer instante, assemelhando-se com o comportamento de pessoas, que possuem características aleatórias no deslocamento ao longo do dia. A análise de segurança das redes tem ganhado foco em diversos campos de pesquisa, sobretudo na detecção de anomalias. Entretanto, a detecção em tempo real torna-se desafiadora em função da quantidade de dados gerados pelos dispositivos, pois requer um monitoramento ininterrupto de eventos, processos e mensagens na infraestrutura [Ariyaluran Habeeb et al., 2019]. As técnicas de detecção de anomalias em tempo real, em função dos desafios previamente listados, utilizam de forma majoritária ferramentas estatísticas por possuírem baixo custo computacional [Ahmad et al., 2017b]. Estudos apresentam os métodos de Holt-Winters [De Assis et al., 2017] e detecção do ponto de mudança (*change point detection*) [Tartakovsky et al., 2013] para detectar anomalias em redes de computadores. Ho Bang *et al.* propõem um modelo utilizando cadeia oculta semi-markoviana (*Hidden Semi-Markovian Model* - HSMM) para detecção de ataques de sinalização na rede LTE (*Long-Term Evolution*) [ho Bang et al., 2017]. Um processo é dito semi-markoviano quando a probabilidade de ocorrer uma mudança de um estado oculto para outro estado depende do tempo decorrido a partir do estado atual. Uma das vantagens da utilização desse modelo é a capacidade de capturar propriedades estatísticas do tráfego de rede e, uma vez utilizando os parâmetros do modelo do HSMM, é possível detectar anomalias de acordo com sua probabilidade ou entropia [Yu, 2010].

Além de técnicas estatísticas clássicas, o aprendizado de máquina também é amplamente utilizado para realizar detecção de anomalias e tráfego. No entanto, para aplicações em tempo real, tais algoritmos precisam de uma sequência ininterrupta de dados, podendo ocasionar um custo computacional superior em função do grande fluxo de informações a serem processadas, podendo limitar em alguns casos a acurácia da predição. A arquitetura da rede 5G tende a ser complexa. Fu *et al.* abordam dois desafios para o gerenciamento do tráfego de rede [Fu et al., 2018]. O primeiro é o fato de a rede ser heterogênea e a simultaneidade de diferentes redes com características distintas tornar a predição de tráfego mais complexa. O segundo é o fato de a rede ser majoritariamente implementada utilizando SDN e NFV, pois, com o fatiamento da rede, os serviços são utilizados de maneira autônoma em infraestruturas compartilhadas e todo o tráfego gerado por cenários distintos é unificado na infraestrutura, tornando o núcleo da rede praticamente imprevisível. Uma das técnicas promissoras é o aprendizado por reforço profundo (*Deep Reinforcement Learning*), que oferece ganhos significativos especialmente em situ-

ações com alta latência e congestionamento da rede, por exemplo. Os métodos baseados no aprendizado por reforço profundo podem aprender informações de roteamento e padrão de tráfego e, assim, gerenciar de forma mais eficiente os recursos da rede [Fu et al., 2018].

A rede 5G tem grande potencial para fomentar a implementação das cidades inteligentes, agregando serviços públicos através de comunicação ubíqua de sensores, dispositivos móveis, câmeras de segurança, entre outros. Além disso, a utilização de celulares inteligentes (*smartphones*) com o sistema operacional Android acrescenta um potencial risco. Sistemas baseados em Android representam 85% do mercado global e, consequentemente, se tornam alvos de ataques massivos [Lu et al., 2021]. É possível realizar a instalação de aplicativos de terceiros, aumentando consideravelmente a chance de implementar *botnets* ou vazamento de informações pessoais.

Do ponto de vista do gerenciamento da interface de transmissão aérea, a estimação das informações de estado do canal (*Channel State Information* - CSI), que representa as propriedades do canal de rádio, continua a ser um dos desafios fundamentais das redes 5G. O estado do canal, representado pelo CSI, tem um impacto significativo na alocação de recursos de rádio e gerenciamento de interferência, sendo utilizado para a determinação dos parâmetros da camada física do enlace. Devido à impossibilidade de envio frequente do valor do CSI pelo receptor, é fundamental que o transceptor seja capaz de estimar acuradamente o valor de CSI para permitir uma comunicação efetiva e otimizada no enlace. Métodos tradicionais para estimação do CSI possuem alta complexidade computacional e não são adequados para o uso em 5G devido ao emprego de tecnologias que aumentam o tráfego de dispositivos móveis, como MIMO (*Multiple Input Multiple Output*) massivo e ondas milimétricas. Luo *et al.* propõem um algoritmo de predição de CSI, denominado OCEAN, baseado em dados históricos de comunicação 5G [Luo et al., 2020]. Primeiramente, são identificadas diversas características importantes que afetam o CSI em um enlace de rádio, como faixa de frequência, localização, horário, temperatura, umidade do ar e tempo. Então, considerando a relação espaço-temporal do CSI, os autores projetam um arcabouço de aprendizado que consiste em uma combinação de duas redes neurais convolucionais (CNN) e uma memória longa de curto prazo (LSTM). A arquitetura do sistema proposto consiste em duas etapas de treinamento, uma *offline* e outra *online*. A etapa *offline* é responsável por treinar a rede com dados históricos. A etapa *online* ocorre com o sistema implementado, em funcionamento. Nessa etapa, a cada 5 minutos uma atualização do valor de CSI medido é utilizada para treinar novamente a rede. Dessa forma, utilizando a retroalimentação *online*, os valores previstos estão sempre sendo corrigidos e se adaptando às mudanças reais sofridas pelo canal, proporcionando resultados mais estáveis nas aplicações em sistemas de comunicação 5G.

4.6. Desafios Futuros para a Rede de Próxima Geração (6G)

O desenvolvimento de novas tecnologias surge para aprimorar as gerações antecessoras. A tecnologia desenvolvida para rede 5G contribui de forma significativa para diminuição da latência das redes móveis através da utilização de novas faixas de frequências com comprimento de onda milimétricas, utilização inteligente do espectro e redefinição do núcleo da rede [Giordani et al., 2020]. No entanto, mesmo antes de sua completa implementação, a rede possui limitações e a sua sucessora já vem sendo amplamente es-

tudada para resolver problemas principalmente de automação e inteligência artificial, com uma quantidade de dispositivos conectados ainda maior e novos conceitos.

A próxima geração das redes de telecomunicações deverá comportar uma quantidade crescente de terminais inteligentes, tais como celulares e sensores, disponibilizar aplicações de tempo real e prover inteligência e confiança embarcadas na infraestrutura de rede. Para atender a esses requisitos, a sexta geração das redes móveis, 6G, vislumbra o uso de novas tecnologias de inteligência artificial, de cadeia de blocos (*blockchain)* e de fornecimento de serviços para Internet das Coisas. A rede 6G está sendo desenvolvida para contornar as limitações existentes atualmente e possibilitar a utilização de novos paradigmas, como novas interações homem-homem e homem-máquina, utilização de frequências na faixa de terahertz, redes tridimensionais, comunicações quânticas, superfícies refletoras inteligentes, entre outras [Chowdhury et al., 2020]. Diversos campos de estudo ainda precisam de aprimoramento para o desenvolvimento e implementação da rede 6G. Estima-se que as taxas de transmissão serão da ordem de 1 Tb/s superando em cinquenta vezes a capacidade da rede 5G [Dogra et al., 2021]. Esta seção apresenta alguns conceitos com foco na futura geração de redes móveis e seus principais desafios no campo da segurança.

As aplicações 6G futuras apresentarão requisitos rigorosos e exigirão recursos de rede estendidos em comparação com as redes 5G desenvolvidas atualmente [Alwis et al., 2021]. Na rede 6G, todos os dispositivos de ponta são concebidos para se conectarem à Internet e os aplicativos de inteligência artificial serão amplamente usados por esses dispositivos. A maioria das aplicações de inteligência artificial são orientadas a dados, aumentando a preocupação com a segurança e privacidade dos dados coletados [Sun et al., 2020]. A privacidade dos clientes pode ser comprometida caso haja o vazamento de dados ou o comprometimento dos modelos de aprendizado [Cunha Neto et al., 2020].

A principal característica da rede 6G é uma conectividade ainda maior através do conceito de *Internet of Everything* (IoE), sendo uma integração entre sensores, dispositivos e qualquer objeto conectado. A IoE pode ser considerada uma extensão da Internet das Coisas abrangendo dados, processos, pessoas e dispositivos [Chowdhury et al., 2020]. O uso de inteligência artificial e questões relacionadas à privacidade de dados são temas em constante ascensão, sendo cada vez mais necessário que questões como a mitigação de vazamento de informações estejam presentes nas arquiteturas do núcleo das redes. As redes veiculares, por exemplo, possuem serviços que precisam de dados transmitidos em tempo real que necessitam de latência praticamente nula, o que não é oferecido com as tecnologias atuais. Com as redes 6G, latências menores que 1 ms serão possíveis e, com isso, a capacidade de controlar remotamente veículos poderá ser alcançada. A segurança de redes veiculares é foco de diversos estudos. Os ataques, como em qualquer outra infraestrutura interconectada, possuem os mesmos propósitos, como ganho de informação e degradação do serviço. No caso de redes veiculares, os ataques podem ser classificados em quatro grupos [Hasrouny et al., 2017]: i) os que representam risco para a interface de comunicação; ii) os que apresentam risco para *software* e *hardware*; iii) os que apresentam riscos para os sensores e iv) os que representam um risco para a infraestrutura. Por ser uma rede altamente dinâmica, os algoritmos de aprendizado de máquina são promissores para realizar detecção em sistemas de detecção de intrusão [Tang et al., 2020].

A utilização da rede 6G tem como objetivo aprimorar a fidelidade nas comunicações, tendo como desafios estabelecer comunicações ultra confiáveis e de baixa latência (*Ultra-Reliable Low Latency Communications* - URLLC). Esses conceitos permitem contribuir de maneira significativa em diversas áreas de missão crítica, permitindo por exemplo, que a **comunicação tátil** seja implementada para garantir que as interações físicas em tempo real sejam executadas, como a teleoperação. Outro conceito que ganhará notoriedade e demandará uma infraestrutura robusta é a holografia. A holografia é uma técnica que utiliza artifícios óticos para projetar luz e fornecer imagem em três dimensões, sendo objeto de estudos principalmente para a telemedicina, provendo atendimento médico em áreas remotas ou e realização de procedimentos cirúrgicos [Giordani et al., 2020].

Os mecanismos de inteligência artificial escaláveis e distribuídos são parte fundamental das redes 6G para proverem o conceito de Auto-X (autoconfiguração, automonitoramento, autocura e auto-otimização) sem qualquer envolvimento humano [Porambage et al., 2021]. Há esforços contínuos de especificação para integrar nativamente elementos de inteligência artificial em redes futuras, envolvendo operação em laço fechado e técnicas de automação por lógica difusa de operações de gerenciamento de rede, incluindo a segurança [Porambage et al., 2021]. Nas redes 6G, mecanismos de inteligência artificial serão instanciados mais próximos da fonte de dados de interesse para garantir a latência ultrabaixa, enquanto funções de aprendizado de máquina serão distribuídas pela rede para obter ganhos de desempenho devido a modelos otimizados e tomada de decisão em conjunto. No entanto, restrições práticas de elementos de rede, como restrições computacionais e conectividade intermitente, constituem um desafio em aberto. Existem também desafios complexos para o uso generalizado de técnicas de aprendizado de máquina aplicadas à cibersegurança, como facilitador da cibersegurança ou como uma técnica que pode levar a problemas de segurança em certas circunstâncias. Problemas de segurança relacionados ao uso massivo de técnicas de aprendizado de máquina estão relacionados:

- **a confiabilidade**, já que fomentam uma grande dependência desses mecanismos em redes futuras;
- **a visibilidade**, pois há uma necessidade de uma visão clara e inteligível dos esquemas baseados em aprendizado de máquina, mas normalmente esses esquemas operam como um esquema oculto para o utilizador;
- **a ética e a responsabilidade**, pois a justiça na aplicação de regras e ações previstas pelos mecanismos de aprendizado, assim como o gerenciamento de responsabilidades com entidades autônomas que operam em um ambiente de tecnologia da informação e da comunicação são tarefas complicadas, incluindo operações de segurança 6G;
- **a escalabilidade e a viabilidade**, já que para configurações distribuídas de aprendizado de máquina, como o aprendizado federado, as transmissões de dados devem ser protegidas e a privacidade preservada. A escalabilidade é um desafio em termos de recursos de computação, comunicação e armazenamento necessários;
- **os modelos de resiliência de dados**, que devem ser protegidos e robustos nas fases de aprendizagem e inferência. Uma das principais soluções previstas para as redes

6G é o uso de cadeia de blocos para uma estrutura de compartilhamento de dados distribuída, transparente e segura;

- **a privacidade**, pois diferentes técnicas de aprendizado podem ser aplicadas para a recuperação e correlação de dados, infringindo a privacidade dos usuários.

As tecnologias de livros-razão distribuídos (*Distributed Ledger Technology* – DLT), como a tecnologia de cadeia de blocos, são apontadas como viabilizadoras das redes 6G. As vantagens adicionais das DLTs como desintermediação, imutabilidade, não-repúdio, prova de procedência, integridade e pseudonimato são particularmente importantes para habilitar diferentes serviços em redes 6G com confiança e segurança distribuídas. Como tecnologias de análise de dados podem ser uma fonte para novos vetores de ataque, como ataques de envenenamento na fase de treinamento e ataques de evasão na fase de teste, as DLTs têm o potencial de proteger a integridade dos dados por meio de registros imutáveis e confiança distribuída entre diferentes partes interessadas, permitindo a confiança em sistemas multi-domínios. Contratos inteligentes baseados em DLT podem ser utilizados para definir Acordos de Nível de Confiança (*Trust Level Agreements* - TLAs) [Varalakshmi e Judgi, 2017] e a responsabilidade de cada parte ou entre componentes, em caso de violações do TLA. A DLT pode ser usada em gerenciamento seguro de funções de rede virtuais (*Virtual Network Functions* - VNFs), corretagem de fatia segura, gerenciamento de nível de serviço de segurança automatizado, gerenciamento de IoT escalonável, *roaming* seguro e manipulação de *offloading* [Camilo et al., 2020]. As cadeias de blocos também são candidatas-chaves para a preservação da privacidade em redes 6G centradas em conteúdo.

Questões relativas à segurança, sigilo e privacidade de dados terão grande foco nas redes 6G. Atualmente, com o aumento do poder computacional, diversos atacantes utilizam processamento gráfico, por exemplo, para otimizar as operações e realizar ataques de força bruta, invadindo serviços públicos e coletando informações sensíveis e pessoais. Atualmente, ainda não é possível identificar e mitigar de maneira antecipada tais eventos com precisão, pois as técnicas de ataques são cada vez mais sofisticadas. Como será o principal meio de comunicação entre os serviços inteligentes, a rede 6G deve garantir sobretudo a privacidade de dados. Será necessário que a inteligência artificial consiga identificar anomalias em tempo real em redes críticas, por exemplo, redes hospitalares em que milhares de dispositivos serão de alguma forma conectados enviando informações sensíveis. No entanto, ainda existem limitações pois os algoritmos possuem foco no desempenho e não em questões éticas. A criptografia é recomendada para qualquer serviço, no entanto na rede 6G será mandatória. A computação quântica eventualmente se tornará acessível a consumidores, o que trará grande impacto nos atuais algoritmos de criptografia. Com isso, estudos já estão sendo realizados para desenvolver a **criptografia pós-quântica** para evitar ataques a partir de computadores quânticos.

A computação quântica foi projetada para uso em redes de comunicação 6G para detecção, mitigação e prevenção de vulnerabilidades de segurança. A comunicação assistida por computação quântica é uma nova área de pesquisa que investiga as possibilidades de substituir os canais quânticos por canais de comunicação clássicos sem ruído para atingir uma confiabilidade extremamente alta em 6G. Com os avanços da computação quântica, prevê-se que a criptografia quântica segura seja introduzida no mundo

pós-quântico. O problema do logaritmo discreto, que é a base da criptografia assimétrica atual, pode se tornar solucionável em tempo polinomial com o desenvolvimento de algoritmos quânticos. Uma vez que a computação quântica tende a usar a natureza quântica da informação, ela pode fornecer intrinsecamente aleatoriedade absoluta e segurança para melhorar a qualidade da transmissão [Porambage et al., 2021].

4.7. Exemplo Prático para Detecção de Anomalias

Nesta seção, são abordados dois cenários práticos utilizando códigos escritos na linguagem Python. Essa linguagem é escolhida pela facilidade na implementação através de diversas bibliotecas já existentes para aprendizado de máquina, manipulação de grandes conjunto de dados, além de modelos estatísticos, mantendo o código sintetizado e eficiente. No primeiro exemplo, é realizada a predição de tráfego utilizando a rede neural de memória longa de curto prazo (LSTM). No segundo, é apresentado o algoritmo SVM de uma classe amplamente utilizado para detecção de anomalias em fluxos de redes. Por último, o código apresentado passa a focar na predição de séries temporais utilizando os modelos ARIMA e SARIMA. No primeiro exemplo, utilizando a rede neural LSTM, são utilizadas métricas coletadas a partir do tráfego da rede sem fio da Universidade Federal Fluminense. Os dados são processados e armazenados em arquivos distintos no formato `CSV` (*Comma Separated Value*) para facilitar o treinamento. Esse arquivo contém o cálculo da entropia de Shannon para as características da 5-tuplas do fluxo TCP (IP de Origem, IP de Destino, Porta de Origem, Porta de Destino e Protocolo) durante seis dias com janela de 1 hora [Reis et al., 2020]. Após isso, aplica-se a transformada discreta *wavelet* para separação das componentes lineares e não lineares, sendo a última a utilizada nesse arquivo. Para o segundo exemplo, foi utilizado um conjunto de dados reais rotulado com alguns tipos de ataques a clientes de uma rede de banda larga residencial [Andreoni Lopez e Mattos, 2021]. Como a proposta do classificador é utilizar apenas uma classe alvo, todos os fluxos que possuem qualquer tipo de ataque são considerados anormais enquanto os demais são ditos normais.

A primeira etapa para execução do código é realizar a instalação das bibliotecas necessárias. Entretanto, existem outras ferramentas que permitem realizar a programação diretamente no navegador, como Google Colab[6]. O Colab é uma plataforma da Google que permite a execução de códigos escritos em Python, assim como outras linguagens, sem a necessidade de configuração ou instalação de bibliotecas no computador. O código-fonte desse exemplo está disponível no repositório no Github[7]. Para realizar a execução, inicialmente são instaladas e importadas as seguintes bibliotecas: Sci-kit Learn, TensorFlow, Numpy e Pandas, como mostrado no Código Fonte 4.1. O TensorFlow [8] e o Scikit-learn [9] são bibliotecas que contêm diversos algoritmos de aprendizado de máquina. O NumPy [10] é uma biblioteca utilizada para funções matemáticas em geral, como álgebra linear, transformada de Fourier, entre outras.

Após carregadas todas as bibliotecas necessárias, o conjunto de dados é carregado

[6]Disponível em https://colab.research.google.com.
[7]Disponível em https://github.com/gnnbarbosa/lstmPrediction.
[8]Disponível em https://www.tensorflow.org/.
[9]Disponível em https://scikit-learn.org/.
[10]Disponível em https://numpy.org.

Código Fonte 4.1. Importação das bibliotecas para execução do exemplo.

```
1  import numpy
2  import time
3  import matplotlib.pyplot as plt
4  from pandas import read_csv
5  import math
6  from tensorflow.keras.models import Sequential
7  from tensorflow.keras.layers import Dense
8  from tensorflow.keras.layers import LSTM
9  from sklearn.preprocessing import MinMaxScaler
10 from sklearn.metrics import mean_squared_error
```

na memória utilizando a biblioteca Pandas [11]. Em seguida, são realizadas etapas de pré-processamento nos dados para otimização do treinamento. O Código Fonte 4.2 monstra a implementação dessa etapa. Dependendo da análise a ser realizada, essa etapa é essencial, principalmente para otimizar recursos computacionais através da redução de dados carregados em memória. O modelo LSTM, em particular, possui certa sensibilidade na escala dos dados, sendo uma boa prática realizar a normalização sem que haja perda na informação. Na maioria dos casos, na normalização, é utilizado o intervalo entre 0 e 1. A função **MinMaxScaler** da biblioteca **Sci-kit Learn** executa a etapa de normalização dos dados. Na próxima etapa, o conjunto de dados é dividido entre treinamento e validação. Nesse exemplo, é utilizado 70% do conjunto de dados para realizar o treinamento e os 30% restantes são utilizados para validação. Uma função *create_dataset* é utilizada para criação de um novo conjunto de dados. Essa função possui dois parâmetros, sendo o primeiro referente aos valores originais da matriz e o segundo, *look_back*, representa o número de entradas anteriores. Assim, será formado um novo conjunto de dados no qual X representa o valor da entropia em um determinado tempo t e Y o valor da entropia no instante $t+1$.

Realizado o pré-processamento do conjunto de dados, são configurados os parâmetros da rede neural LSTM. Nesse exemplo, o modelo possui uma camada oculta de entrada, quatro neurônios e a camada de saída que realiza a predição de apenas um único valor. A rede é treinada em 100 épocas, tamanho do lote igual a 1 e utilização da função de ativação padrão (***sigmoide***). O Código Fonte 4.3 apresenta os parâmetros utilizados no treinamento. Nesse caso, a **função de perda** utilizada é o erro médio quadrático (***mean squared error***). Essa função é fundamental para calcular o custo que o modelo deve minimizar durante o treinamento.

Após o treinamento do modelo através da função ***model.fit*** é feita a predição, conforme apresenta o Código Fonte 4.4. Nessa etapa, é necessário mensurar o desempenho da rede neural treinada. É aplicada uma transformação inversa às predições, antes de calcular as taxas de erros, para recuperar o conjunto de dados original. Destaca-se que até o momento anterior à aplicação da transformada inversa, os dados estão transformados pelos métodos de pré-processamento.

[11] Disponível em https://pandas.pydata.org/.

Código Fonte 4.2. Processamento do conjunto de dados.

```
# Convertendo os valores em uma matriz
def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
                a = dataset[i:(i+look_back), 0]
                dataX.append(a)
                dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)

# Carregando o Dataset
start_time = time.time()
df = read_csv('__.csv', usecols=[5], \
        engine='python')
dataset = df.values

# Normalização do dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

# Separando o dataset para treinamento e teste
train_size = int(len(dataset) * 0.70)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

# formatacao para X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)

# formatação dos dados de entrada para [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Código Fonte 4.3. Parametrização do modelo LSTM.

```
# Criação e treinamento do modelo LSTM
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

Por fim, para validar as métricas de desempenho e qualidade do modelo, é calculada a raiz do erro quadrático médio (***Root Mean-Square Error*** - RMSE). Essa métrica é

Código Fonte 4.4. Predição dos pontos da série.

```
# Realizando as predições
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
```

frequentemente utilizada para avaliar a diferença entre os valores previstos e valores observados em modelos obtidos a partir de algoritmos de aprendizado de máquina. Também é verificada ao término da execução do código, a quantidade de tempo necessária para finalizar a execução. Os resultados obtidos são apresentados pelo Código Fonte 4.5. Para exibir de maneira gráfica o resultado do modelo, o Código Fonte 4.6 é usado para gerar um gráfico da predição contendo os valores originais da série e a predição realizada. Cabe destacar que, dependendo do estudo a ser realizado, é interessante armazenar os valores preditos em listas.

Código Fonte 4.5. Resultados do modelo e tempo de execução.

```
# Resultados
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print ("\n")
print("=====================================================")
print("Resultados do algoritmo")
print("=====================================================")
print('Valor RMSE do treinamento: %.2f' % (trainScore))
print('Valor RMSE do teste: %.2f' % (testScore))
print("Tempo de treinamento e predição: %.2f segundos" % \
        (time.time() - start_time))
print ("\n")
```

No segundo exemplo, é utilizado o classificador *One-Class* SVM para detectar fluxos anormais no tráfego de uma rede. Para isso, utiliza-se um conjunto de dados com informações reais, rotulado com diversos tipos de ataque. Como a premissa desse classificador é identificar apenas uma classe alvo, é necessário treinar o modelo com os fluxos normais e fluxos anormais. Inicialmente, como no exemplo anterior, é realizada a importação das bibliotecas necessárias: Pandas, NumPy e Scikit-Learn, conforme apresentado no Código Fonte 4.7.

A etapa de processamento para realização do treinamento é a mais importante. Um fluxo contém informações que nem sempre são utilizadas, sendo fundamental que haja uma redução da dimensão do conjunto de dados para otimizar os recursos computacionais, principalmente alocação de memória. Evita-se, dessa forma, que informações

Código Fonte 4.6. Configuração do gráfico da série temporal e predição realizada

```
# Plotagem da série original e valores preditos
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[ \
        look_back:len(trainPredict)+look_back, :] = trainPredict
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+ \
        1:len(dataset)-1, :] = testPredict
plt.plot(dataframe, label='Valores Originais', color="deepskyblue")
plt.plot(testPredictPlot, label='Valores da Predição', \
        color="darkred", linestyle='--')
plt.ylabel('Componentes não lineares da entropia para protocolo', \
        fontsize=16)
plt.xlabel('Amostras',fontsize=16)
plt.legend(loc="lower left", prop={'size': 13})
plt.yticks(fontsize=16)
plt.xticks(fontsize=16)
plt.show()
```

Código Fonte 4.7. Importação das bibliotecas

```
from __future__ import division
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn import metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

que não sejam pertinentes sejam armazenadas em memória. Cabe destacar que a utilização da biblioteca Pandas possui grande eficiência para conjunto de dados com tamanhos até 1 GB. Para conjunto de dados maiores, a biblioteca passa a oferecer baixo desempenho sendo necessário utilizar outras ferramentas como o Apache Spark[12] e Apache Hadoop[13]. No exemplo, é utilizado um conjunto de dados com quarenta e sete características de fluxo e são utilizadas apenas três para identificar anomalias, sendo elas *sflow_fbytes*, *sflow_bbytes* e *class*. Uma função de normalização é aplicada para otimizar o treinamento do modelo. Em seguida, é utilizada a característica *class* para criar dois novos tipos de classes, entre normal e anormal. Nesse caso, como os valores das classes correspondem a valores numéricos, 0 representa fluxos normais e qualquer outro valor representa algum

[12]https://spark.apache.org/
[13]https://hadoop.apache.org/

tipo de anomalia. Então, é feito um mapeamento no qual anomalias são representadas pelo valor -1 e tráfego normal é caracterizado por 1. Por último, o conjunto de dados é dividido entre treinamento e validação com 70% e 30% respectivamente. O Código Fonte 4.8 apresenta essa etapa.

Código Fonte 4.8. Processamento do conjunto de dados.

```
# carregando o Dataset e seleção das características
df = pd.read_csv('__.csv', low_memory=False)
features = ["sflow_fbytes", "sflow_bbytes", "class"]

#redução do dataset apenas com as características necessárias para treinar o modelo
df = df[features]

# normalização dos dados
df["sflow_fbytes"] = np.log((df["sflow_fbytes"]).astype(float))
df["sflow_bbytes"] = np.log((df["sflow_bbytes"]).astype(float))

# este dataset esta rotulado com diversos tipos de ataque. Serão tratados então 0 como trafego "normal" e diferente
# de 0 "anomalia" atribuindo 1 e -1 respectivamente
df.loc[df['class'] == 0, "ataque"] = 1
df.loc[df['class'] != 0, "ataque"] = -1

#
target = df['ataque']

#separação das do dataset em treinamento de teste
trainX, testX, trainY, testY = train_test_split(df, target, train_size = 0.70)
```

A última etapa desse exemplo é apresentada no Código Fonte 4.9, no qual o modelo é treinado e o seu desempenho é avaliado. Na função **svm.OneClassSVM**, são repassados os parâmetros *nu*, *kernel* e *gamma* que representam a precisão da regressão, o mapeamento de observações não lineares em um espaço de dimensão superior e a influência que um único exemplo de treinamento pode alcançar, respectivamente. Além disto, são utilizadas as principais métricas de desempenho de aprendizado de máquinas para avaliar o quão satisfatório é o modelo. Nesse caso, são avaliadas as métricas tanto do treinamento quanto da detecção. Assim, no código, são utilizadas as funções da biblioteca Scikit-Learn: *metrics.accuracy_score* que infere a acurácia; *metrics.precision_score* que avalia a precisão; *metrics.recall_score* diz respeito à revocação; e *metrics.f1_score* utilizada para mensurar medida F1, que é a média harmônica entre precisão e revocação.

Código Fonte 4.9. Validação do modelo.

```
#treinamento do modelo
model = svm.OneClassSVM(nu=0.2, kernel='rbf', gamma='auto')
model.fit(trainX)
values_preds = model.predict(trainX)
values_targs = train_target
print "============TREINAMENTO============"
print "Acurácia: %.2f" % (100 *  metrics.accuracy_score(values_targs,
 ↪  values_preds)), str("%")
print "Precisão: %.2f" % (100  * metrics.precision_score(values_targs,
 ↪  values_preds)), str("%")
print "Recall: %.2f" % (100* metrics.recall_score(values_targs,
 ↪  values_preds)), str("%")
print "F1-Score: %.2f" % (100 * metrics.f1_score(values_targs,
 ↪  values_preds)), str("%")

#Validação do modelo
values_preds_test = model.predict(test_data)
values_targs = test_target
print "============VALIDAÇÃO============"
print "Acurácia: %.2f" % (100* metrics.accuracy_score(values_targs,
 ↪  values_preds_test)), str("%")
print "Precisão: %.2f" % (100 * metrics.precision_score(values_targs,
 ↪  values_preds_test)), str("%")
print "Recall: %.2f" % (100 * metrics.recall_score(values_targs,
 ↪  values_preds_test)), str("%")
print "F1-Score: %.2f" % (100 * metrics.f1_score(values_targs,
 ↪  values_preds_test)), str("%")
print "================================="
```

4.8. Considerações Finais

As redes móveis propiciaram um grande avanço nas comunicações, pois em função de suas características físicas podem estar presentes em qualquer localidade. Além disto, a comunicação de dados através destas redes, influenciou como as relações humanas se desenvolveram ao longo dos últimos anos, pois diversas aplicações foram criadas e outras aprimoradas para possibilitar a interação da informação em tempo real, transmissão de vídeos, entre outras formas de interação humana mediadas pelas redes móveis. Com a chegada da rede 5G, as relações entre dispositivos serão abordadas de maneira acentuada, permitindo que a inteligência artificial seja aplicada para otimização de recursos energéticos, hídricos e até mesmo alimentares. A rede 5G possui avanços significativos com relação as gerações anteriores, tais como Serviços de Missão Crítica (MCS) e Banda larga móvel aprimorada (eMBB). Veículos autônomos e serviços de saúde remotos são exemplos de novos paradigmas para serviços de missão crítica, no qual a baixa latência e as altas taxas de transmissão são fundamentais para serem efetivas, pois podem ocasionar danos irreparáveis. Além disso, as altas taxas de transmissão permitirão uma

maior demanda de vídeos com alta definição, gerando um tráfego acentuado nas redes. Essas premissas podem ser aplicadas para casos em que sejam necessários a utilização de realidade virtual (*Virtual Reality* - VR), realidade aumentada (*Augumented Reality* - AR) ou realizar o monitoramento e rastreamento de doenças infectocontagiosas através de sistemas de vigilância com câmeras infravermelhas.

A predição de tráfego e detecção de anomalias em redes sempre foram atividades desafiadoras em virtude da grande massa de dados. O cenário torna-se ainda mais complexo com na rede 5G, em virtude de a análise em tempo real demandar um processamento de grandes fluxos de dados principalmente na borda da rede. Este capítulo apresentou as principais técnicas de predição de tráfego utilizando modelos estocásticos amplamente estudados, além de ferramentas de aprendizado de máquinas para detecção de anomalias. As **redes definidas por *software*** fazem parte do núcleo das comunicações das tecnologias 5G e 6G, ambas com foco em um maior número de dispositivos conectados nas bordas. A predição de tráfego permite que um controlador SDN realize modificações no roteamento, distribuindo políticas para todos os segmentos de rede e evite congestionamentos de maneira pró-ativa. Além disso, permite a implementação dinâmica de novos serviços como balanceadores de carga, *firewalls*, comutadores virtuais, entre outros, através da **virtualização de funções de rede**. A predição de anomalias por sua vez, permite realizar contramedidas para mitigar os efeitos de ataques na eminência de ocorrer. Com o aumento de dispositivos móveis, técnicas como o aprendizado federado poderão ser úteis para realizar a detecção de ameaças garantido a privacidade dos dados e distribuindo o processamento. Esse paradigma permite ainda, que diferentes entidades possam contribuir no treinamento do modelo sem que os dados possam ser acessados entre as organizações.

O capítulo analisou ainda desafios futuros para as redes de próxima geração, 6G. Os desafios relativos ao aprendizado de máquina são ainda mais críticos para a próxima geração de redes móveis, pois nessas novas redes as aplicações de aprendizado de máquina farão parte do núcleo da rede, em contraposição ao que ocorre nas redes 5G, em que as aplicações de aprendizado de máquina são acessórias ao gerenciamento e controle da rede. O capítulo desenvolveu ainda três exemplos práticos de aplicação de aprendizado de máquina para a gerência da segurança em redes 5G. A predição de tráfego com o modelo auto-regressivo integrado de médias móveis (*Autoregressive Integrated Moving Average* - ARIMA) mostrou a aplicação de uma técnica clássica para a regressão de séries temporais. A predição de tráfego com a rede neural de memória longa de curto prazo (*Long Short-Term Memory* - LSTM) evidenciou o uso de uma aplicação atual, aplicando bibliotecas de código aberto, e com alta acurácia na predição. Por fim, o modelo de máquina de vetor de suporte de uma única classe (*One Class Support Vector Machine* - OCSVM) exemplificou um cenário de identificação de anomalias nos fluxos de rede através do treinamento de um mecanismo de aprendizado de máquina voltado para detecção de amostras destoantes.

Referências

[Ahmad et al., 2020] Ahmad, A., Harjula, E., Ylianttila, M. e Ahmad, I. (2020). Evaluation of machine learning techniques for security in SDN. Em *2020 IEEE Globecom Workshops (GC Wkshps*, p. 1–6.

[Ahmad et al., 2017a] Ahmad, I., Kumar, T., Liyanage, M., Okwuibe, J., Ylianttila, M. e Gurtov, A. (2017a). 5G security: Analysis of threats and solutions. Em *2017 IEEE Conference on Standards for Communications and Networking*, p. 193–199. IEEE.

[Ahmad et al., 2018] Ahmad, I., Kumar, T., Liyanage, M., Okwuibe, J., Ylianttila, M. e Gurtov, A. (2018). Overview of 5G security challenges and solutions. *IEEE Communications Standards Magazine*, 2(1):36–43.

[Ahmad et al., 2019] Ahmad, I., Shahabuddin, S., Kumar, T., Okwuibe, J., Gurtov, A. e Ylianttila, M. (2019). Security for 5G and beyond. *IEEE Communications Surveys & Tutorials*, 21(4):3682–3722.

[Ahmad et al., 2017b] Ahmad, S., Lavin, A., Purdy, S. e Agha, Z. (2017b). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147. Online Real-Time Learning Strategies for Data Streams.

[Alawe et al., 2018] Alawe, I., Ksentini, A., Hadjadj-Aoul, Y. e Bertin, P. (2018). Improving traffic forecasting for 5G core network scalability: A machine learning approach. *IEEE Network*, 32(6):42–49.

[Alwis et al., 2021] Alwis, C. D., Kalla, A., Pham, Q.-V., Kumar, P., Dev, K., Hwang, W.-J. e Liyanage, M. (2021). Survey on 6G frontiers: Trends, applications, requirements, technologies and future research. *IEEE Open Journal of the Communications Society*, 2:836–886.

[Andreoni Lopez et al., 2021] Andreoni Lopez, M., Baddeley, M., Lunardi, W. T., Pandey, A. e Giacalone, J.-P. (2021). Towards secure wireless mesh networks for uav swarm connectivity: Current threats, research, and opportunities. Em *Proceeding of 3rd International Workshop on Wireless Sensors and Drones in Internet of Things (WiDroIT) 2021*, p. 1–6.

[Andreoni Lopez e Mattos, 2021] Andreoni Lopez, M. e Mattos, D. (2021). Resumo de grandes volumes de dados com filtro de bloom: Uma abordagem eficiente para aprendizado profundo com redes neurais convolucionais em fluxos de rede. Em *Anais do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, p. 532–545, Porto Alegre, RS, Brasil. SBC.

[Andreoni Lopez et al., 2019] Andreoni Lopez, M., Mattos, D. M., Duarte, O. C. M. e Pujolle, G. (2019). Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data. *Concurrency and Computation: Practice and Experience*, 31(20):e5344.

[Andreoni Lopez et al., 2016] Andreoni Lopez, M., Mattos, D. M. F. e Duarte, O. C. M. (2016). An elastic intrusion detection system for software networks. *Annals of Telecommunications*, 71(11):595–605.

[Ariyaluran Habeeb et al., 2019] Ariyaluran Habeeb, R. A., Nasaruddin, F., Gani, A., Targio Hashem, I. A., Ahmed, E. e Imran, M. (2019). Real-time big data processing for anomaly detection: A survey. *International Journal of Information Management*, 45:289–307.

[Benslimen et al., 2021] Benslimen, Y., Sedjelmaci, H. e Manenti, A.-C. (2021). Attacks and failures prediction framework for a collaborative 5G mobile network. *Computing*, 103(6):1165–1181.

[Bochie et al., 2020] Bochie, K., Gilbert, M., Gantert, L., Barbosa, M., Medeiros, D. e Campista, M. (2020). Aprendizado profundo em redes desafiadoras: Conceitos e aplicações. Em *Minicursos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, p. 140–189. SBC.

[Bockelmann et al., 2016] Bockelmann, C., Pratas, N., Nikopour, H., Au, K., Svensson, T., Stefanovic, C., Popovski, P. e Dekorsy, A. (2016). Massive machine-type communications in 5G: Physical and mac-layer solutions. *IEEE Communications Magazine*, 54(9):59–65.

[Boukerche et al., 2020] Boukerche, A., Tao, Y. e Sun, P. (2020). Artificial intelligence-based vehicular traffic flow prediction methods for supporting intelligent transportation systems. *Computer Networks*, 182:107484.

[Camilo et al., 2020] Camilo, G. F., Rebello, G. A. F., de Souza, L. A. C. e Duarte, O. C. M. B. (2020). AutAvailChain: Automatic and secure data availability through blockchain. Em *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, p. 1–6.

[Chadza et al., 2020] Chadza, T., Kyriakopoulos, K. G. e Lambotharan, S. (2020). Analysis of hidden markov model learning algorithms for the detection and prediction of multi-stage network attacks. *Future Generation Computer Systems*, 108:636–649.

[Chakraborty et al., 2020] Chakraborty, P., Corici, M. e Magedanz, T. (2020). A comparative study for time series forecasting within software 5G networks. Em *2020 14th International Conference on Signal Processing and Communication Systems (ICSPCS)*, p. 1–7.

[Chowdhury et al., 2020] Chowdhury, M. Z., Shahjalal, M., Ahmed, S. e Jang, Y. M. (2020). 6G wireless communication systems: Applications, requirements, technologies, challenges, and research directions. *IEEE Open Journal of the Communications Society*, 1:957–975.

[Cunha et al., 2019] Cunha, V. A., da Silva, E., de Carvalho, M. B., Corujo, D., Barraca, J. P., Gomes, D., Granville, L. Z. e Aguiar, R. L. (2019). Network slicing security: Challenges and directions. *Internet Technology Letters*, 2(5):e125.

[Cunha Neto et al., 2020] Cunha Neto, H. N., Mattos, D. M. F. e Fernandes, N. C. (2020). Privacidade do usuário em aprendizado colaborativo: Federated learning, da teoria à prática. *Minicursos do Simpósio Brasileiro de Segurança de Informação e de Sistemas Computacionais - SBSeg*, 20:142–195.

[De Assis et al., 2017] De Assis, M. V. O., Hamamoto, A. H., Abrão, T. e Proença, M. L. (2017). A game theoretical based system using holt-winters and genetic algorithm with fuzzy logic for DoS/DDoS mitigation on sdn networks. *IEEE Access*, 5:9485–9496.

[De Ree et al., 2019] De Ree, M., Mantas, G., Radwan, A., Mumtaz, S., Rodriguez, J. e Otung, I. E. (2019). Key management for beyond 5G mobile small cells: A survey. *IEEE Access*, 7:59200–59236.

[Di Bernardino e Brogi, 2019] Di Bernardino, E. e Brogi, G. (2019). Hidden markov models for advanced persistent threats. *International Journal of Security and Networks*, 14:181.

[Dogra et al., 2021] Dogra, A., Jha, R. K. e Jain, S. (2021). A survey on beyond 5G network with the advent of 6G: Architecture and emerging technologies. *IEEE Access*, 9:67512–67547.

[Fernandez Maimo et al., 2018] Fernandez Maimo, L., Perales Gomez, A. L., Garcia Clemente, F. J., Gil Perez, M. e Martinez Perez, G. (2018). A self-adaptive deep learning-based system for anomaly detection in 5G networks. *IEEE Access*, 6:7700–7712.

[Fu et al., 2018] Fu, Y., Wang, S., Wang, C.-X., Hong, X. e McLaughlin, S. (2018). Artificial intelligence to manage network traffic of 5g wireless networks. *IEEE Network*, 32(6):58–64.

[Giordani et al., 2020] Giordani, M., Polese, M., Mezzavilla, M., Rangan, S. e Zorzi, M. (2020). Toward 6G networks: Use cases and technologies. *IEEE Communications Magazine*, 58(3):55–61.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y. e Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Hanbanchong e Piromsopa, 2012] Hanbanchong, A. e Piromsopa, K. (2012). SARIMA based network bandwidth anomaly detection. Em *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE)*, p. 104–108.

[Hasrouny et al., 2017] Hasrouny, H., Samhat, A. E., Bassil, C. e Laouiti, A. (2017). VANet security challenges and solutions: A survey. *Vehicular Communications*, 7:7–20.

[ho Bang et al., 2017] ho Bang, J., Cho, Y.-J. e Kang, K. (2017). Anomaly detection of network-initiated LTE signaling traffic in wireless sensor and actuator networks based on a hidden semi-markov model. *Computers and Security*, 65:108–120.

[Huang et al., 2017] Huang, C.-W., Chiang, C.-T. e Li, Q. (2017). A study of deep learning networks on mobile traffic forecasting. Em *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, p. 1–6.

[Jiang e Schotten, 2019] Jiang, W. e Schotten, H. D. (2019). Neural network-based fading channel prediction: A comprehensive overview. *IEEE Access*, 7:118112–118124.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L. e Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

[Kakkar, 2020] Kakkar, A. (2020). A survey on secure communication techniques for 5G wireless heterogeneous networks. *Information Fusion*, 62:89–109.

[Khan et al., 2019] Khan, R., Kumar, P., Jayakody, D. N. K. e Liyanage, M. (2019). A survey on security and privacy of 5G technologies: Potential solutions, recent advancements, and future directions. *IEEE Communications Surveys & Tutorials*, 22(1):196–248.

[Kromkowski et al., 2019] Kromkowski, P., Li, S., Zhao, W., Abraham, B., Osborne, A. e Brown, D. E. (2019). Evaluating statistical models for network traffic anomaly detection. Em *2019 Systems and Information Engineering Design Symposium (SIEDS)*, p. 1–6.

[Kumar Dwivedi et al., 2018] Kumar Dwivedi, R., Pandey, S. e Kumar, R. (2018). A study on machine learning approaches for outlier detection in wireless sensor network. Em *2018 8th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, p. 189–192.

[Lal et al., 2017] Lal, S., Taleb, T. e Dutta, A. (2017). NFV: Security threats and best practices. *IEEE Communications Magazine*, 55(8):211–217.

[Li et al., 2020] Li, J., Li, Z., Tyson, G. e Xie, G. (2020). Your privilege gives your privacy away: An analysis of a home security camera service. Em *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, p. 387–396.

[Li et al., 2011] Li, Y., Kaur, B. e Andersen, B. (2011). Denial of service prevention for 5G. *Wireless Personal Communications*, 57(3):365–376.

[Lobato et al., 2021] Lobato, A. G. P., Andreoni Lopez, M., Cardenas, A. A., Duarte, O. C. M. B. e Pujolle, G. (2021). A fast and accurate threat detection and prevention architecture using stream processing. *Concurrency and Computation: Practice and Experience*, e6561.

[Lopez-Martin et al., 2017] Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A. e Lloret, J. (2017). Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in IoT. *Sensors*, 17(9).

[Lu et al., 2021] Lu, N., Li, D., Shi, W., Vijayakumar, P., Piccialli, F. e Chang, V. (2021). An efficient combined deep neural network based malware detection framework in 5G environment. *Computer Networks*, 189:107932.

[Luo et al., 2020] Luo, C., Ji, J., Wang, Q., Chen, X. e Li, P. (2020). Channel state information prediction for 5G wireless communications: A deep learning approach. *IEEE Transactions on Network Science and Engineering*, 7(1):227–236.

[Medeiros et al., 2019] Medeiros, D., Cunha Neto, H., Andreoni, M., Magalhães, L., Silva, E., Borges, A., Fernandes, N. e Menezes, D. (2019). *Análise de Dados em Redes Sem Fio de Grande Porte: Processamento em Fluxo em Tempo Real, Tendências e Desafios*, p. 142–195. Sociedade Brasileira de Computação.

[Nie et al., 2017] Nie, L., Jiang, D., Yu, S. e Song, H. (2017). Network traffic prediction based on Deep Belief Network in wireless mesh backbone networks. Em *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, p. 1–5.

[Nieto et al., 2019] Nieto, A., Acien, A. e Fernandez, G. (2019). Crowdsourcing analysis in 5G IoT: Cybersecurity threats and mitigation. *Mobile Networks and Applications*, 24(3):881–889.

[Popovski et al., 2018] Popovski, P., Trillingsgaard, K. F., Simeone, O. e Durisi, G. (2018). 5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view. *IEEE Access*, 6:55765–55779.

[Porambage et al., 2021] Porambage, P., Gür, G., Moya Osorio, D. P., Livanage, M. e Ylianttila, M. (2021). 6G security challenges and potential solutions. Em *2021 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit)*, p. 622–627.

[Ramakrishnan e Soni, 2018] Ramakrishnan, N. e Soni, T. (2018). Network traffic prediction using recurrent neural networks. Em *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, p. 187–193.

[Reddy e Thilagam, 2020] Reddy, K. e Thilagam, P. (2020). Naïve bayes classifier to mitigate the DDoS attacks severity in ad-hoc networks. *International Journal of Communication Networks and Information Security*, 12:221–226.

[Reis et al., 2020] Reis, L. H. A., Magalhães, L. C. S., de Medeiros, D. S. V. e Mattos, D. M. (2020). An unsupervised approach to infer quality of service for large-scale wireless networking. *Journal of Network and Systems Management*, 28(4):1228–1247.

[Scott-Hayward et al., 2015] Scott-Hayward, S., Natarajan, S. e Sezer, S. (2015). A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, 18(1):623–654.

[Sedjelmaci, 2021] Sedjelmaci, H. (2021). Cooperative attacks detection based on artificial intelligence system for 5G networks. *Computers and Electrical Engineering*, 91:107045.

[Sivanathan et al., 2019] Sivanathan, A., Gharakheili, H. H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A. e Sivaraman, V. (2019). Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 18(8):1745–1759.

[Sone et al., 2020] Sone, S. P., Lehtomäki, J. J. e Khan, Z. (2020). Wireless traffic usage forecasting using real enterprise network data: Analysis and methods. *IEEE Open Journal of the Communications Society*, 1:777–797.

[Sun et al., 2020] Sun, Y., Liu, J., Wang, J., Cao, Y. e Kato, N. (2020). When machine learning meets privacy in 6G: A survey. *IEEE Communications Surveys Tutorials*, 22(4):2694–2724.

[Swarnkar e Hubballi, 2016] Swarnkar, M. e Hubballi, N. (2016). OCPAD: One class naive bayes classifier for payload based anomaly detection. *Expert Systems with Applications*, 64:330–339.

[Tang et al., 2020] Tang, F., Kawamoto, Y., Kato, N. e Liu, J. (2020). Future intelligent and secure vehicular network toward 6G: Machine-learning approaches. *Proceedings of the IEEE*, 108(2):292–307.

[Tartakovsky et al., 2013] Tartakovsky, A. G., Polunchenko, A. S. e Sokolov, G. (2013). Efficient computer network anomaly detection by changepoint detection methods. *IEEE Journal of Selected Topics in Signal Processing*, 7(1):4–11.

[Trinh et al., 2019] Trinh, H. D., Zeydan, E., Giupponi, L. e Dini, P. (2019). Detecting mobile traffic anomalies through physical control channel fingerprinting: A deep semi-supervised approach. *IEEE Access*, 7:152187–152201.

[Varalakshmi e Judgi, 2017] Varalakshmi, P. e Judgi, T. (2017). Multifaceted trust management framework based on a trust level agreement in a collaborative cloud. *Computers & Electrical Engineering*, 59:110–125.

[Wan et al., 2020] Wan, Y., Xu, K., Xue, G. e Wang, F. (2020). IoTArgos: A multi-layer security monitoring system for internet-of-things in smart homes. Em *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, p. 874–883.

[Wang et al., 2017] Wang, J., Tang, J., Xu, Z., Wang, Y., Xue, G., Zhang, X. e Yang, D. (2017). Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach. Em *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, p. 1–9.

[Wang et al., 2021] Wang, Y., Masoud, N. e Khojandi, A. (2021). Real-time sensor anomaly detection and recovery in connected automated vehicle sensors. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1411–1421.

[Wasicek, 2020] Wasicek, A. (2020). The future of 5G smart home network security is micro-segmentation. *Network & Security*, 2020(11):11–13.

[Wazid et al., 2020] Wazid, M., Das, A. K., Shetty, S., Gope, P. e Rodrigues, J. J. (2020). Security in 5G-enabled internet of things communication: issues, challenges, and future research roadmap. *IEEE Access*, 9:4466–4489.

[Wu et al., 2020] Wu, D., Nekovee, M. e Wang, Y. (2020). Deep learning-based autoencoder for m-user wireless interference channel physical layer design. *IEEE Access*, 8:174679–174691.

[Xu et al., 2015] Xu, Q., Liao, Y., Miskovic, S., Mao, Z. M., Baldi, M., Nucci, A. e Andrews, T. (2015). Automatic generation of mobile app signatures from traffic observations. Em *2015 IEEE Conference on Computer Communications (INFOCOM)*, p. 1481–1489.

[Yang et al., 2021] Yang, H., Li, X., Qiang, W., Zhao, Y., Zhang, W. e Tang, C. (2021). A network traffic forecasting method based on sa optimized arima–bp neural network. *Computer Networks*, 193:108102.

[Yao et al., 2019] Yao, J., Han, Z., Sohail, M. e Wang, L. (2019). A robust security architecture for SDN-based 5G networks. *Future Internet*, 11(4):85.

[Yu, 2010] Yu, S.-Z. (2010). Hidden semi-markov models. *Artificial Intelligence*, 174(2):215–243. Special Review Issue.

[Zafeiropoulos et al., 2020] Zafeiropoulos, A., Fotopoulou, E., Peuster, M., Schneider, S., Gouvas, P., Behnke, D., Müller, M., Bök, P.-B., Trakadas, P., Karkazis, P. e Karl, H. (2020). Benchmarking and profiling 5G verticals' applications: An industrial IoT use case. Em *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, p. 310–318.

[Zhan et al., 2020] Zhan, M., Li, Y., Yang, X., Cui, W. e Fan, Y. (2020). NSAPs: A novel scheme for network security state assessment and attack prediction. *Computers and Security*, 99:102031.

[Zhang et al., 2019a] Zhang, C., Patras, P. e Haddadi, H. (2019a). Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys Tutorials*, 21(3):2224–2287.

[Zhang et al., 2019b] Zhang, S., Wang, Y. e Zhou, W. (2019b). Towards secure 5G networks: A survey. *Computer Networks*, 162:106871.

[Zhu et al., 2019] Zhu, G., Zan, J., Yang, Y. e Qi, X. (2019). A supervised learning based QoS assurance architecture for 5G networks. *IEEE Access*, 7:43598–43606.

Patrocínio

Diamante

nic.br cgi.br

Prata

Banpará

PRODEPA

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

Bronze

CLAVIS
segurança da informação

ripple impact
university blockchain research initiative