David Henrique da Costa, Pedro Henrique Florio Mendes

Ampliando a Expressividade dos Tokens SPIFFE

São Paulo, SP

2023

David Henrique da Costa, Pedro Henrique Florio Mendes

Ampliando a Expressividade dos Tokens SPIFFE

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Prof. Dr. Marcos Antonio Simplicio Junior

São Paulo, SP

2023

Agradecimentos

Nós estamos muito felizes e emocionados por ter finalizado este trabalho de conclusão de curso. Foi um desafio que exigiu muito esforço, dedicação e aprendizado. Queremos agradecer a todos que nos apoiaram e nos orientaram nessa jornada acadêmica.

Em primeiro lugar, agradecemos aos nossos professores, que nos ensinaram os conceitos, as técnicas e as metodologias que aplicamos neste trabalho. Eles foram fundamentais para o nosso desenvolvimento e para a qualidade do nosso projeto.

Em segundo lugar, agradecemos aos nossos colegas, que compartilharam conosco as suas experiências, as suas dúvidas e as suas sugestões. Eles foram nossos parceiros e nossos amigos, que nos incentivaram e nos ajudaram a superar as dificuldades.

Em terceiro lugar, agradecemos às nossas famílias, que nos deram todo o amor, o carinho e o apoio que precisávamos. Eles foram a nossa motivação e a nossa inspiração, que nos fizeram acreditar nos nossos sonhos e nos nossos objetivos.

Por fim, agradecemos a nós mesmos, que nos unimos e nos dedicamos a este trabalho com paixão, criatividade e responsabilidade. Nós nos orgulhamos do que fizemos e do que aprendemos. Nós nos sentimos realizados e gratos por ter concluído esta etapa tão importante das nossas vidas.

Resumo

A computação em nuvem vem ganhando espaço nas organizações devido às vantagens oferecidas, como escalabilidade, recuperação de falhas e redução de custos operacionais. No entanto, para aproveitar ao máximo as possibilidades da nuvem, as aplicações devem ser projetadas com uma arquitetura *"Cloud Native"*, como a de microsserviços. Isso também traz a necessidade de projetos de segurança específicos, pois a gestão de identidades e o controle de acesso em ambientes descentralizados são diferentes das aplicações monolíticas. O uso do SPIFFE, uma especificação aberta, junto ao SPIRE, sua implementação, pode fornecer identidades verificáveis e gerenciar a validação de identidade em ambientes heterogêneos. Este trabalho tem como objetivo criar um *framework* para integrar ao SPIFFE uma lógica de asserções distribuídas feitas por agentes localizados dentro de um mesmo domínio de confiança. O *framework* será utilizado, em sequência, na criação de uma prova de conceito que, através dessas asserções assinadas, realiza autenticação de um usuário final de maneira segura em uma aplicação através de um token OAuth.

Palavras-chave: asserções. SPIFFE. SPIRE. segurança. computação em nuvem.

Abstract

Cloud computing has been gaining traction in organizations due to the advantages it offers, such as scalability, fault recovery, and operational cost reduction. However, to fully leverage the possibilities of the cloud, applications must be designed with a "Cloud Native" architecture, such as microservices. This also brings the need for specific security projects, as identity management and access control in decentralized environments differ from monolithic applications. The use of SPIFFE, an open specification, along with SPIRE, it's implementation, can provide verifiable identities and manage identity validation in heterogeneous environments. This work aims to create a framework to integrate SPIFFE with a logic of distributed assertions, made by agents inside a trust domain. The framework will then be tested in a Proof of Concept application, that will use this distributed assertions to grant authentication to an end-user in a secure way via an Oauth token.

Keywords: assertions. SPIFFE. SPIRE. security. cloud computing.

Lista de ilustrações

Lista de abreviaturas e siglas

AC	Autoridade Certificadora
API	*Application Programming Interface*
AWS	*Amazon Web Services*
CD	*Continuous Deployment*
CI	*Continuous Integration*
CLI	*Command Line Interface*
CN	*Common Name*
CSP	*Cloud Services Provider*
DA-SVID	*Delegated Authority SPIFFE Verifiable Identity Document*
ECDSA	*Elliptic Curve Digital Signature Algorithm*
EdDSA	*Edwards-curve Digital Signature Algorithm*
ESB	*Enterprise Service Bus*
HPE	*Hewlett Packard Enterprise*
IaaS	*Infrastructure as a Service*
IaC	*Infrastructure as Code*
IBM	*International Business Machines Corporation*
IDP	*Identity Provider*
IETF	*Internet Engineering Task Force*
IFC	Instituto Federal Catarinense
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
LSVID	*Lightweight* SVID

mTLS	*Mutual Transport Layer Security*
PaaS	*Platform as a Service*
PK	*Private Key*
RFC	*Request for Comments*
SaaS	*Software as a Service*
SPIFFE	*Secure Production Identity Framework for Everyone*
SPIRE	SPIFFE *Runtime Environment*
SVID	SPIFFE *Verifiable Identity Document*
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
TLS	*Transport Layer Security*
UDESC	Universidade do Estado de Santa Catarina
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locators*
USP	Universidade de São Paulo
VPN	*Virtual Private Network*

Sumário

1 Introdução

1.1 Motivação

A computação em nuvem é um tipo de tecnologia que vem ganhando muito espaço dentro das organizações nos últimos anos. A possibilidade de utilizar de máquinas com acesso virtual, ao invés de comprar servidores físicos, pode levar a diversas vantagens se utilizada corretamente, como aplicações mais escaláveis, maior agilidade na recuperação de falhas e menores preços. A diversidade de modelos, como IaaS, PaaS e SaaS também pode facilitar o setup de servidores e diminuir custos operacionais, em especial para em empresas mais jovens, como startups.

Estudos de 2014 já mostravam que a adoção de computação em nuvem em 2014 já haviam atingido 77% das grandes empresas e 73% das empresas pequenas/médias, de acordo com (EL-GAZZAR, 2014).

Entretanto, para que uma aplicação aproveite ao máximo as possibilidades do ambiente em nuvem, ela deve ser projetada para funcionar nesse tipo de ambiente. O resultado de uma aplicação que passa a rodar em um ambiente de nuvem sem um planejamento de arquitetura prévio normalmente é um sistema sem a escalabilidade desejada para esse tipo de ambiente. Para atingir esse potencial, é necessário uma arquitetura *Cloud Native*, como o caso da arquitetura de microsserviços (BALALAIE; HEYDARNOORI; JAMSHIDI, 2016).

Junto com arquiteturas *Cloud Native* surge a necessidade de projetos de segurança específicos para esse tipo de aplicação. O controle de acesso e gestão de identidades em um ambiente granularizado como o de microsserviços tem especificidades muito diferentes das de uma aplicação monolítica. Em ambientes descentralizados, com a comunicação de diversos provedores de nuvem, há a necessidade de confiabilidade para todas as partes e componentes, a fim de garantir a segurança no fluxo de dados e permitindo o uso de recursos apenas por entidades devidamente credenciadas e autorizadas. Além disso, deve-se garantir a autenticação, revogação e federação de ambientes, para gerar mais confiança nos ambientes de nuvem.

Recursos como o autoscaling fazem com que ambientes em nuvem sejam pouco fixos com relação a recursos computacionais. É comum que instâncias computacionais sejam criadas e destruídas rapidamente para suprir demandas em momentos de picos de acesso sem que seja necessário pagar por todos esses recursos computacionais em momentos nos quais não há necessidade deles. Dessa maneira, não é possível tratar a questão de identidade de cargas de trabalho em ambientes de nuvem da mesma maneira que ela é

tratada em ambientes *on-premise*.

Para resolver essas questões, surge o SPIFFE, uma especificação aberta que define um quadro para a emissão e inicialização de identidades de serviços em ambientes heterogêneos formados por várias organizações. O SPIFFE depende de credenciais de curta duração, chamadas de Documento de Identidade Verificável SPIFFE (SVID), usadas quando as cargas de trabalho precisam se autenticar entre si. O SPIRE é uma implementação do SPIFFE que gerencia a validação de identidade de plataforma e carga de trabalho, definindo APIs para gerenciar políticas, bem como emissão e rotação de certificados.

O SPIFFE já é um *framework* bem consolidado, e que muitas empresas vêm adotando. Podemos ver pela página oficial do projeto (SPIFFE, 2023) que empresas como Google, International Business Machines Corporation (IBM), Amazon Web Services (AWS) e Netflix por exemplo, adotaram o seu uso.

Este trabalho faz parte do projeto "Gerenciamento seguro de identidades federadas: aprimorando e extendendo a arquitetura SPIFFE (SPIFFE-IdT)", que junta alunos da Universidade de São Paulo (USP) e Universidade do Estado de Santa Catarina (UDESC) em grupo de pesquisa; apoiado pelas próprias universidades, além do Instituto Federal Catarinense (IFC) e a *Hewlett Packard Enterprise* (HPE).

O grupo tem por objetivo conduzir pesquisas que contribuam com a expansão da utilidade e dos casos de uso do *framework* SPIFFE. Uma parte desta linha de pesquisa, na qual este trabalho está inserido, busca especificamente soluções seguras para mecanicamos de criação e uso de *tokens* e asserções dentro de uma ambiente SPIFFE.

Asserções podem ser entendidas como declarações arbitrárias que podem ser feitas por algum agente dentro de um ambiente, que, de maneira geral, assumem o formato *chave:valor*. Um *token* representa um bloco de informação. Para o proposito deste trabalho, quando citarmos *tokens*, normalmente estaremos nos referindo a um conjunto de asserções assinadas por uma entidade.

1.2 Objetivos e Justificativa

A criação de asserções e uso de *tokens* integrados ao SPIFFE é um assunto discutido dentro da comunidade. Podemos observar que no próprio github do projeto há diversas observações sobre como fazer uso desses mecanismos, erros a se evitar e possíveis problemas. (SPIFFE, s.d.). Note entretanto, que a página de especificação apenas expressa de maneira expositiva possíveis problemas a se evitar, mas não há uma definição concreta de um paradigma de como fazer isso dentro do *framework*.

Como citado anteriormente, o SPIFFE limita-se ao escopo de autenticação de *workloads*. Basicamente, sua intenção principal é criar uma maneira segura de identificar

um *workload* dentro de um ambiente altamente distribuído. Está fora do escopo do *framework* questões de autorização por exemplo (ou seja, dada uma identidade, determinar quais ações ela pode tomar no sistema). Delega-se questões desse tipo à aplicação quando se usa deste *framework*.

Mecanismos seguros de asserções, entretanto, poderiam facilitar que aplicações rodando em ambientes SPIFFE lidassem com essas questões que não são resolvidas diretamente pelo *framework*. Asserções e *tokens* podem ser muito úteis, por exemplo em diversos contextos relacionados a autorização, ou autenticação delegada (quando a identidade de um usuário é provida por outra aplicação).

Dessa maneira, este trabalho busca, de maneira propositiva, criar a especificação de um *framework*, feito a princípio para ser utilizado em conjunto com o SPIFFE/SPIRE (mas que futuramente poderia iclusive tornar-se parte da especificação destes), que permita a criação e utilização de asserções de maneira segura em um ambiente distribuído no qual os *workloads* têm maneiras de se identificar.

Para isto, este trabalho irá usar o conceito de *Nested Tokens* (explicado melhor no Capítulo 2), já desenvolvido pelo projeto que este trabalho faz parte; e atrelar a este conceito, um modo de uso seguro pensando especificamente em um ambiente SPIRE, no qual os *workloads* têm informação confiável a respeito da identidade uns dos outros.

Paralelamente a este trabalho, houve o desenvolvimento de um outro modo de uso de *Nested Tokens*, que não tem a necessidade de confiança a respeito da identidade de outros *workloads* no ambiente distribuído. Daqui em diante, iremos nos referir ao modo de uso proposto neste trabalho como *ID-Mode*, e a este outro modo como *Anonymous Mode*.

É importante ressaltar que não é escopo deste trabalho tratar da interpretação das asserções criadas. O trabalho limita-se a sugerir um padrão para a criação de asserções, que podem ser consumidas pela aplicação da maneira que convir. Portanto, o trabalho também não resolve por si só questões como a de autorização e autenticação delagada; mas cria um padrão para diminuir o trabalho da aplicação em lidar com essas questões; reduzindo também, dessa maneira, as chances de problemas de segurança na solução final.

Numa segunda etapa, este trabalho tem por objetivo, após a criação conceitual do *framework*, de elaborar uma prova de conceito que simule uma aplicação em nuvem, e um usuário final que utiliza de autenticação delegada. Através do uso do *framework*, somado à regras de negócio da aplicação relacionadas a autorização, é desejado que o usuário consiga se autenticar na aplicação utilizando de um provedor de identidade, basicamente assumindo um fluxo parecido ao do *framework* OAuth (HARDT, 2012), (JONES; CAMPBELL; MORTIMORE, 2015), (CAMPBELL et al., 2015).

1.3 Organização do Trabalho

Descreveremos aqui de maneira breve as proximas seções que serão encontradas neste trabalho.

- Capítulo 2: Aspectos Conceituais - aqui serão dadas bases teóricas que são necessárias para compreender o desenvolvimento do trabalho. Destre essas, vale citar que serão citadas propriedades e conceitos do SPIFFE/SPIRE, bem como pesquisas anteriores do SPIFFE-IdT que serviram como base para este trabalho.
- Capítulo 3: Método do Trabalho - aqui será descrito a metodologia do projeto, bem como as etapas de sua realização
- Capítulo 4: Especificação de Requisitos - aqui serão definidos os requisitos funcionais e não funcionais esperados da solução proposta, com ênfase nos requisitos de segurança, dada a natureza do projeto.
- Capítulo 5: Desenvolvimento do Trabalho - aqui será descrito o funcionamento do *ID-Mode*, bem como o funcionamento da prova de conceito criada. O *Anonymous Mode* será brevemente descrito, para possibilitar comparações no capítulo seguinte.
- Capítulo 6: Considerações Finais - aqui serão feitas ponderações sobre o trabalho desenvolvido, comparações com o *Anonymous Mode*, e serão propostas possibilidades de continuidade do trabalho

2 Aspectos Conceituais

Este capítulo apresenta uma série de aspectos conceituais essenciais, cujo entendimento é fundamental para uma apreciação abrangente e aprofundada do trabalho em questão. Neste contexto, abordaremos definições e contextualizações que servem como alicerces teóricos, proporcionando uma base sólida para a compreensão dos temas abordados posteriormente.

2.1 Conceitos Gerais de Computação em Nuvem

Segundo a IBM, a computação em nuvem é o acesso sob demanda, via internet, a recursos computacionais, como aplicativos, servidores (físicos e virtuais), armazenamento de dados, ferramentas de desenvolvimento, capacidades de rede e muito mais, hospedados em um centro de dados remoto gerenciado por um provedor de serviços em nuvem (ou CSP).

A computação em nuvem oferece benefícios significativos às organizações, incluindo redução de custos de TI ao transferir a gestão e as despesas de infraestrutura. Ao utilizar a nuvem, as organizações podem reduzir sua infraestrutura física de TI e eliminar as tarefas onerosas associadas à gestão de data centers. Essa otimização de custos permite que as empresas redirecionem os recursos de TI para tarefas mais valiosas. Além disso, melhora a agilidade e o tempo para obter valor, permitindo a implantação rápida de aplicativos corporativos e capacitando desenvolvedores e cientistas de dados com recursos de autosserviço. Além disso, a escalabilidade em nuvem permite que as empresas ajustem facilmente os recursos computacionais em resposta às demandas flutuantes, aproveitando as redes globais para otimizar o desempenho de aplicativos. Essas vantagens aumentam a eficiência operacional, reduzem os custos e permitem que as empresas se adaptem rapidamente às dinâmicas do mercado, posicionando-as para o sucesso em um ambiente competitivo. (DAVIS, 2022)

Há vários modelos na computação em nuvem que permitem às organizações aproveitarem diferentes níveis de controle, flexibilidade e gerenciamento para suas necessidades computacionais. Os três principais modelos de serviços em nuvem são Infraestrutura como Serviço (IaaS), Plataforma como Serviço (PaaS) e Software como Serviço (SaaS). Esses modelos, frequentemente referidos como "*stack*" de computação em nuvem, podem ser usados individualmente ou em combinação, permitindo que as empresas adaptem seu ambiente em nuvem para atender a requisitos específicos e alcançar seus resultados desejados. Ao entender as distinções entre esses modelos, as organizações podem aproveitar efetivamente a computação em nuvem para impulsionar seus objetivos comerciais.

- Software como Serviço (SaaS) é um modelo de computação em nuvem que oferece aos consumidores acesso a aplicativos hospedados em uma infraestrutura em nuvem. Os usuários podem utilizar esses aplicativos por meio de um navegador da web ou interface de programa, sem a necessidade de gerenciar ou controlar a infraestrutura em nuvem subjacente. O provedor lida com a rede, servidores, sistemas operacionais, armazenamento e recursos de aplicativos, embora os usuários possam ter controle limitado sobre as configurações de aplicativos. (MELL; GRANCE, 2011)

- Plataforma como Serviço (PaaS) fornece aos consumidores a capacidade de implantar seus próprios aplicativos na infraestrutura em nuvem usando linguagens de programação suportadas, bibliotecas, serviços e ferramentas. Embora os usuários não tenham controle sobre a infraestrutura em nuvem subjacente, eles têm controle sobre os aplicativos implantados e as configurações dentro do ambiente de hospedagem de aplicativos. (MELL; GRANCE, 2011)

- Infraestrutura como Serviço (IaaS) permite que os consumidores provisionem recursos essenciais de computação, como capacidade de processamento, armazenamento e redes. Os usuários podem implantar e executar seu próprio software, incluindo sistemas operacionais e aplicativos, na infraestrutura fornecida. Embora os consumidores não tenham controle sobre a infraestrutura em nuvem subjacente, eles têm controle sobre sistemas operacionais, armazenamento e aplicativos implantados. Também pode haver controle limitado sobre componentes de rede selecionados, como *firewalls* do host. (MELL; GRANCE, 2011)

2.1.1 *Workloads* & Escalabilidade

Um conceito que é importante definir pois será bastante utilizado neste trabalho é o conceito de *workload*, ou carga de trabalho. Esse conceito basicamente serve para identificar uma unidade de processamento de um programa em execução, podendo por exemplo estar alocado em uma máquina virtual ou em um container. A mais baixo nível de abstração, uma carga de trabalho sempre estará sendo executado em algum processador, e consumindo memória.

O conceito de escalabilidade tem relação com a ideia estabelecida de *workload*. A escalabilidade é a capacidade de uma aplicação, em momentos de picos de consumo, poder consumir mais recursos do que o que é consumido normalmente. No geral pode-se dividir o conceito em dois principais métodos de escalabilidade:

- Escalabilidade Vertical: nos momentos de picos de consumo de recursos, é provisionado a um *workload* maior capacidade de processamento e mais memória, para que ele consiga lidar com os maiores requisitos.

- Escalabilidade Horizontal: nos momentos de pico de consumo de recursos, são provisionados mais *workloads* para lidas com a maior quantidade de trabalho, paralelizando o processo.

2.1.2 Microsserviços

Os microsserviços são um estilo de arquitetura de aplicativos que envolve a divisão de um aplicativo em uma coleção de serviços independentes que se comunicam por meio de APIs leves. Cada microsserviço representa uma função específica dentro do aplicativo. Por exemplo, em um varejista online, a barra de pesquisa, as recomendações de produtos e o carrinho de compras seriam todos microserviços separados. Essa arquitetura permite um desenvolvimento mais eficiente, onde equipes de desenvolvimento e operação podem trabalhar simultaneamente em diferentes serviços, resultando em ciclos de desenvolvimento mais rápidos e menos tempo gasto no desenvolvimento.

Os benefícios dos microsserviços vão além da velocidade de desenvolvimento. Comparados a aplicativos monolíticos, os microsserviços são mais fáceis de construir, testar, implantar e atualizar. (BLINOWSKI; OJDOWSKA; PRZYBYLEK, 2022) Eles permitem que as empresas reajam rapidamente a novas demandas e entreguem valor aos clientes imediatamente. Os microsserviços também promovem o desenvolvimento distribuído, permitindo que diferentes partes da equipe de desenvolvimento trabalhem de forma simultânea e ágil. No entanto, a integração dos microsserviços com aplicativos legados e fontes de dados pode ser um desafio. Embora uma camada de malha de serviços simplifique a comunicação entre os serviços, a integração com tecnologias centralizadas, como um barramento de serviços corporativos (ESB), pode dificultar a agilidade e os objetivos dos microsserviços.

Para superar esses desafios, é recomendada a integração ágil, que combina tecnologias de integração, técnicas ágeis de entrega e plataformas nativas da nuvem para melhorar a velocidade e a segurança da entrega de software. Ela permite uma conexão perfeita entre os microsserviços e outros recursos, ao mesmo tempo em que mantém a agilidade e a capacidade de resposta da arquitetura de microsserviços. Ao adotar uma arquitetura de microsserviços e implementar práticas de integração ágil, as empresas podem alcançar maior flexibilidade, escalabilidade e eficiência em seus processos de desenvolvimento e entrega de aplicativos. (Red Hat, 2023)

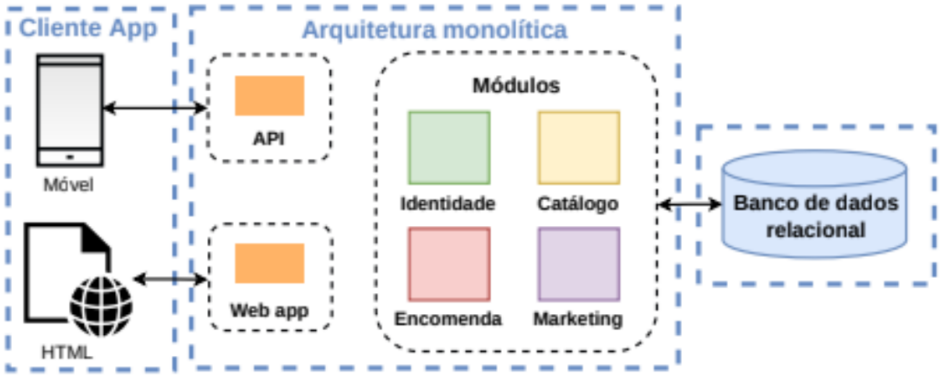
2.1.2.1 Arquitetura Monolítica vs. Arquitetura de Microsserviços

A abordagem tradicional para a construção de aplicativos tem se baseado em uma arquitetura monolítica, na qual todas as funções e serviços estão intimamente acoplados como uma única unidade. Essa complexidade dificulta a otimização de funções individuais sem interromper todo o aplicativo. Além disso, escalar um processo requer escalar todo o aplicativo. Por outro lado, a arquitetura de microserviços permite que cada função central

de um aplicativo opere de forma independente. Essa independência permite que equipes de desenvolvimento construam e atualizem componentes sem interromper todo o aplicativo. A arquitetura de microserviços promove flexibilidade, escalabilidade e a capacidade de atender às necessidades de negócio em constante mudança, sem as limitações impostas pelas arquiteturas monolíticas.

Em uma arquitetura monolítica, todos os processos estão intimamente acoplados e executados como um único serviço, o que exige a escalabilidade de toda a arquitetura, mesmo para um aumento repentino na demanda de um único processo. Além disso, à medida que a base de código cresce, adicionar ou aprimorar recursos em um aplicativo monolítico se torna mais complexo, limitando a experimentação e tornando desafiador implementar novas ideias. O risco de indisponibilidade do aplicativo é maior em arquiteturas monolíticas devido às interdependências entre os processos, uma vez que uma falha em um processo pode ter um impacto em cascata nos demais.

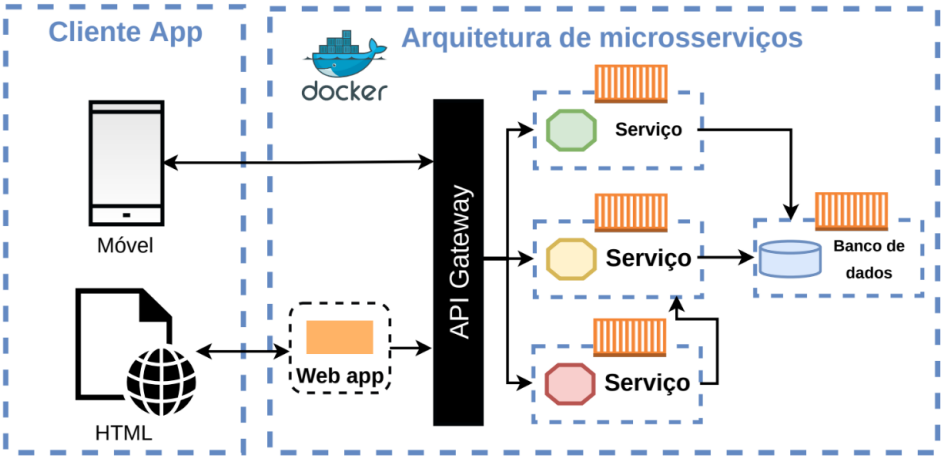
Figura 1 – Exemplo de uma aplicação baseada em arquitetura monolítica.



Fonte: Adaptado de (VETTOR; SMITH, 2023)

Em contraste, uma arquitetura de microserviços divide um aplicativo em componentes independentes, nos quais cada componente opera como um serviço separado. (LARSSON, 2014) Esses serviços se comunicam por meio de interfaces bem definidas e APIs leves. Cada serviço é projetado para desempenhar uma função específica ou capacidade de negócio, permitindo que os serviços individuais sejam atualizados, implantados e dimensionados independentemente, com base na demanda. Essa abordagem modular e descentralizada oferece maior agilidade, escalabilidade e confiabilidade em comparação com as arquiteturas monolíticas, permitindo que equipes de desenvolvimento construam e atualizem componentes separadamente, atendendo às mudanças nas necessidades do negócio sem impactar todo o aplicativo.

Figura 2 – Exemplo de uma aplicação baseada em arquitetura de microsserviços.



Fonte: Adaptado de (VETTOR; SMITH, 2023)

2.1.3 *Cloud Native*

De acordo com a *Cloud Native Computing Foundation*, as tecnologias *Cloud Native* (em português: nativas em nuvem) capacitam organizações a construir e executar aplicativos escaláveis em ambientes modernos e dinâmicos, como nuvens públicas, privadas e híbridas. *Containers*, *service meshes*, microsserviços, infraestrutura imutável e APIs declarativas exemplificam essa abordagem. Essas técnicas permitem criar sistemas de baixo acoplamento, que são resilientes, gerenciáveis e observáveis. Combinadas com automações robustas, elas permitem que os engenheiros realizem mudanças de alto impacto com frequência e previsibilidade, com o mínimo de esforço.

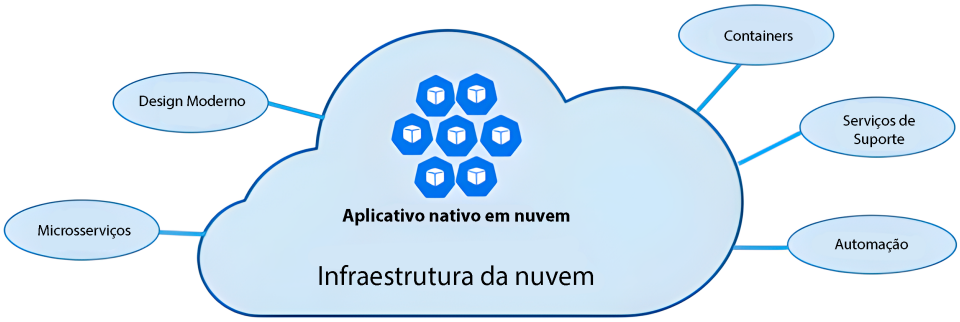
Cloud native é uma mudança de paradigma no design e desenvolvimento de sistemas empresariais, focando em velocidade, agilidade e escalabilidade. Ele permite que as organizações transformem seus aplicativos em ativos estratégicos que aceleram o crescimento e a inovação dos negócios. Aplicações nativas em nuvem são caracterizadas pela sua autonomia e agilidade, alcançadas por meio da encapsulação de serviços em contêineres autônomos e leves. Estes contêineres, como os contêineres Docker, garantem que o aplicativo e suas dependências estejam isolados da infraestrutura subjacente. Essa contêinerização possibilita a implantação perfeita em diferentes ambientes, desde que eles suportem o mecanismo de execução de contêineres. Ao aproveitar essa abordagem, as aplicações *cloud native* obtêm portabilidade e a capacidade de escalar rapidamente em resposta a flutuações na demanda. (ORACLE, 2022)

Esse conceito enfatiza a necessidade de uma resposta rápida às condições do mercado e às demandas dos usuários. Sistemas tradicionais com problemas de desempenho e atualizações lentas não são mais aceitáveis. Sistemas *cloud native*, como exemplificado por empresas como Netflix, Uber e WeChat, aproveitam um estilo arquitetônico que permite

rápida adaptação e alta resiliência em grande escala. Esses sistemas consistem em muitos serviços independentes que podem ser atualizados individualmente sem exigir o *redeploy* completo da aplicação.

O *cloud native* obtém sua velocidade e agilidade de diversos pilares fundamentais. Embora a infraestrutura em nuvem desempenhem um papel crucial, existem outros cinco pilares que contribuem para seu sucesso: microsserviços, design moderno, containers, serviços de suporte e automação. (VETTOR; SMITH, 2023)

Figura 3 – Pilares fundamentais do *cloud native*.



Fonte: Adaptado de (VETTOR; SMITH, 2023)

Microsserviços já foram definidos em 2.1.2, portanto, seguiremos para os outros pilares.

2.1.3.1 *Design* Moderno

No âmbito das aplicações *cloud native*, os princípios de *design* moderno desempenham um papel fundamental na formação da arquitetura e operações. Aderir à metodologia *Twelve-Factor Application* fornece uma base sólida para a construção de aplicativos *cloud native* otimizados para ambientes de nuvem contemporâneos. Essa metodologia abrange aspectos essenciais, como manter um único código-base para cada microsserviço, isolar e empacotar dependências, externalizar configurações, desacoplar serviços de suporte, impor separação estrita entre as etapas de construção, lançamento e execução, isolar microsserviços em processos separados, microsserviços autocontidos com suas próprias interfaces e portas, escalabilidade horizontal por meio de concorrência, instâncias de serviço descartáveis, manter paridade ambiental, registro orientado a eventos e executar tarefas administrativas separadamente. Além dos fatores originais do *Twelve-Factor*, o autor Kevin Hoffman introduz três fatores adicionais em seu livro "*Beyond the Twelve-Factor App*", enfatizando a importância de uma abordagem API-*first*, telemetria abrangente para monitoramento e coleta de dados de saúde, e implementação de mecanismos de autenticação e autorização robustos desde o início. Juntos, esses princípios de *design* moderno capacitam as aplicações *cloud native* a implantar, dimensionar e se adaptar rapidamente às mudanças dinâmicas do mercado.

2.1.3.2 *Containers*

Containers são um componente fundamental dos sistemas *cloud native*. Eles permitem o empacotamento de código, dependências e tempo de execução em uma única unidade portátil chamada imagem de container. Os *containers* oferecem consistência e portabilidade em diferentes ambientes, permitindo implantação e escalabilidade fáceis. Ferramentas de orquestração de *containers*, como o Docker, facilitam o gerenciamento de serviços containerizados em grande escala.

2.1.3.3 Serviços de Suporte

Os sistemas *cloud native* dependem de diversos recursos auxiliares conhecidos como serviços de suporte. Isso inclui armazenamento de dados, corretores de mensagens, serviços de monitoramento e serviços de identidade. Os provedores de nuvem oferecem serviços de suporte gerenciados, eliminando a necessidade das organizações provisionarem e gerenciarem esses recursos por conta própria. Tratar os serviços de suporte como recursos externos com configuração externa permite flexibilidade e intercambiabilidade.

2.1.3.4 Automação

A automação é um aspecto fundamental das práticas *cloud native*. A Infraestrutura como Código (IaC) permite a automação do provisionamento da plataforma e implantação de aplicativos. Existem ferramentas que permitem a criação de *scripts* declarativos para infraestrutura em nuvem. Ao automatizar a infraestrutura e as implantações, as organizações podem alcançar consistência, repetibilidade e escalabilidade. Sistemas de Integração Contínua e Entrega Contínua (CI/CD) desempenham um papel crucial na automação das etapas de compilação e liberação do desenvolvimento de aplicativos.

Em resumo, *cloud native* é uma abordagem transformadora que prioriza velocidade, agilidade e escalabilidade em sistemas empresariais. Ela aproveita microsserviços, princípios de design moderno, containers, serviços de suporte e automação para permitir inovação rápida e capacidade de resposta às demandas do mercado. Ao adotar práticas *cloud native*, as organizações podem desbloquear todo o potencial da nuvem e impulsionar sua transformação estratégica.

2.2 Conceitos de Criptografia

A criptografia é um processo que utiliza algoritmos matemáticos para transformar dados e informações em um formato que só pode ser lido se o leitor tiver uma "chave" específica. O objetivo da criptografia é garantir a confidencialidade, integridade e autenticidade das informações, protegendo-as contra acessos não autorizados ou alterações (AWS, 2023).

A criptografia é fundamental para a segurança da informação, pois permite que dados sensíveis sejam armazenados de forma segura e transmitidos por redes inseguras, como a internet. Ela é usada em uma variedade de aplicações, incluindo a proteção de dados em dispositivos de armazenamento, a autenticação de usuários, a proteção de transações financeiras online e a garantia da integridade de dados transmitidos ou armazenados (FORTINET, 2023).

Existem dois tipos principais de criptografia: a criptografia simétrica, que usa a mesma chave para criptografar e descriptografar os dados, e a criptografia assimétrica, que usa chaves diferentes para criptografar e descriptografar os dados.

Definiremos aqui, brevemente, alguns conceitos de criptografia que serão utilizados no desenvolvimento deste trabalho. O intuito deste trecho não é verificar a fundo provas matemáticas destes conceitos, apenas explicar a utilidade deles.

2.2.1 Assinaturas Digitais

A assinatura digital é um conceito que se baseia na criptografia e desempenha um papel crucial na validação de documentos e transações digitais. Ela é uma maneira de garantir a autenticidade de uma mensagem digital ou de um documento eletrônico. A assinatura digital fornece uma camada adicional de segurança, permitindo a verificação da identidade do remetente e confirmando que a mensagem não foi alterada durante o trânsito.

A assinatura digital utiliza a criptografia assimétrica, que envolve um par de chaves: uma chave privada e uma chave pública. A chave privada, que é mantida em segredo pelo proprietário, é usada para criar a assinatura digital, enquanto a chave pública, que é distribuída livremente, é usada para verificar a assinatura (GALVãO, 2023).

Quando uma mensagem é assinada digitalmente, o remetente da mensagem cria um resumo da mensagem (também conhecido como *hash*) e o criptografa com sua chave privada. Este *hash* criptografado é a assinatura digital. Quando o destinatário recebe a mensagem, ele pode usar a chave pública do remetente para descriptografar o *hash*. Se o *hash* descriptografado corresponder ao resumo da mensagem recebida, a assinatura é considerada válida, pois isso prova que a mensagem veio do remetente declarado e não foi alterada durante o trânsito.

A assinatura digital tem várias aplicações, incluindo a autenticação de identidade, a garantia de integridade de dados, a criação de contratos digitais legalmente vinculativos e a garantia de não repúdio, que é a garantia de que o remetente não pode negar a autenticidade da mensagem enviada (GOIáS, 2023).

Um exemplo de algoritmo de assinatura digital é o ECDSA (*Elliptic Curve Digital Signature Algorithm*). O ECDSA é uma versão do algoritmo de assinatura digital que

utiliza a criptografia de curva elíptica como seu algoritmo de chave pública. Para o ECDSA, uma curva elíptica é selecionada, que define o tamanho da chave e o nível de segurança. As assinaturas ECDSA são seguras devido à dificuldade de calcular logaritmos discretos no grupo de pontos na curva escolhida. O ECDSA tem várias aplicações, incluindo criptomoedas e certificados digitais (KARATI et al., 2014).

2.2.2 Certificados Digitais

Os certificados digitais são uma extensão do conceito de assinatura digital. Eles são documentos eletrônicos que contêm dados sobre a pessoa física ou jurídica que o utiliza, servindo como uma identidade virtual que confere validade jurídica e aspectos de segurança digital em transações digitais (SOUZA; NETO, 2023).

Um certificado digital é emitido por uma Autoridade Certificadora (AC), que é uma entidade confiável responsável por emitir, distribuir, renovar e revogar certificados digitais. A AC verifica a identidade do solicitante antes de emitir um certificado digital.

O certificado digital contém a chave pública do titular, além de outras informações como o nome do titular, a entidade emissora, a assinatura digital da AC e o prazo de validade do certificado. Quando alguém recebe um certificado digital, pode verificar a assinatura digital da AC para confirmar que o certificado é autêntico e não foi alterado desde que foi emitido.

Os certificados digitais são usados em várias aplicações, incluindo autenticação de sites, assinatura de código, criptografia de *e-mail*, VPNs e muito mais (CERTISIGN, 2023).

2.2.3 Conexão TLS

O TLS (*Transport Layer Security*) é um protocolo de segurança amplamente utilizado, criado para aumentar a privacidade e a segurança dos dados em comunicações pela internet (CLOUDFLARE, 2023b). O TLS permite que aplicações cliente/servidor se comuniquem de uma maneira que é projetada para prevenir a interceptação, adulteração ou falsificação de mensagens.

O TLS funciona entre a camada do protocolo de aplicativo e a camada TCP/IP, onde pode proteger e enviar dados do aplicativo para a camada de transporte. Como os protocolos funcionam entre a camada de aplicativo e a camada de transporte, o TLS pode dar suporte a vários protocolos de camada de aplicativo (LEARN, 2023).

O TLS utiliza criptografia para ocultar os dados transferidos de terceiros, autenticação para garantir que as partes que estão trocando informações são autênticas e integridade para verificar se os dados não foram falsificados ou adulterados.

Para usar o TLS, um site ou aplicativo precisa ter um certificado TLS instalado no servidor de origem. O certificado TLS é emitido por uma autoridade de certificação para a pessoa ou empresa dona de um domínio. O certificado contém dados importantes sobre quem é o proprietário do domínio, bem como a chave pública do servidor, duas informações essenciais para validar a identidade do servidor.

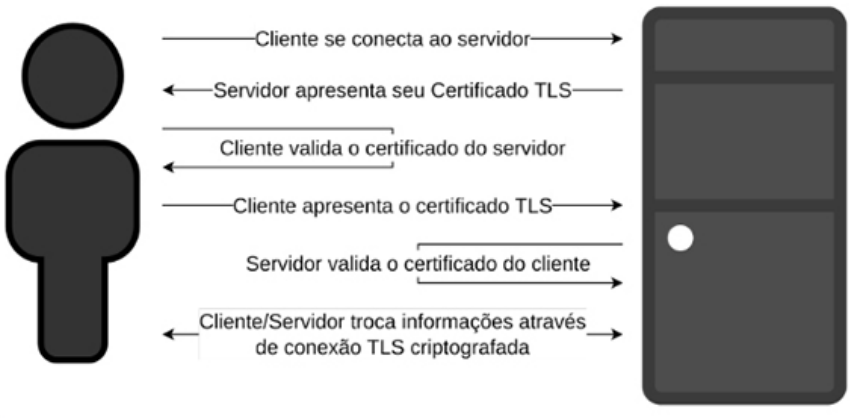
A versão 1.0 do protocolo TLS é especificada na RFC2246 (DIERKS; ALLEN, 1999). A RFC2246 descreve como o protocolo TLS funciona e fornece links para os RFCs IETF para TLS 1.0. A versão atual (1.3) do protocolo TLS é especificada na RFC8446 (RESCORLA, 2018).

2.2.3.1 mTLS

O mTLS (*Mutual Transport Layer Security*), também conhecido como TLS mútuo, é uma extensão do protocolo TLS que fornece autenticação bidirecional. No TLS padrão, apenas o servidor é autenticado ao cliente por meio de um certificado digital, enquanto no mTLS, tanto o cliente quanto o servidor são autenticados mutuamente (CLOUDFLARE, 2023a).

No mTLS, tanto o cliente quanto o servidor possuem um par de chaves pública/privada e um certificado digital. Durante o processo de *handshake*, tanto o cliente quanto o servidor apresentam seus certificados digitais, permitindo que ambos verifiquem a identidade um do outro. Isso garante que ambas as partes em uma conexão de rede são quem afirmam ser, verificando que ambas têm a chave privada correta.

Figura 4 – Exemplo de funcionamento do mTLS



Fonte: (GOCACHE, 2022)

O mTLS é frequentemente usado em uma estrutura de segurança *Zero Trust* para verificar usuários, dispositivos e servidores dentro de uma organização. Também pode ajudar a manter as APIs seguras.

2.3 SPIFFE

Entraremos agora numa melhor descrição do *framework* SPIFFE, e a *toolchain* SPIRE a ele associada.

Os conceitos aqui apresentados foram fortemente baseados no livro "*Solving The Bottom Turtle: A SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity*" (FELDMAN et al., 2020).

2.3.1 SPIFFE ID

O SPIFFE ID consiste no modelo de identificador que é utilizado no *framework*, devendo ser único para cada serviço.

Seu formato é parecido com uma URL, sendo um exemplo de SPIFFE ID o seguinte: *spiffe://example.com/my-service*

Destrinchando as partes do SPIFFE ID, pegando os componentes do exemplo anterior, temos

- spiffe:// - esquema de URI
- example.com - domínio de confiança (a definição mais precisa será coberta mais à frente)
- my-service - é a porção de nome do componente, devendo ser um identificador do *workload* em questão

Vale ressaltar que tudo que vier após o domínio de confiança será considerado parte do nome, sendo possível portanto, gerar qualquer nome que seja representativo para o domínio da aplicação em questão

2.3.2 *Trust Domain*

O *Trust Domain*, ou domínio de confiança, representado no exemplo anterior como a porção *example.com* do SPIFFE ID, serve para delimitar o escopo de um determinado ID, criando fronteiras entre organizações.

Essa delimitação é de grande importância quando há comunicação entre serviços de diferentes ambientes (como produção e homologação) ou diferentes organizações, cenários nos quais não há confiança plena entre as partes envolvidas na comunicação.

2.3.3 SPIFFE *Verifiable Identity Document* (SVID)

O SVID é um documento de identificação criptograficamente verificável utilizado para provar a identificação de um *workload* para outros serviços

Um SVID inclui um SPIFFE ID, e é assinado pela autoridade que representa o domínio de confiança contido nesse ID.

O documento utilizado para representar um SVID pode ser um certificado X509 (BOEYEN et al., 2008) ou um JWT. No caso do X509 (modelo mais importante para o contexto desse trabalho), basicamente temos um certificado X509 padrão, com o SPIFFE ID no lugar do campo da URI.

2.3.4 *Trust Bundle*

Um *Trust Bundle* é um conjunto de chaves públicas da autoridade de um determinado domínio de confiança. Esse documento é utilizado para confirmar a autenticidade de um SVID, dado que um SVID é assinado com a chave privada da autoridade.

É bom citar que o *Trust Bundle* não contém segredos dentro dele (dado que contém apenas chaves públicas). Entretando ele deve ser trafegado de maneira segura dentro do domínio para garantir que não seja modificado (é necessário integridade, mas não confidencialidade)

2.3.5 *Federation*

O conceito de *Federation* (federação) é necessário quando há comunicação entre mais de um domínio de confiança. Esse conceito representa basicamente a troca contínua de *Trust Bundles* entre esses diferentes domínios, a fim de que seja possível a comunicação entre eles, dentro do nível de confiança adequado.

A ideia do *federation* faz com que seja possível o reconhecimento de workloads de outros domínios de confiança, garantido pelo SPIFFE. Qual o nível de confiança que o domínio vai ter em identidades providas por outro domínio cabe ao contexto da aplicação.

2.3.6 *Workload* API

A *Workload* API consiste numa API que deve ser acessível a um *workload* localmente (deve esta alocada na mesmo recurso computacional do *workload*), capaz de prover a esse *workload* seus documentos de identificação SPIFFE.

Como essa API é usada para criar esse documento de identificação, espera-se que não haja uma identificação prévia do *workload* antes dele fazer uso da API.

2.4 SPIRE

SPIFFE *Runtime Environment*, ou SPIRE, é uma implementação do SPIFFE, contendo todas as 5 partes fundamentais que foram descritas anteriormente para o funcionamento adequado do *framework*.

Ele pode ser dividido em 2 partes, que serão detalhadas nas subseções abaixo.

2.4.1 SPIRE *Server*

O Servidor SPIRE é a parte do SPIRE responsável por providenciar identidades dentro de um domínio de confiança, e distribuir um *Trust Bundle* para os *workloads* dentro desse domínio.

Ele que detém a chave privada da autoridade daquele domínio, e deve também ter um banco de dados próprio no qual são guardadas informações sobre os *workloads* e suas identidades.

O SPIRE server pode ser controlado via API ou comandos de linha de comando (CLI).

2.4.2 SPIRE *Agent*

O SPIRE Agent é a porção do SPIRE responsável por providenciar a *Workload* API para um determinado *workload*. Ele deve, portanto, estar rodando no mesmo node do(s) *workload(s)* que for servir.

O SPIRE *Agent* é o responsável por comunicar-se com o SPIRE *server* para requisitar documentos de identidade, e de conseguir identificar um *workload*, através de um processo conhecido como *Attestation*.

2.5 *Attested Claims*

Juntaremos aqui referências a pesquisas do grupo *Attested Claims*. Algumas serviram como base para a implementação do *ID-Mode*, enquanto outras serão conceitos utilizados na Prova de Conceito; portanto é essencial o entendimento dos tópicos aqui citados para a compreensão do desenvolvimento deste trabalho.

Note que aqui já estamos tratando de pesquisas, não de mecanismos que já estão em uso. Os conceitos aqui apresentados não são parte do SPIFFE em si, mas propostas de melhorias no *framework* encontradas pelo grupo de pesquisa.

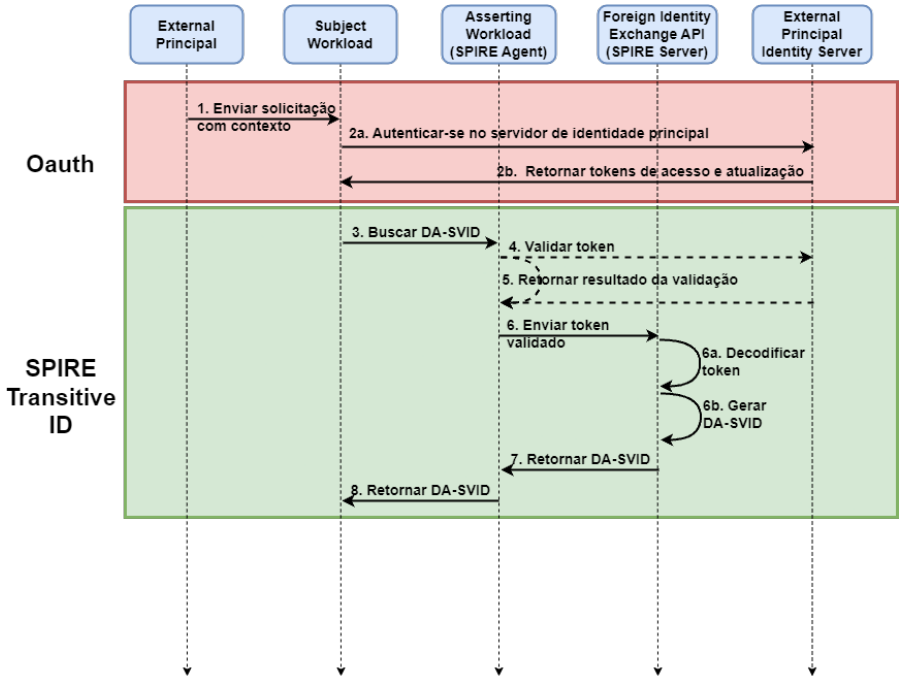
2.5.1 DA-SVID

O DA-SVID é uma extensão para a especificação do SPIFFE, que é um conjunto de padrões abertos para a identificação segura de *software*. O SPIFFE fornece uma estrutura para identificar e proteger as comunicações entre os serviços de *software*, independentemente da infraestrutura subjacente.

O DA-SVID, ou Documento de Identidade Verificável SPIFFE Delegado, é uma implementação específica dessa estrutura. Ele é um *token* JWT, que é um padrão aberto para a transmissão segura de informações entre partes como um objeto JSON. Este *token* é assinado digitalmente usando a chave privada de uma *Asserting Workload*, que é uma *workload* identificada pelo SPIFFE que faz o papel de provedor local de identidades.

O processo de emissão do DA-SVID é iniciado quando um usuário faz uma requisição e apresenta seu *bearer token* a uma *workload* Front-End. Esta workload então apresenta o *bearer token* à *workload* IDP, que é a API não-nativa do SPIRE e a raiz de confiança para esta aplicação. A emissão do DA-SVID ocorre quando o IDP decompõe o *bearer token* e emite o DA-SVID, usando o SPIFFE-ID da *workload Front-End* como sujeito e o SPIFFE-ID do IDP como emissor.

Figura 5 – Fluxo do DA-SVID



Fonte: Adaptado de SPIFFE-IdT *Working Group*, 2022

O DA-SVID contém várias informações importantes. Ele inclui detalhes sobre o usuário, como nome de usuário e *e-mail*, bem como informações sobre a *workload* que receberá as permissões delegadas. A validação do DA-SVID envolve a verificação da credencial emitida pelo provedor de identidade externo, a validação do certificado

X.509 (BOEYEN et al., 2008) e a validação da cadeia de certificação, que é um grupo de certificados obtidos pelo domínio de confiança.

Apesar de ser um *token*, o DA-SVID não é considerado um *bearer token*. Quando apresentado sozinho, ele não autentica o portador. Em vez disso, ele é usado para fornecer afirmações adicionais a uma parte já autenticada, expandindo os direitos existentes concedidos à *Subject Workload* pela *workload* que recebe o DA-SVID, chamada de *Target Workload*.

Comparando o DA-SVID com outros *tokens*, ele se destaca por ser usado tanto para autorização quanto para autenticação. No entanto, ao contrário de alguns outros *tokens*, o DA-SVID não permite o aninhamento de *tokens* nem assinaturas encadeadas.

Em resumo, o DA-SVID é uma solução robusta e recente que busca proteger informações sigilosas do usuário, evitando a propagação interna de credenciais e oferecendo uma forma segura de identificar o usuário sob o qual uma determinada *workload* está atuando. Ele é uma parte importante do ecossistema SPIFFE e desempenha um papel crucial na segurança das comunicações entre os serviços de software.

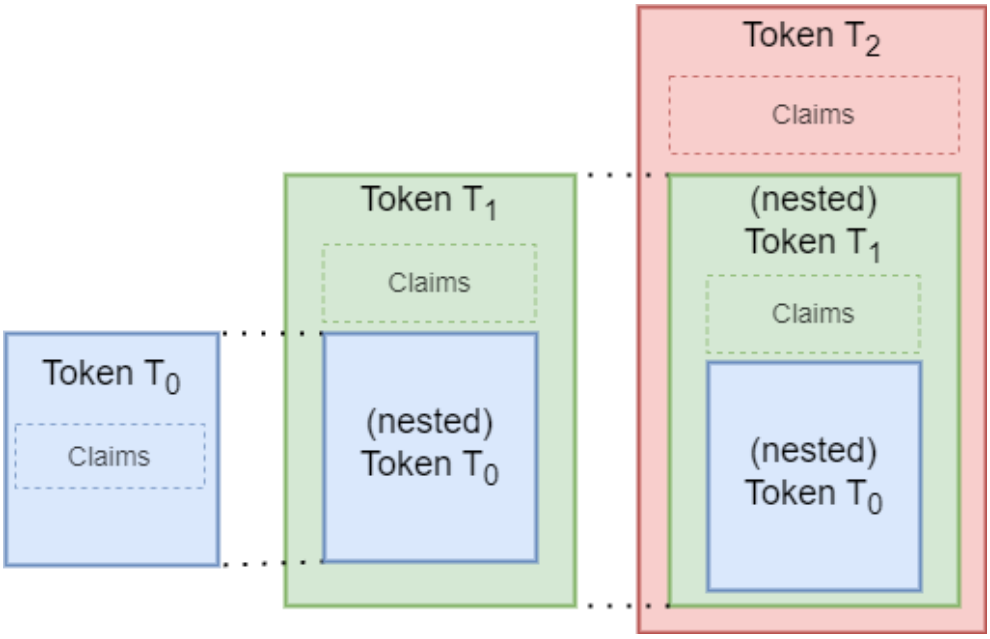
2.5.2 *Nested Tokens*

Os "*Nested Tokens*", ou *tokens* aninhados, são uma estrutura de dados avançada que desempenham um papel crucial na transmissão de informações de uma entidade para outra. Essas informações podem variar de asserções textuais simples a estruturas mais complexas. Um aspecto único dos *tokens* aninhados é que eles podem ser estendidos pelo seu detentor. Isso é feito através da adição de informações adicionais ao *token*, criando assim uma estrutura aninhada. Esta capacidade de extensão permite que os *tokens* aninhados sejam adaptados e personalizados para atender a uma variedade de necessidades e aplicações. Esta abordagem, também adotada em soluções como *Biscuits* (BISCUIT, s.d.) e *Macaroons* (BIRGISSON et al., 2014), permite a construção de um sistema de gerenciamento de acesso mais seguro e flexível do que o que pode ser obtido tradicionalmente com *tokens* regulares.

A estrutura de um *token* é composta por duas partes principais: o *payload* e a assinatura. O *payload* é a seção do *token* que contém as asserções. As asserções no payload podem ser independentes, ou seja, podem existir por si mesmas, independentemente de qualquer outro elemento no mesmo *payload*. Alternativamente, uma asserção pode complementar outra asserção no mesmo *payload*, bem como asserções de *tokens* aninhados.

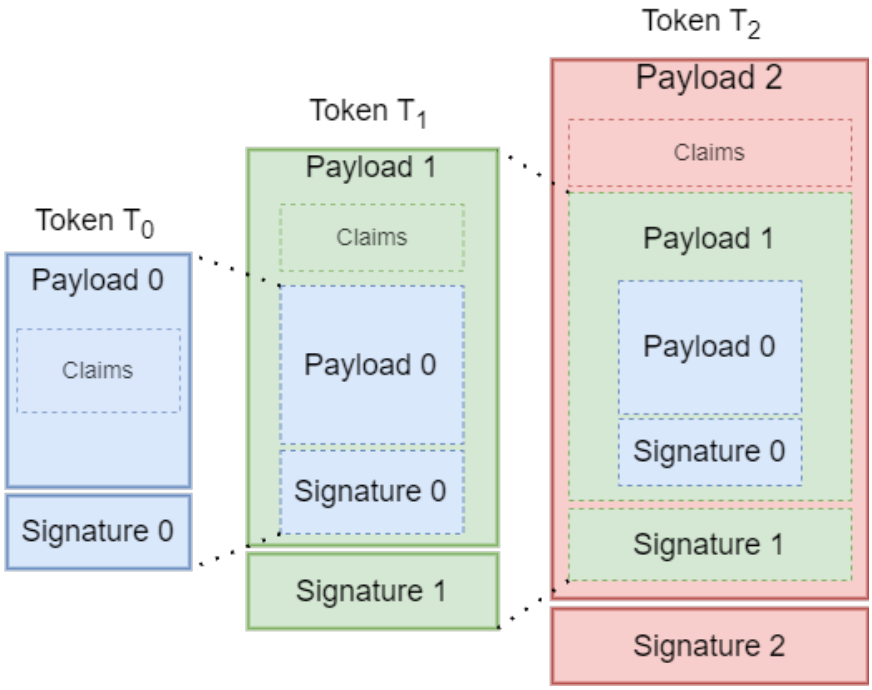
A seção de assinatura é composta pela assinatura digital do *payload*. Esta assinatura é gerada usando um algoritmo criptográfico adequado. A inclusão de uma assinatura digital em um *token* é fundamental para garantir sua autenticidade e integridade. Isso impede qualquer modificação ou adulteração não autorizada do *token*. A assinatura serve como uma prova de que o *token* foi emitido por uma entidade autorizada e que não foi alterado

Figura 6 – Exemplo de funcionamento dos *Nested Tokens*



Fonte: Adaptado de SPIFFE-IdT *Working Group*, 2022

Figura 7 – Estrutura dos *Nested Tokens*



Fonte: Adaptado de SPIFFE-IdT *Working Group*, 2022

desde a sua emissão.

O objetivo de incluir uma assinatura digital em um *token* é garantir sua autenticidade e integridade, prevenindo qualquer modificação ou adulteração não autorizada. A estrutura de duas partes do esquema de *token* aninhado proposto permite o uso de muitos algoritmos de assinatura diferentes. Além disso, para facilitar a administração

das possíveis múltiplas assinaturas incluídas em um *token*, prevemos a possibilidade de empregar várias técnicas, incluindo: verificação de assinatura em lote (KITTUR; PAIS, 2017), para reduzir o custo total geral da validação do *token*; assinaturas concatenadas, para reduzir o tamanho total ocupado pelos campos de assinatura quando os emissores são permitidos ser simplesmente pseudônimos (ABE; OKAMOTO, 1999) (PETERSENL; HORSTER, 1997); assinaturas agregadas, para reduzir ainda mais o tamanho ocupado pelos campos de assinatura (CHEN; ZHAO, 2022).

A criação de um novo *token* começa com a definição das asserções que ele deve incluir. Uma vez definidas as asserções, o *payload* que contém essas asserções é assinado. Os parâmetros mínimos necessários para criar um novo *token* incluem o *payload*, que deve conter todas as asserções definidas como obrigatórias, o algoritmo de assinatura a ser usado e a chave privada do signatário, necessária para criar uma assinatura válida sobre todo o *payload*.

A extensão de um *token* envolve a adição de novas informações a um *token* existente. Isso é feito criando um novo *payload* e, em seguida, assinando o resultado. O resultado é a criação de um novo *token* que inclui o *token* anterior. Este *token* anterior é então chamado de *token* aninhado.

O consumo de um *token* por um receptor envolve a validação da assinatura do *token* e do documento de identidade do signatário. Uma vez que o *token* é consumido, as ações subsequentes podem incluir o descarte do *token*, o encaminhamento do *token* para outra carga de trabalho sem qualquer modificação, ou a extensão do *token* com asserções adicionais e, em seguida, o encaminhamento para outra carga de trabalho.

Além da assinatura, outra definição importante ao lidar com *tokens* refere-se ao gerenciamento de identidade. Isso se refere a como a identidade de um emissor de *token* é verificada, e até mesmo se as identidades realmente precisam ser verificadas. Em muitos casos, a estratégia comum é confiar em um Provedor de Identidade confiável, que pode emitir certificados para entidades válidas do sistema.

2.5.2.1 Asserções Obrigatórias dos *Nested Tokens*

O modelo proposto estabelece algumas asserções essenciais, que são fundamentais para garantir a validação adequada dos *tokens* dentro de uma estrutura aninhada. Além disso, o modelo foi concebido com uma flexibilidade considerável em termos de quais asserções podem ser incorporadas em um *token* e como essas asserções são interpretadas. Ou seja, partimos do princípio de que uma asserção pode ser qualquer afirmação feita pelo emissor, normalmente seguindo um formato de (tipo, valor), embora outras formas de representação também sejam possíveis. Assim, as aplicações têm total liberdade para criar suas próprias asserções, adaptando-as conforme o caso de uso pretendido.

A seguir, mostraremos quais são as asserções obrigatórias nos *tokens* aninhados:

- issuer (*iss*): uma string usada na identificação do emissor. Pode ser um nome comum (CN) respaldado por um certificado fornecido ao verificador juntamente com o *token* ou por meio de um IdP, uma chave pública (PK) que permite a verificação do campo de assinatura mesmo que o sistema não dependa de certificados para associar identidades a chaves públicas, ou um documento de identidade (ID) que permite ao verificador extrair o documento incorporado para verificar a identidade e a assinatura do signatário.
- issuing time (*iat*): um timestamp que indica quando o *token* foi emitido. É útil para reduzir a superfície de ataque, pois as aplicações podem definir uma janela de tempo aceitável dentro da qual o *token* é considerado inválido.
- algorithm (*alg*): uma string para identificar o algoritmo usado na assinatura, como definido na RFC7518 (JONES, 2015).

Essas soluções propostas serão melhor abordadas e aprofundadas posteriormente no capítulo 5.

3 Método do trabalho

Este trabalho será realizado em diversas fases, que serão melhores descritas neste capítulo.

3.1 Pesquisa e Compreensão

Dado que o trabalho situa-se dentro de um campo bastante técnico, utilizando como base um *framework* que não é tão conhecido, a fase inicial desse trabalho irá consistir em uma pesquisa exploratória, visando a familiarização com o SPIFFE e o SPIRE.

Também cabe a essa fase do projeto delimitar melhor o escopo de pesquisa, visando definir o objetivo do projeto, especialmente em questão a qual melhoria poderia ser feita no protocolo SPIFFE para que ele possa abranger um escopo mais diverso

3.2 Especificação de Requisitos

Cabe a essa fase definir os requisitos da solução que será buscada através desse trabalho. Ou seja, cabe aqui pensar a respeito do problema formulado anteriormente, e tentar especificar de maneira mais direta quais são os requisitos de uma solução para esse problema.

3.3 Projeto da Solução

Aqui iremos projetar uma solução possível para o problema endereçado. Sendo assim, após compreender o problema e os requisitos de uma solução, aqui cabe de fato a elaboração de uma solução prática que atenda os requisitos funcionais anteriormente listados.

Nesta etapa também serão discutidas questões técnicas da solução, como arquitetura, tecnologias a serem utilizadas, etc.

É importante levar em conta durante essa fase todos os requisitos previamente descritos, dada que essa é a parte do projeto com maior responsabilidade de garantir que a solução obedeça aos requisitos desejados.

3.4 Implementação do *ID-Mode*

Aqui será o início da parte prática do projeto. O intuito desta fase do trabalho será criar na prática o projeto anteriormente descrito, atentando-se especialmente a cumprir os requisitos não funcionais listados. Nesse caso, essa fase do projeto consistirá em criar um código que cumpra as especificações do projeto e que possa ser acoplado a uma prova de conceito.

De maneira prática, então, serão criados códigos em Go que dêem suporte à criação de *tokens* aninhados e que implementem as restrições específicas do *framework* do *ID-Mode*

3.5 Criação da Prova de Conceito

Na fase final deste projeto, serão colocados em prática os códigos de suporte do *ID-Mode*, ao criar uma prova de conceito que envolva diversos *containers*; simulando uma aplicação em nuvem distribuída, na qual seja possível realizar autenticação delegada através de um *token* OAuth, utilizando de asserções criadas no *framework* do *ID-Mode*.

4 Especificação de Requisitos

Neste capítulo serão detalhados os requisitos da solução proposta para que ela possa cumprir o seu propósito.

Para isso cabe revisar aqui como é a situação de uma aplicação em nuvem, utilizando de microsserviços, e que já faz uso do SPIRE. Nesse cenário, considerando que dentro de um domínio de federação, há uma identidade para cada *workload* dentro dessa aplicação garantida pelo SPIRE server (que assumimos que é confiável).

4.1 Requisitos Funcionais do *Framework*

- Uso das identidades providas pelo SPIRE *Server*: o primeiro requisito para implementar o projeto então seria que a solução proposta fizesse uso das identidades providas pelo SPIRE *server* para assinarem uma asserção, dado que os *workloads* já confiam nessas identidades
- Independência de Plataforma: a solução proposta, tal qual o SPIFFE/SPIRE deve ser agnóstico a plataforma, funcionando em qualquer provedor de nuvem.
- Integração entre plataformas: novamente tal qual o SPIFFE/SPIRE, a solução buscada deve conseguir realizar a integração entre *workloads* que estejam abrigados em plataformas diferentes, respeitando os domínios de confiança do SPIFFE.
- *Zero Trust*: a solução buscada deve considerar que nem todos os *workloads* identificados são, necessariamente confiáveis, e portanto, deve impedir a possibilidade de um agente alterar os conteúdos da asserção de outro componente da aplicação.
- Rastreabilidade das asserções: deve-se ter a possibilidade de conseguir rastrear com garantia criptográfica quem foi o responsável por uma asserção. Isso pois apesar de haver confiança na identidade dos demais *workloads*, não necessariamente há a confiança de qualquer asserção que seja feita por eles.
- Requisitos de segurança: o ponto principal de requisito do *framework* é que ele seja bastante seguro, dado que o maior incentivo para a sua criação pe evitar erros na hora de deixar lógicas de autorização e delegação de identidades apenas a nível de aplicação

4.2 Requisitos de Segurança

Como o projeto é fortemente ligado a área de segurança da informação, estarão separados aqui requisitos específicos de segurança para o modelo criado.

- Irretratabilidade nas asserções feitas: deve ser possível conferir para qualquer asserção, de maneira irretratável, a carga que realizou aquela asserção.
- Resistência a tentativas de modificar o *token*: o *token* com as asserções deve ser resistente a tentativas de ataque que tentem modificar ou remover alguma parte sua.
- Resistêcia a ataques de reutilização: o modelo criado deve ter maneiras de previnir ataques de reutilização de um *token* que foi emitido, para que um atacante que consiga acesso a um *token* que acabou vazando não consiga fazer utilização dele.

4.3 Requisitos não Funcionais do *Framework*

- Integração simples com o SPIRE: é importante que o projeto consiga ter uma integração direta com o SPIRE, para que cumpra seu propósito de agregar ao SPIRE novos casos de uso que hoje não estão dentro de seu escopo.
- Uso computacional: é desejado que o uso computacional para as operações criptográficas previamente citadas não seja de custo muito alto. Isso porque como essas operações serão realizadas na e autenticação entre *workloads*, e não são necessariamente dependentes de ações humanas diretas, em alguns casos de uso elas podem ocorrer em grande quantidade.
- Operação simples: é desejado que a solução não dificulte muito o trabalho de operação da aplicação que fizer uso dela, para que sua implantação em possíveis ambientes não leve muito tempo ou esteja sujeita a muitos erros.
- Alta flexibilidade: apesar da prova de conceito que será realizada com o *framework* em questão, é imoportante que ele seja bastante flexível, e que possa ser utilizado de maneira simples para aplicações que desejam aplicar as asserções a algum outro escopo (não necessariamente o de delegação de identidade).
- Testes de segurança: é desejado também que o projeto passe por testes de segurança, testando possíveis ataques ao modelo criado a partir de entidades não confiadas. Por ser um projeto intimamente ligado à segurança, faz se bastante necessário a confiança de que ele tem uma robustez contra possíveis ataques.

4.4 Requisitos da Prova de Conceito

- Simulação de ambiente de microsserviços: uma das exigências, quando se trata da etapa da prova de conceito, é que ela seja uma simulação bastante confiável de um ambiente em nuvem, especificamente um ambiente de microsserviços.
- Autenticação delegada: como descrito anteriormente, a prova consistirá num usuário que faz *login* através de um provedor de identidade.

5 Desenvolvimento do Trabalho

5.1 Tecnologias Utilizadas

5.1.1 Golang

A linguagem de programação Go já é amplamente utilizada, em especial em trabalhos que têm relação com ambientes em nuvem e redes distribuídas. A linguagem foi criada pela Google em 2007, e teve seu código aberto em 2009. Além de ser uma excelente ferramenta, a linguagem foi adotada para o projeto principalmente pois o próprio SPIRE é feito em Go, então uma possível integração do projeto ao SPIRE ocorreria de maneira muito mais fácil com esse projeto já feito nesta linguagem de programação.

A linguagem Go foi utilizada tanto para criar as funções essenciais do *framework ID-Mode* quanto para criar a prova de conceito

5.1.2 Docker

Para simular uma aplicação distribuída na prova de conceito, foi utilizada da ferramenta Docker, de criação de *containers*. Isso é útil pois, primeiramente, o isolamento dos *containers* faz uma boa simulação de um ambiente distribuído em nuvem; ainda mais pelo fato de que mesmo em nuvem, muitas vezes ambientes distribuídos e aplicações em microsserviços rodam em *containers*, fazendo que a simulação seja bastante parecida com o ambiente alvo.

Ainda, o docker teoricamente gera uma facilidade no teste da prova de conceito, já que a aplicação rodando em docker evita alguns problemas de compatibilidade com a máquina local

Como o intuito da prova é simular diversos *containers*, foi utilizado, junto ao docker, a ferramenta docker-compose. Ela é uma ferramente do próprio Docker feita justamente para a criação de múltiplos *containers* conectados, então se mostrou bastante útil

5.1.3 Okta

A Okta é uma plataforma de identidade, que serve para diversos fins relacionados a autenticação e autorização. Para o fim deste trabalho, ela será utilizada na prova de conceito, para prover um *token* OAuth; que será utilizado no sistema como meio de autenticação do usuário.

5.2 O *ID-Mode*

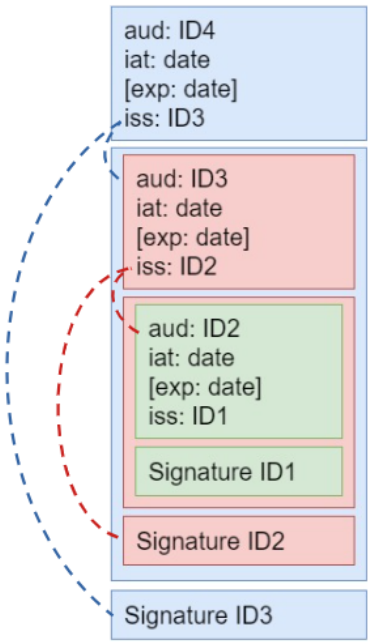
Nesta seção do trabalho definiremos o *framework* criado. Tomando como base, em especial, o modelo de *Nested Tokens*, definido no capítulo 2, definiremos as características de um modo de uso que implementa os requisitos desejados e definidos no capítulo 4.

5.2.1 A Concepção

Descrevendo de maneira geral o formato do *framework ID-Mode*, ele consiste em uma aplicação prática do modelo de *Nested Tokens* específica para um ambiente onde os *workloads* têm certificados que comprovam a sua identidade dentro do ambiente (como os fornecidos pelo SPIRE).

Esse uso então, adiciona uma claim obrigatória ao modelo de *Nested Tokens* que representa a identidade do próximo *workload*, chamada de *audience*; e realiza mTLS nas conexões entre os *workloads*, enquanto repassa todos os certificados das conexões anteriores junto ao *token* com as asserções.

Figura 8 – *ID-Mode Nested Tokens*



Fonte: Adaptado de SPIFFE-IdT *Working Group*, 2023

Ou seja, um fluxo normal de uso do *ID-Mode*, para *workloads* intermediários é o seguinte:

1. O *workload* recebe um *token*.
2. O *workload* confere se o valor de *audience*; no último *token* aninhado (ou seja, o *token* mais externo), corresponde a ele próprio.

3. O *workload* confere se o valor de issuer do último *token* corresponde ao identificador do *workload* com o qual está se realizando mtls.

4. O *workload* opcionalmente valida o *token* (processo a ser definido em seguida)

5. O *workload* adiciona asserções suas ao *token*; incluindo necessariamente, mas não se restringindo às asserções obrigatórias do modelo de *Nested Tokens*, e a asserção de *audience*.

6. O *workload* assina digitalmente todo o *token* (ou seja, o *token* anterior e as novas asserções), garantindo irretratabilidade às novas asserções, e à verificação por ele das asserções anteriores.

7. O *workload* passa adiante o *token*, para o *workload* representado pela identidade de *audience*.

8. O processo se repete para o próximo *workload*.

A validação do *token* consiste em, primeiramente, verificar se todos o valor de *audience* do *token* aninhado T[n] corresponde ao valor de *issuer* do *token* aninhado T[n+1]. Em seguida, verificar, se os certificados recebidos são válidos e as assinaturas estão corretas de acordo com a chave pública dos certificados. Este processo foi dado como opcional, pois é estritamente necessário somente para o último *workload* do fluxo, ou seja, aquele que irá de fato fazer uso do *token*; e não adicionar informações a ele.

Note que o processo do primeiro *workload* da cadeia é bem parecido com o processo descrito anteriormente, com excessão do fato de que ele deve criar o *token* inicial, portanto não há validações a serem feitas.

5.2.2 A Implementação

Enquanto a descrição anterior tentou ser geral quanto ao uso de tecnologias, aqui descreveremos de fato, uma sugestão de implementação mais direta, citando tegnologias mais específicas.

Consideraremos então, que o ambiente em questão é um ambiente que roda o SPIRE, e que os *workloads* possuem X509 SVID válido para representar a sua identidade.

Dentro desse contexto, os *workloads* utilizaram como valor de *audience*, o SPIFFE-ID do próximo *workload* a ser acionado no fluxo da aplicação, e como *issuer* o seu próprio SPIFFE-ID.

Cada *workload* assinará o *token* com o algoritmo ECDSA, utilizando de suas chaves privadas fornecidas pelo SPIRE; que poderão, então, ser verificadas por qualquer outro *workload* que possua seu SVID.

A conexão mTLS entre os *workloads* ocorrerá usando como identidade o X509 SVID, e ele será repassado junto ao *token* para possibilitar a validação do *Nested Token* pelo *workload* que for consumir suas asserções.

5.2.3 Resistência a Ataques

Este modo de uso do modelo de *Nested Tokens*, além de possibilitar todas as vantagens de possibilidades de asserções distribuídas, ainda é bastante seguro a diversos tipos de ataques. Serão resgatados aqui os ataques que se deseja previnir; analisados no capítulo 4, e o porquê do modelo do *ID-Mode*; se utilizado corretamente deve previní-los.

5.2.3.1 Ataques de Modificação do *Token*

Ataques visando modificar o conteúdo de um *token* podem ser úteis para tentar obter acessos que normalmente o *token* não daria.

Primeiramente, dado que toda conexão entre os *workloads* é feita através de mTLS, isso já dificulta o ataque. Isso porque desta maneira, este *token* deve ser acessível apenas aos *workloads* do fluxo da aplicação.

Entretanto, se um dos *workloads* do fluxo estiver corrompido, ou no caso de ocorrer algum vazamento do *token* por outros meios, é desejável que não seja possível alterar asserções feitas por outro agente. Vale ressaltar que no caso de um *workload* corrompido; parte da responsabilidade de evitar falhas de segurança é da aplicação, já que a interpretação e uso das asserções não é escopo do *framework*.

O primeiro caso, mais simples, seria um ataque com a tentativa de efetivamente modificar o conteúdo de um *token* anterior. Vamos supor que, na figura 9, haja a tentativa de alterar a *claim* de *audience* feita pelo ID1; de ID2 para ID3.

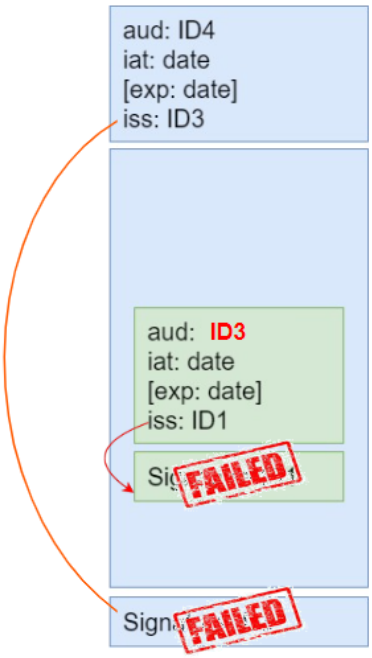
A validação do *token* irá falhar pois a assinatura de ID1 foi feita tendo em vista o *token* anterior; e portanto a verificação de assinatura irá falhar, mostrando que o *token* foi corrompido.

Um caso um pouco mais específico seria o caso de modificação tentando retirar alguma parte do *token*, ao invés de modificá-la. Por exemplo, retirar as asserções de algum *workload* específico dentro do *Nested Token*. Isso pode ser perigoso, pois essas asserções poderiam ser restritivas, e se um atacante conseguir retirá-las, ele poderia aumentar o acesso do *token*.

A proteção contra esse ataque divide-se em dois casos: a retirada de asserções do meio do *Nested Token*, e a retirada do último *token* aninhado.

O primeiro falha porque isso quebrará o link entre *issuer* e *audience* de algum ponto do *token*. Na figura 10, estamos supondo que houve a tentativa de retirar as asserções

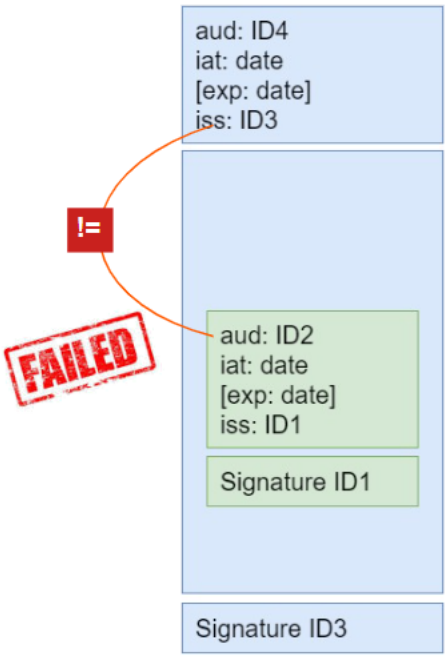
Figura 9 – Ataque de Modificação de *Token*



Fonte: Adaptado de SPIFFE-IdT *Working Group*, 2023

e assinatura do *workload* ID2 do *token*; e vemos que isso gera uma desconexão que impossibilita a validação do *token*. O segundo caso falha porque o *workload* a consumir o *token* poderá notar que o valor do *audience* não corresponde a ele mesmo.

Figura 10 – Ataque de Retirada de *Token*



Fonte: Adaptado de SPIFFE-IdT *Working Group*, 2023

5.2.3.2 Ataques de Reutilização

Outro tipo de ataque que o *framework* deve tentar impedir ou amenizar é o ataque de reutilização de *token*.

No caso do vazamento de um *token*; é desejável que não seja possível reutilizar este *token* para obter acesso a algum recurso.

Há algumas camadas de proteção para este tipo de ataque que o *ID-Mode* fornece:

- Troca de mensagens criptografada entre *workloads*: como é utilizado do protocolo mTLS na conexão entre os *workloads*, as informações trocadas por ele são criptografadas, dificultando o vazamento do *token* em primeiro lugar.
- Conferência da identidade da conexão mTLS: caso algum agente tente utilizar um *token* e ele não seja o agente suposto a fazer isso (ou seja, o *issuer* do *token* mais externo do *Nested Token*), será possível, através do certificado fornecido na conexão mTLS verificar que a conexão não está estabelecida com o agente correto.
- Marca temporal no *token*: o próprio modelo de *Nested Token* já estabelece uma asserção obrigatória de instante de tempo, e é esperado que a aplicação estabeleça um limite te tempo de limite de uso de uma asserção após sua criação, evitando a reutilização de *tokens* emitidos há muito tempo.

5.3 A Prova de Conceito

A prova de conceito do *framework* será uma aplciação extremamente simples, pois para esse teste a única importância é a etapa de *login* e a autenticação. A aplicação em si é apenas um "simulador de banco" onde é possível depositar valores (sem qualquer lastro financeiro real) e conferir o saldo atual (soma dos valores depositados) da conta logada. Para todos efeitos, a única regra de negócio é que deve se estar corretamente logado para depositar valores e também para conferir o saldo, e o saldo deve corresponder a todos os valores depositado.

A importância da aplicação está no fato de que o *login* será feito utilizando um *token* OAuth provisionado pela Okta, sem a necessidade, portanto, do usuário inserir dados de *login* (sendo necessário apenas que ele esteja corretamente logado na Okta).

Dentro desse cenário, cada um dos *workloads* descritos na imagem são *containers* criados utilizando de Docker; e localmente há um servidor SPIRE rodando que provisiona identidades para esses *workloads*. As funções de cada *workload* são as que seguem:

- *Front-end* (ou *subject workload*): provisiona a parte da aplicação de interação com o usuário, e recebe o *token* OAuth do usuário.

- *Local IDP* (ou *asserting workload*): funciona como um provedor de identidades local, recebendo o *token* Oauth do usuário e a partir dele criando o primeiro *token* dos *Nested Tokens*
- *Middle Tier*: o *middle-tier* apenas repassa a requisição sem regras de negócio. Foi criado para simular um ambiente mais distribuído. É possível, inclusive, rodar o projeto com mais *Middle-Tiers*.
- *Target*: o *target* é o servidor onde de fato a aplicação está rodando. Possui um arquivo que funciona como banco de dados, guardando informações de usuário e de balanço total dos depósitos.

O Fluxo normal da aplicação é, então, o seguinte:

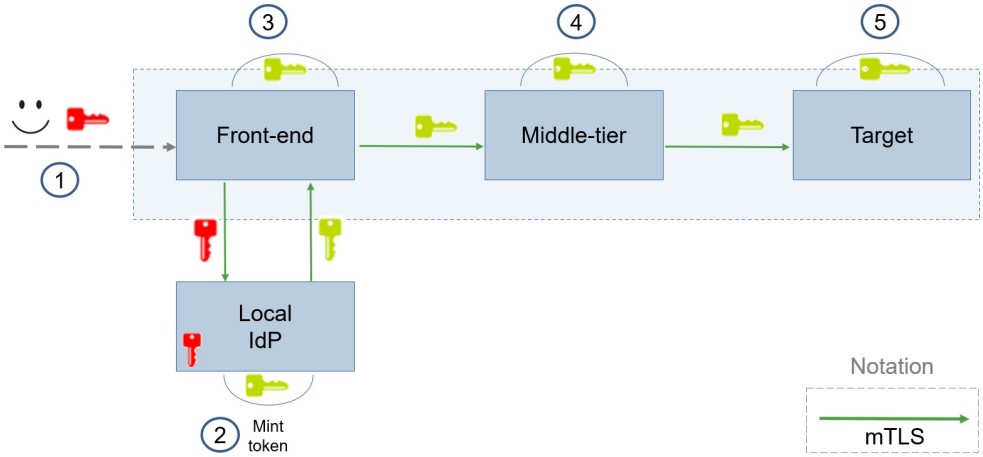
- O usuário realiza *login* na aplicação, fornecendo um *token* OAuth da Okta.
- O *front-end* envia o *token* para o provedor de identidades local, que troca esse *token* por um modelo mais restritivo (usando o modelo de DA-SVID, explicado no capítulo 2); e dá início ao *Nested Token*.
- O provedor de identidades retorna esse *token* para o *front-end*, junto ao seu certificado (para que seja possível futuramente validar o LSVID por inteiro).
- O *front-end* adiciona as asserções essenciais ao *token*, assina e repassa para o *Middle-Tier* junto ao seu certificado.
- Similarmente, o *Middle-Tier* repete o processo anterior feito pelo frontend, mas enviando para o *token* para o *Target*.
- *Target* realiza a validação do *token*, e retorna as informações do usuário para o *front-end*.

5.4 *Anonymous Mode*

O *Anonymous Mode*, como citado anteriormente, corresponde a outro modo de uso dos *Nested Tokens*, desenvolvido paralelamente ao *ID-Mode*, mas cujo desenvolvimento não fez parte deste trabalho.

Iremos fazer uma breve descrição do seu funcionamento, pois isso virá a ser útil nas conclusões deste trabalho para fins comparativos, mostrando os aspectos que o *Anonymous Mode* tem de vantajoso quando comparado ao *ID-Mode*; e também será de bastante utilidade para elencar possíveis trabalhos futuros.

Figura 11 – Fluxograma de funcionamento do *ID mode*



Fonte: SPIFFE-IdT *Working Group*, 2023

O *Anonymous Mode* consiste em um modo de uso dos *Nested Tokens* no qual as identidades dos *workloads* não precisam, necessariamente, serem conhecidas. Note que, desta maneira, o *Anonymous Mode* não está tão fortemente atrelado a um ambiente SPIFFE, onde identidades dos *workloads* são conhecidas. Ele expande o uso de *Nested Tokens* para um cenário mais genérico onde se deseje asserções distribuídas.

No *Anonymous Mode* é feito uso de uma ferramenta parecida com a ideia de *biscuits* (BISCUIT, s.d.). Basicamente, no fluxo normal deste modo, um *workload* que recebe o *Nested Token* cria um par de chaves público-privada para o próximo *workload*. A chave pública será anexada como identidade ao *token*, e a chave privada será passada para o próximo *workload*. Desta maneira, não há a necessidade de que um *workload* conheça a identidade do próximo *workload* no fluxo; dado que ele próprio irá gerar sua "identidade"(no caso a chave pública, que atuará como um identificador dali em diante).

Entretanto, apesar desta vantagem, esse esquema acaba sendo menos interessante no ponto de vista de casos de uso, dado que de maneira geral, se deseja ter conhecimento das identidades que fizeram cada uma das asserções. Desta maneira, este modo de uso acaba sendo mais difícil de se colocar em prática.

6 Considerações Finais

6.1 Conclusões do Projeto de Formatura

De maneira geral, podemos considerar que o trabalho cumpriu seus objetivos principais. *O ID-Mode*, como modo de uso dos *Nested Tokens* pode ser extremamente útil. Ele resolve em boa parte as questões relacionadas a asserções distribuídas dentro de um ambiente SPIFFE; e de maneira segura e confiável, evitando diversos tipos de ataques.

Claro que cabe dizer que o *ID-Mode* por si só é mais uma ferramenta do que uma solução. Sua utilidade é criar uma maneira de fazer asserções, mas ele não se responsabiliza pela natureza em si de tais asserções (ou seja, a interpretação e confiança cabe à camada de aplicação).

A parte final do trabalho, sendo esta a prova de conceito, demonstrou que a ideia do *ID-Mode* é praticável de fato no contexto de uma aplicação. Apesar dela não testar todas as possibilidades de uso do *ID-Mode* (até porque são várias possibilidades, dependendo de como uma aplicação quiser utilizar deste recurso), ela mostrou que ele é praticável num cenário bem parecido com uma boa parte dos ambientes que utilizam do SPIFFE.

De forma geral, concluí-se que este trabalho gerou modelos muito úteis dentro do contexto de asserções num ambiente SPIFFE que, poderiam vir a se tornar parte da própria especificação do *framework*, expandindo sua utilidade e dando mais possibilidades para os usuários.

6.2 Perspectivas de Continuidade

Aqui trataremos de trabalhos futuros possíveis partindo deste projeto. Alguns deles têm a atenção do grupo de pesquisa, enquanto outros dependem mais da comunidade SPIFFE como um todo para serem feitos.

6.2.1 Integração ao SPIFFE/SPIRE

Um dos caminhos de continuação deste trabalho, que caberia principalmente à comunidade SPIFFE/SPIRE como um todo, seria anexar os conceitos aqui apresentados à arquitetura do próprio *framework*.

Note que o trabalho feito aqui foi feito em um código separado ao do SPIRE. Apesar da prova de conceito simular um ambiente SPIRE, toda a lógica de *Nested Tokens* e do *ID-Mode* foram feitas como se fossem regra da aplicação em si.

Caberia então, à desenvolvedores da comunidade SPIFFE/SPIRE, se assim tiverem interesse, analisar o código gerado e, se acharem útil, integrá-lo à especificação do SPIFFE e ao código do SPIRE; mediante alterações e melhorias que acharem necessárias. Note que o projeto foi feito na linguagem Go (a linguagem na qual o próprio SPIRE é escrito), e é de código aberto.

Outra possibilidade, caso a comunidade SPIFFE conclua que o contexto do *ID-Mode* não deve ser do escopo do *framework*, seria a criação de um *framework* específico, a partir da prova de conceito, que permita a criação de tokens e asserções através do *ID-Mode*. Cabe aqui observar que todo o código do projeto consiste numa prova de conceito, então não está apropriado para ser utilizado de fato por uma aplicação em produção; mas pode ser utilizado de base para a construção de uma biblioteca que implemente o *ID-Mode* de modo a possibilitar este uso.

6.2.2 *Double Mode*

Para esclarecer a ideia do *Double Mode*, é importante notar que o *ID-Mode* possui uma questão quanto a sua natureza em si: seu funcionamento depende que uma carga de trabalho sempre conheça a identidade da próxima carga de trabalho no fluxo de uma aplicação (para saber o valor do *audience*)

Por mais que isso talvez pareça natural, comumente ambientes em nuvem possuem balanceadores de carga na frente de uma série de *workloads*, e muitas vezes um microsserviço sabe apenas o endereço do balanceador de carga que representa o próximo microsserviço que ele deve acessar em um fluxo, sem ter uma possibilidade de identificar qual *workload* em específico será acionado pelo balanceador de carga. Em casos como o descrito, o *ID-Mode* não funciona corretamente.

Por outro lado, o *Anonymous Mode*; que numa visão geral tem uma utilidade menos prática do que o *ID-Mode* (note que na maioria dos casos se deseja saber com certeza qual a carga de trabalho que fez uma determinada asserção para poder saber sua validade), funciona corretamente mesmo dentro dessas restrições.

A ideia do *Double Mode* então, é um trabalho em progresso para tentar juntar as características úteis de ambos os modos de uso apresentados. Ele se mostra útil num cenário no qual os *workloads* saibam a identidade uns dos outros por meio de SPIFFE; mas as requisições muitas vezes sejam feitas através de load balancers, e não diretamente para o próximo *workload*.

Desta maneira, o *Double Mode* poderia tentar juntar ideias de ambos os modos de funcionamento: cada *workload* assina um token com sua chave privada SPIFFE (como no *ID-Mode*) e com uma chave criada pelo *workload* anterior e fornecida a ele (como no *Anonymous Mode*). Ambas as identidades do *workload* (a identidade SPIFFE e a identidade

fornecida pelo *workload* anterior) são especificadas no token. Desta maneira, tem-se uma identidade confiável de qual *workload* fez cada asserção, mas não há a necessidade tão direta de saber qual o próximo *workload* do fluxo.

Note que o *Double Mode* por enquanto é apenas uma concepção, e ainda não foi efetivamente colocado em prática nem foi claramente especificado como *framework*; mas pesquisas nessa direção poderiam render frutos interessantes.

6.3 Contribuições

Como já citado, este trabalho foi realizado como parte de um projeto maior. Houve, portanto, ajuda de outros membros do grupo de pesquisa para a sua realização, bem como houve ajuda dos realizadores deste trabalho para com o restante do grupo de pesquisa em diversas outras frentes do projeto. Por ser um projeto de código aberto, tabém houveram contribuições da comunidade SPIFFE, bem como da HPE, uma das principais colaboradoras do projeto.

Enquanto os pesquisadores do grupo deram suporte para a construção teórica e prática do *ID-Mode*, também houve ajuda por parte dos realizadores deste trabalho em partes do projeto que não são diretamente deste escopo, bem como o modo de uso *Anonymous Mode*, e os passos iniciais de concepção do *Double Mode*.

Referências

ABE, M.; OKAMOTO, T. Delegation chains secure up to constant length. In: VARADHARAJAN, V.; MU, Y. (Ed.). *Information and Communication Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999. p. 144–156. ISBN 978-3-540-47942-0. Citado na página 30.

AWS, A. W. S. *O que é criptografia?* 2023. Disponível em: <https://aws.amazon.com/pt/what-is/cryptography/?nc1=h_ls>. Citado na página 20.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Migrating to cloud-native architectures using microservices: An experience report. In: CELESTI, A.; LEITNER, P. (Ed.). *Advances in Service-Oriented and Cloud Computing*. Cham: Springer International Publishing, 2016. p. 201–215. ISBN 978-3-319-33313-7. Citado na página 10.

BIRGISSON, A. et al. *Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud*. 2014. Disponível em: <https://doi.org/10.14722/ndss.2014.23212>. Citado na página 28.

BISCUIT. *Biscuit Documentation*. s.d. Disponível em: <https://doc.biscuitsec.org/reference/cryptography.html>. Citado 2 vezes nas páginas 28 e 44.

BLINOWSKI, G.; OJDOWSKA, A.; PRZYBYLEK, A. Monolithic vs. microservice architecture: A performance and scalability evaluation. *IEEE Access*, v. 10, p. 20357–20374, Feb 2022. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9717259>. Citado na página 16.

BOEYEN, S. et al. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC Editor, 2008. RFC 5280. (Request for Comments, 5280). Disponível em: <https://www.rfc-editor.org/info/rfc5280>. Citado 2 vezes nas páginas 25 e 28.

CAMPBELL, B. et al. *Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants*. RFC Editor, 2015. RFC 7521. (Request for Comments, 7521). Disponível em: <https://www.rfc-editor.org/info/rfc7521>. Citado na página 12.

CERTISIGN. Certificado digital: O que é? 2023. Disponível em: <https://blog.certisign.com.br/o-que-e-certificado-digital/>. Citado na página 22.

CHEN, Y.; ZHAO, Y. Half-aggregation of schnorr signatures with tight reductions. In: ATLURI, V. et al. (Ed.). *Computer Security – ESORICS 2022*. Cham: Springer Nature Switzerland, 2022. p. 385–404. ISBN 978-3-031-17146-8. Citado na página 30.

Cloud Native Computing Foundation. *TOC/definition.md at Main · CNCF/TOC*. 2022. Disponível em: <https://github.com/cncf/toc/blob/main/DEFINITION.md>. Citado na página 18.

CLOUDFLARE. O que é mtls? 2023. Disponível em: <https://www.cloudflare.com/learning/ssl/what-is-mutual-tls/>. Citado na página 23.

CLOUDFLARE. O que é protocolo tls. 2023. Disponível em: <https://www.cloudflare.com/learning/ssl/what-is-tls/>. Citado na página 22.

DAVIS, L. *What is cloud computing? The ultimate guide*. Forbes Magazine, 2022. Disponível em: <https://www.forbes.com/advisor/business/what-is-cloud-computing/>. Citado na página 14.

DIERKS, T.; ALLEN, C. *The TLS Protocol Version 1.0*. 1999. Disponível em: <https://tools.ietf.org/html/rfc2246>. Citado na página 23.

EL-GAZZAR, R. F. A literature review on cloud computing adoption issues in enterprises. In: BERGVALL-KÅREBORN, B.; NIELSEN, P. A. (Ed.). *Creating Value for All Through IT*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. p. 214–242. Citado na página 10.

FELDMAN, D. et al. *Solving the Bottom Turtle: a SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity*. [s.n.], 2020. Disponível em: <https://spiffe.io/book/>. Citado na página 24.

FORTINET. *O que é criptografia?* 2023. Disponível em: <https://www.fortinet.com/resources/cyberglossary/what-is-cryptography>. Citado na página 21.

GALVãO, M. Introdução à criptografia. 2023. Disponível em: <https://www.academia.edu/45620514/Introdu%C3%A7%C3%A3o_%C3%A0_criptografia>. Citado na página 21.

GOCACHE. *O que é mTLS*. 2022. Disponível em: <https://www.gocache.com.br/seguranca/o-que-e-mtls/>. Citado na página 23.

GOIáS, P. U. C. de. Assinatura digital e a evoluÇÃo da tecnologia na pandemia. 2023. Disponível em: <https://repositorio.pucgoias.edu.br/jspui/bitstream/123456789/2142/2/TCC%20-%20Assinatrua%20Digital%20-%20ELYSA%20PUC%20A05%20%281%29.pdf>. Citado na página 21.

HARDT, D. *The OAuth 2.0 Authorization Framework*. RFC Editor, 2012. RFC 6749. (Request for Comments, 6749). Disponível em: <https://www.rfc-editor.org/info/rfc6749>. Citado na página 12.

IBM. *What is cloud computing?* Https://www.ibm.com/topics/cloud-computing. Disponível em: <https://www.ibm.com/topics/cloud-computing>. Citado na página 14.

JONES, M. *JSON Web Algorithms (JWA)*. [S.l.], 2015. Citado na página 31.

JONES, M. B.; CAMPBELL, B.; MORTIMORE, C. *JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants*. RFC Editor, 2015. RFC 7523. (Request for Comments, 7523). Disponível em: <https://www.rfc-editor.org/info/rfc7523>. Citado na página 12.

KARATI, S. et al. New algorithms for batch verification of standard ecdsa signatures. *Journal of Cryptographic Engineering*, v. 4, p. 237–258, 2014. Disponível em: <https://doi.org/10.1007/s13389-014-0082-x>. Citado na página 22.

KITTUR, A. S.; PAIS, A. R. Batch verification of digital signatures: Approaches and challenges. *Journal of Information Security and Applications*, v. 37, p. 15–27, 2017. ISSN 2214-2126. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2214212617301862>. Citado na página 30.

LARSSON, M. *Microservices*. Aarhus Universitet, Datalogisk Institut, 2014. Disponível em: <https://aws.amazon.com/microservices/>. Citado na página 17.

LEARN, M. Protocolo tls. 2023. Disponível em: <https://docs.microsoft.com/pt-br/windows-server/security/tls/tls-protocol-overview>. Citado na página 22.

MELL, P.; GRANCE, T. The nist definition of cloud computing. *The NIST Definition of Cloud Computing*, v. 800–145, p. 2–3, Sep 2011. Disponível em: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf>. Citado na página 15.

ORACLE. *What is cloud native?* 2022. Disponível em: <https://www.oracle.com/cloud/cloud-native/what-is-cloud-native/>. Citado na página 18.

PETERSENL, H.; HORSTER, P. Self-certified keys — concepts and applications. In: _____. *Communications and Multimedia Security: Volume 3*. Boston, MA: Springer US, 1997. p. 102–116. ISBN 978-0-387-35256-5. Disponível em: <https://doi.org/10.1007/978-0-387-35256-5_8>. Citado na página 30.

Red Hat. *What are microservices?* Red Hat, 2023. Disponível em: <https://www.redhat.com/en/topics/microservices/what-are-microservices>. Citado na página 16.

RESCORLA, E. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC Editor, 2018. RFC 8446. (Request for Comments, 8446). Disponível em: <https://www.rfc-editor.org/info/rfc8446>. Citado na página 23.

SOUZA, I. P. M. de; NETO, B. B. Certificação digital: Conceitos e aplicações. 2023. Disponível em: <https://simtec.fatectq.edu.br/index.php/simtec/article/download/273/221/>. Citado na página 22.

SPIFFE. *SPIFFE oficial website*. 2023. Disponível em: <https://spiffe.io/>. Citado na página 11.

SPIFFE. *SPIFFE-Assertions*. s.d. Disponível em: <https://github.com/spiffe/spiffe/blob/main/standards/SPIFFE-ID.md#41-svid-assertions>. Citado na página 11.

VETTOR, R.; SMITH, S. *Architecting Cloud native .net applications for Azure*. 2023. Disponível em: <https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/>. Citado 3 vezes nas páginas 17, 18 e 19.