UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

THIAGO CAMARGO DA COSTA
Orientador: Prof. Dr. Carlos Frederico M. Cunha Cavalcanti

IMPLEMENTAÇÃO DE SOFTWARE PARA DEMONSTRAÇÃO DE PROVA DE POSSE EM SERVIÇOS WEB

Ouro Preto, MG
2024

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS
DEPARTAMENTO DE COMPUTAÇÃO

THIAGO CAMARGO DA COSTA

IMPLEMENTAÇÃO DE SOFTWARE PARA DEMONSTRAÇÃO DE PROVA DE POSSE EM SERVIÇOS WEB

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Carlos Frederico M. Cunha Cavalcanti

Ouro Preto, MG
2024

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

C837i Costa, Thiago Camargo da.
Implementação de software para demonstração de prova de posse em serviços Web. [manuscrito] / Thiago Camargo da Costa. - 2024.
40 f.: il.: color., tab..

Orientador: Prof. Dr. Carlos Frederico Marcelo da Cunha Cavalcanti.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Graduação em Ciência da Computação .

1. Autenticação. 2. Criptografia de dados (Computação). 3. Web - Medidas de segurança. 4. Segurança de sistemas. I. Cavalcanti, Carlos Frederico Marcelo da Cunha. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.738.5

Bibliotecário(a) Responsável: Michelle Karina Assuncao Costa - SIAPE: 1.894.964

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E BIOLOGICAS
DEPARTAMENTO DE COMPUTACAO

FOLHA DE APROVAÇÃO

Thiago Camargo Da Costa

Implementação de software para demonstração de prova de posse em serviços Web

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Ciência da Computação

Aprovada em 27 de Março de 2024.

Membros da banca:

Carlos Frederico Marcelo da Cunha Cavalcanti (Orientador) - Doutor - Universidade Federal de Ouro Preto
Fernando Cortez Sica (Examinador) - Doutor - Universidade Federal de Ouro Preto
Agnaldo Jose da Rocha Reis (Examinador) - Doutor - Universidade Federal de Ouro Preto

Carlos Frederico Marcelo da Cunha Cavalcanti, Orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 27/02/2024.

Documento assinado eletronicamente por **Carlos Frederico Marcelo da Cunha Cavalcanti**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 18/04/2024, às 08:07, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0675109** e o código CRC **08F5DA89**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591692 - www.ufop.br

Pai, mãe e irmão, nenhuma conquista será superior ao prazer de tê-los em minha vida.

Agradecimentos

Agradeço meus familiares por todo o amparo. Professor Carlos Frederico Cavalcanti, muito obrigado pela oportunidade de ser seu orientando. Professores Rodrigo Ribeiro e Denis Plec, obrigado pelas oportunidades de monitoria. Professora Izinara Rosse, Lauro Moraes e Endgel Aguiar, obrigado pela experiência no projeto do LBCM. Professores Rodrigo Pedrosa e Tiago Garcia, obrigado pelas orientações acadêmicas. Aos ex-moradores e moradores da república Canaan - StandBy, Boca, Canta, Sovado, Sidcley, Belo, Queixas, Refri, Fiscal, Dendê, Pelé e Fazcena - minha gratidão pelos momentos compartilhados. Colegas de curso - Gustavo de Castro, Pedro Igor e Patrick Oliveira - obrigado pela companhia durante a jornada. Por fim, agradeço aos professores do Departamento de Computação da Universidade Federal de Ouro Preto pelos ensinamentos e pelo esforço empenhado para oferta de um ensino de qualidade.

"On the Internet, nobody knows you’re a dog".

(Peter Stein, 1993, The New Yorker)

Resumo

Protocolos e mecanismos de Autenticação, Autorização e Auditoria (AAA) são utilizados em sistemas computacionais para garantir a integridade e o não repúdio em rotinas de troca de informação. Demonstração de prova de posse busca evitar a utilização de informações, roubadas ou vazadas, por entidades ilegítimas. Um contexto das áreas de criptografia e sistemas computacionais é apresentado junto de dois dos padrões de autorização utilizados atualmente. O mecanismo apresentado no RFC 9449 é objeto de estudo deste trabalho. Uma análise de requisitos é conduzida com base na especificação do mecanismo contido no RFC 9449. Algumas das funcionalidades do mecanismo são implementadas e validadas por meio de testes automatizados. Para inviabilizar *token replay attacks* e *token export attacks* são implementados, respectivamente, um método para geração de *nonces* e um método para a obtenção do `jkt` de uma chave pública. Em relação ao fluxo representado na figura 1 do RFC 9449, têm-se que é possível construir requisições por tokens de acesso utilizando provas DPoP. A validação das reivindicações de um DPoP-Proof JWT não é possível de ser realizada de forma programática pela implementação elaborada.

Palavras-chave: autorização. autenticação. criptografia.

Abstract

Authentication, Authorization, and Accountability (AAA) protocols and mechanisms are used in computer systems to guarantee integrity and non-repudiation in information exchange routines. Demonstrating proof of possession seeks to prevent the usage of stolen or leaked information by illegitimate entities. A context about cryptography and computer systems is provided, along with two currently used authorization standards. The mechanism presented in RFC 9449 is the object of study in this work. A requirements analysis is conducted based on the specification contained in RFC 9449. Some of its functionalities are implemented and validated through automated testing. A method for generating nonces and one for obtaining the jkt of a public key are implemented to address token replay attacks and token export attacks. Concerning the flow represented in Figure 1 of RFC 9449, it is possible to construct requests for access tokens using DPoP proofs. It is not possible to programmatically validate the claims contained in a DPoP-Proof JWT by the implementation provided in this work.

Keywords: authorization, authentication, cryptography.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

DECOM	Departamento de Computação
UFOP	Universidade Federal de Ouro Preto
AAA	*Authorization, Authentication, and Accountability*
DPoP	*Demonstrating Proof-of-Possession*
RFC	*Request for Comments*
TLS	*Transport Layer Security*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
JWS	*JSON Web Signature*
JWK	*JSON Web Key*
JKT	*JSON Web Key Thumbprint*
ACL	*Access Control List*
API	*Application Programming Interface*
ABNF	*Augmented Backus-Naur form*
SDK	*Software Development Kit*

Sumário

1 Introdução

Sistemas computacionais são acessórios na realização de uma pluralidade de tarefas de escopo pessoal e profissional. A comunicação feita entre usuários e sistemas requer o reconhecimento entre as partes. Serviços de Autenticação, Autorização e Auditoria (AAA) - *Authentication*, *Authorization* e *Accountability* - desempenham as rotinas necessárias para atingir esse reconhecimento (Kajan, 2002). A disponibilização de serviços desse tipo busca fornecer a entidade, seja uma pessoa ou uma organização, segurança de que as ações realizadas por meio de dispositivos computacionais foram realizadas com as devidas permissões e entre as devidas partes.

Dois padrões são utilizados atualmente para prover a autenticidade de usuários e controle de acesso. São eles: OAuth 2.0 (Hardt, 2012) e OpenID Connect (Sakimura *et al.*, 2014). O primeiro padroniza as formas de requisição e concessão de autorização enquanto o segundo estende as funcionalidades do primeiro para padronizar rotinas de autenticação.

DPoP é um mecanismo de camada da aplicação para restringir o remetente de tokens de acesso (*access tokens*) e de atualização (*refresh tokens*) elaborado por (Fett *et al.*, 2023). Os tokens de acesso e de atualização mencionados anteriormente são definidos na proposta de padrão OAuth 2.0 por (Hardt, 2012). O mecanismo visa previnir que entidades ilegítimas utilizem tokens roubados ou vazados. Essa prevenção é obtida através da criação de um vínculo entre um token emitido e uma chave pública e, da exigência de que o usuário do token prove que possui a chave privada relacionada a chave pública vinculada.

1.1 Justificativa

O provisionamento de componentes em software capazes de desempenhar as funcionalidades do mecanismo proposto é importante para adoção do mecanismo por serviços que utilizam tokens de acesso com a finalidade de prevenir que entidades ilegítimas utilizem tokens roubados ou vazados.

1.2 Objetivos

Têm-se como objetivo geral a implementação de componentes em software necessários para a utilização de DPoP nas interações entre usuários e servidores de aplicações Web.

1.2.1 Objetivos específicos

Os seguintes objetivos específicos serão realizados para cumprir o objetivo geral acima:

- Analisar as especificações do mecanismo definido no documento;
- Implementar os componentes;
- Validar os componentes implementados de acordo com a especificação do mecanismo DPoP e,
- Analisar os resultados obtidos.

1.3 Organização do Trabalho

O conteúdo do trabalho é disposto em 5 capítulos. O primeiro é destinado a introdução do tema e apresentação dos objetivos a serem alcançados. O segundo contém a fundamentação teórica construída a partir de uma pesquisa bibliográfica sobre o tema. O terceiro documenta o desenvolvimento do trabalho. O quarto apresenta os resultados obtidos junto de sua interpretação e análise. O quinto possui as considerações finais.

2 Fundamentação Teórica

Esta seção é destinada a exposição de conceitos relacionados ao trabalho. Serão apresentadas definições dos termos utilizados para contextualização da implementação desenvolvida. O conteúdo está dividido em seções intituladas por tópicos das áreas de criptografia, redes de computadores e padrões de autorização.

2.1 Criptografia assimétrica (*Asymmetric Cryptography*)

É um ramo da área de criptografia no qual os algoritmos usam um par de chaves - uma chave pública e uma chave privada - e utilizam um componente diferente do par para cada uma das duas operações criptográficas correspondentes e.g. encriptação e decriptação. A criptografia assimétrica pode ser utilizada na criação de algoritmos para encriptação, assinatura digital e acordo de chave (Shirey, 2007).

A criptografia visa proteger informações e garantir sua manipulação e utilização de forma eficiente por meio de algoritmos criptográficos (Al-Shabi, 2019). Práticas de segurança em computadores fazem uso do modelo CIA - *confidentiality*, *integrity* e *availability* - que é composto por serviços focados em restringir o acesso à informação e assegurar de que os dados transmitidos e recebidos não foram adulterados (Mitali; Sharma *et al.*, 2014).

O relatório técnico *Digital Signature Standard* - DSS - (FIPS 186-5) (Moody, 2023) especifica um conjunto de algoritmos que podem ser utilizados na geração de assinaturas digitais. Alguns deles são o RSA - especificado no RFC 8017 - e o *Elliptic Curve Digital Signature Algorithm*. As assinaturas digitais são utilizadas para: detecção de modificações não autorizadas aos dados; autenticação da identidade do assinante e, como evidência para um terceiro, de que quem assinou é aquele que apresentou a mensagem com a assinatura.

2.2 Paradigma de comunicação baseado em passagem de mensagem

O paradigma de comunicação baseado em passagem de mensagem é utilizado para tratar a troca de informação entre sistemas. Nesse paradigma, é feita uma distinção entre dois tipos de papéis desempenhados pelas partes envolvidas: o papel de remetente e o papel de destinatário. Assume-se que as partes envolvidas possuem acesso a um meio de comunicação capaz de transmitir as mensagens emitidas (Steen; Sips *et al.*, 1995). O remetente é aquele que envia requisições de informação a um destinatário que em troca retorna uma resposta com base na informação requisitada e no contexto provido.

2.2.1 Arquitetura cliente-servidor

A arquitetura cliente-servidor faz uso do paradigma de comunicação baseado em passagem de mensagem para troca de informação entre duas entidades. As entidades definidas nessa arquitetura são denominadas **cliente** e **servidor**. De acordo com (Steen; Tanenbaum, 2023), um **cliente** é um processo que emite uma requisição por um serviço a um servidor e em seguida aguarda por uma resposta. Um **servidor** é um processo que implementa um serviço específico como um serviço de banco de dados.

2.3 Mecanismos da camada de aplicação

Esta subseção é dedicada a descrição de mecanismos utilizados na criação de rotinas de comunicação que utilizam a infraestrutura da *Internet* para disponibilizar serviços específicos. A camada de aplicação tratada aqui é a definida por (Braden, 1989) dentro do *Internet Protocol Suite*. Nessa definição, a camada de aplicação é composta por protocolos pertencentes a uma das seguintes categorias: protocolos de usuário que fornecem serviços diretamente aos usuários e protocolos de suporte que proveem funções de sistema comuns.

2.3.1 Hypertext Transfer Protocol

O *Hypertext Transfer Protocol* (HTTP) é uma família de protocolos de requisição/resposta, sem estado - do inglês *stateless* - em nível de aplicação com a finalidade de permitir a comunicação via hipertexto em sistemas de informação conectados em rede (Fielding; Nottingham; Reschke, 2022).

A especificação, padronizada atualmente no RFC 9110, provê uma interface para interação com um recurso por meio do envio de mensagens que manipulam ou transferem representações. Cada mensagem é uma requisição ou uma resposta. Temos aqui um paradigma de comunicação que segue a arquitetura cliente-servidor. Um cliente constrói mensagens de requisição que comunicam suas intenções e as encaminha para um servidor. Um servidor aguarda por requisições, analisa a mensagem recebida, interpreta o sentido semântico da mensagem com base no remetente e responde a requisição com uma ou mais mensagens. Por fim, o cliente examina a resposta recebida para interpretar o resultado da requisição realizada.

2.3.2 Transport Layer Security

O protocolo *Transport Layer Security* (TLS) busca permitir que a comunicação entre aplicações cliente/servidor esteja protegida de espionagem, adulteração e falsificação de mensagem (Rescorla, 2018). Seus componentes são *handshake protocol* e o *record protocol*. O primeiro autentica as partes, seleciona o método e parâmetros criptográficos e estabelece o compartilhamento de chaves. O segundo faz uso dos parâmetros estabelecidos pelo *handshake protocol* para

proteger o tráfego entre as partes. Os detalhes da última versão até o momento são apresentados no padrão proposto no RFC 8446.

2.4 Application Programming Interface (API)

Uma API consiste de uma especificação e implementação de uma interface em software. Sua finalidade é definir e possibilitar a comunicação entre duas ou mais aplicações. Web APIs são um tipo de API que utilizam o protocolo HTTP para transmitir e acessar informações em aplicações Web.

No momento de escrita deste documento, apesar de existir vários formatos de API usados para o desenvolvimento de aplicações, como gRPC, GraphQL, REST (Fielding, 2000), AsyncAPI e OpenAPI, APIs usando o regramento RESTful são as mais usadas.

Representational State Transfer (abreviado REST), em português "Transferência de Estado Representacional", é um padrão de arquitetura de software apresentado na tese de doutorado de Roy Fielding na Univesity of California at Irvine, que define regramentos para a criação de uma API que foi denominado *Web services RESTful*. Um serviço Web genuinamente REST é construído de acordo com os regramentos RESTful e é baseado em troca de mensagens no formato HTTP, fornecendo interoperabilidade entre sistemas de computadores conectados via Internet. RESTful permite que os sistemas solicitantes acessem e manipulem representações textuais de recursos da Web usando um conjunto uniforme e predefinido de operações sem estado (requisição e resposta independentes). Uma das representações de dados usadas na criação de uma interface REST é:

- **JSON**, definido por (Bray, 2017), é um formato de arquivo de padrão aberto e formato de troca de dados que usa uma cadeia de de caracteres - em inglês *string* - para armazenar e transmitir objetos de dados que consistem em pares atributo-valor e matrizes ou outros valores serializáveis.

2.5 Autenticação baseada em tokens

Em um dado momento, a maior parte dos serviços disponibilizados via Internet eram construídos para serem consumidos por um navegador. Com a adoção de APIs por modelos de desenvolvimento de software e por provedores de serviços baseados em software surgiu a necessidade exercer o controle de autorização e autenticação daqueles que consumiam a API ofertada.

A autenticação feita a partir de nome de usuário e senha precisava ser realizada toda vez que uma chamada a API fosse realizada. Os *cookies* foram criados para contornar esse inconveniente. Com eles, o servidor passa a ser capaz de enviar um valor para o navegador do

cliente e requisitar o seu armazenamento para indicar que uma sessão já foi estabelecida com sucesso e que ela poderia ser reutilizada em chamadas subsequentes. Essa abordagem trouxe vulnerabilidades como *session fixation*, o problema de resolução de CORS além da necessidade de editar o código fonte exposto ao cliente para suportar o gerenciamento de sessões.

O padrão de autorização OAuth adicionou uma funcionalidade para possibilitar o acesso a recursos protegidos. A funcionalidade é denominada *bearer token* (Jones; Hardt, 2012) e o paradigma de autenticação é chamado de *Bearer Authentication Scheme*. Um cliente passa a ser capaz de enviar um *bearer token* ao servidor de recurso para acessar o recurso protegido. Implementações iniciais do *bearer authentication scheme* faziam uso de tokens que eram gerados pelo serviço responsável pelo controle de acesso a recursos. O serviço de controle de acesso precisa gerar, enviar e armazenar tokens para verificação.

Atualmente, serviços que utilizam o paradigma de *Bearer Authentication Scheme* fazem uso de tokens padronizados como *JSON Web Tokens* (Jones; Bradley; Sakimura, 2015b).

2.6 Estruturas JSON para AAA

Algumas estruturas de representação de dados, baseados em *JavaScript Object Notation* (JSON), são utilizados em rotinas de comunicação com aplicações Web e estão dentro do contexto de AAA. As estruturas mencionadas no RFC 9449 são descritas a seguir.

2.6.1 JSON Web Signature (JWS)

JSON Web Signature (JWS) é uma forma de representação do conteúdo de dados em formato JSON definida no RFC 7515 por (Jones; Bradley; Sakimura, 2015a). Essa forma utiliza mecanismos criptográficos para construir uma representação assinada digitalmente do conteúdo de uma mensagem.

2.6.2 JSON Web Key (JWK)

JSON Web Key (JWK) é estrutura de dados, baseada no formato JSON, definida no RFC 7517 por (Jones, 2015) que representa uma chave criptográfica.

2.6.3 JSON Web Key Thumbprint (JKT)

JSON Web Key Thumbprint é valor do *hash* de uma *JSON Web Key*. A especificação do método para o computar o *hash* é definido no RFC 7638 por (Jones; Sakimura, 2015). O valor do *hash* pode ser utilizado para identificação ou seleção de uma chave criptográfica representada por uma JWK.

2.6.4 JSON Web Token (JWT)

JSON Web Token (JWT) é uma estrutura de representação de reivindicações e é empregado em ambientes como cabeçalhos de autorização HTTP e parâmetros de consulta de *Uniform Resource Identifier* (URI) (Jones; Bradley; Sakimura, 2015b).

A estrutura de um JWT é definida pelos campos *Header*, *Payload* e *Signature*. O *Header* armazena o tipo do token e o algoritmo criptográfico utilizado para assinatura. O *Payload* armazena as reinvidicações. O *Signature* é criado a partir da codificação dos campos *Header* e *Payload* junto de um segredo e do algoritmo especificado no *Header*.

2.6.5 DPoP Proof JWT

Esta estrutura é um JWT criado por um cliente e enviado em uma requisição HTTP dentro do campo do cabeçalho DPoP destinado unicamente para esse formato (Fett *et al.*, 2023).

Uma DPoP *proof* válida demonstra ao servidor que o cliente possui a chave privada utilizada para assinar o DPoP Proof JWT. Isso permite que *authorization servers* vinculem os tokens emitidos à chave pública correspondente e, que *resource servers* verifiquem os tokens recebidos para prevenir que tais tokens sejam utilizados por entidades que não possuem acesso à chave privada (Fett *et al.*, 2023).

A estrutura mínima de um DPoP *proof* JWT é apresentada abaixo. *Header* e *Payload* são campos e os itens presentes em cada um são pares do tipo chave-valor.

- *Header*:
  - typ: "dpop+jwt";
  - alg: identificador de um algoritmo de assinatura digital assimétrica suportado por JWS e.g. "ES256";
  - jwk: a chave pública escolhida pelo cliente representada em formato JWK;
- *Payload*:
  - jti: identificador único para o DPoP *Proof* JWT;
  - htm: valor do método da requisição HTTP ao qual o JWT está anexado;
  - htu: URI do destino HTTP sem as partes *query* e *fragment* da requisição ao qual o JWT está anexado;
  - iat: *timestamp* do momento de criação do JWT;
  - ath: *hash* do token de acesso - no formato especificado no RFC 7515 - para quando o DPoP JWT é apresentado junto de um token de acesso durante o acesso a um recurso protegido;

2.7 Padrões de autorização

A autorização é a interpretação de uma regra de acesso (Lampson *et al.*, 1992). Regras de acesso em sistemas computacionais são implementadas em modelos de controle de acesso e.g. *Access Control List* (ACL) (Samarati; Vimercati, 2001), para restrição de acesso a algum recurso.

2.7.1 OAuth 2.0

OAuth 2.0 é um padrão proposto por (Hardt, 2012) para possibilitar que uma aplicação obtenha autorização para emissão de requisições a APIs de terceiros em nome do usuário. O framework busca contornar dificuldades advindas da interação entre serviços distintos que seguem o modelo cliente-servidor.

O *framework* denomina um usuário como *resource owner* e um terceiro como *resource server*. Um exemplo de caso de uso é apresentado por (Wilson; Hingnikar, 2023) e reproduzido a seguir.

Considere um usuário que deseja compartilhar um recurso associado a sua conta em um serviço que hospeda uma aplicação A com um outro serviço que hospeda uma aplicação B. Sem a padronização do *framework* OAuth2, o usuário precisaria compartilhar suas credenciais de acesso à aplicação A com o servidor da aplicação B. Já com a padronização, o usuário é capaz de emitir uma autorização para a aplicação B requisitar um recurso hospedado pela aplicação A.

A resolução da requisição de acesso a uma API de terceiro é feita por um componente denominado *authorization server*.

Os tokens definidos pelo OAuth 2.0 são *access token* e *refresh token*. O primeiro é utilizado por uma aplicação para acessar uma API e simboliza que a aplicação tem autorização para realizar o acesso a API. O segundo é utilizado por uma aplicação na requisição de um novo *access token* quando um *access token* anterior estiver expirado.

O *framework* OAuth 2.0 define tipos distintos de interações entre aplicações e *authorization servers* para concessão de autorização de acesso a uma API. Os tipos de concessão definidos são: *Authorization code*, *Client credentials*, PKCE, e *Refresh token*.

2.7.2 OpenID Connect

OpenID Connect (OIDC) é um protocolo que provê um serviço de identidade através de uma camada sobre o padrão OAuth 2.0 concebido com a finalidade de permitir que servidores de autorização autorizem usuários de aplicações de uma maneira padronizada (Wilson; Hingnikar, 2023).

2.8 Ataques cibernéticos

Esta seção é destinada à descrição de ataques que o mecanismo busca inviabilizar.

2.8.1 Token Replay Attack

Um ataque de repetição - *replay attack* - é um tipo de ataque onde o atacante é capaz de repetir uma mensagem capturada anteriormente - entre um requerente legítimo e um verificador - para se disfarçar como esse requerente para o verificador ou vice-versa (Grassi; Garcia; Fenton, 2017). No caso de um *Token Replay Attack*, a mensagem capturada e repetida é aquela que contém um token capaz de ser utilizado para acessar algum recurso.

2.8.2 Token Export Attacks

Um ataque de exportação de token - *token export attack* - é um tipo de ataque onde um cliente, comprometido por uma vulnerabilidade, vaza um token recebido para um atacante (Heilman *et al.*, 2023). O atacante passa a ser capaz de personificar o usuário do cliente comprometido.

2.9 Trabalhos Relacionados

Esta seção apresenta as referências utilizadas na elaboração deste trabalho.

Uma pesquisa bibliográfica foi conduzida para identificar trabalhos que tratam dos paradigmas de autenticação *Bearer Authentication* e *Proof-of-Possession*. O Google Scholar foi utilizado como ferramenta de busca de publicações e a consulta realizada foi: `"bearer authentication" AND "proof-of-possession"`. Os resultados foram filtrados com a restrição de data de publicação em 2023 totalizando três documentos. Os documentos retornados são apresentados nas subseções a seguir.

2.9.1 OpenPubkey

No artigo intitulado "*OpenPubkey: Augmenting OpenID Connect with User-held Signing Keys*", (Heilman *et al.*, 2023) apresentam o protocolo OpenPubkey. Esse protocolo realiza uma modificação no programa OpenID Connect que culmina na mudança do paradigma de autenticação suportado, de *Bearer Authentication* para *Proof-of-Possession*, produzindo uma melhoria na segurança quanto a integridade de mensagens trocadas entre o cliente e o servidor de autenticação. A melhoria possibilita a prevenção de *token replay attacks* e *token export attacks*.

2.9.2 OAuth 2.0 Demonstrating Proof of Possession (DPoP)

O mecanismo proposto por (Fett *et al.*, 2023) apresenta um exemplo de um fluxo de interações entre cliente e servidor de autorização e entre cliente e servidor de recurso que culmina

na obtenção de acesso, por parte do cliente, a um recurso protegido .

A interação entre o cliente e o servidor de autorização é feita em dois passos. O primeiro passo é a requisição, feita pelo cliente, de um token de acesso. O segundo passo é o envio de um token de acesso DPoP pelo servidor de autorização ao cliente.

A interação entre cliente e servidor de recurso também é feita em dois passos. No primeiro passo, o cliente apresenta ao servidor de recurso o token de acesso DPoP enviado pelo servidor de autorização. O servidor de autorização analisa o token recebido e retorna o recurso protegido.

2.9.3 OAuth 2.0 Step Up Authentication Challenge Protocol

O mecanismo proposto por (Bertocci; Campbell, 2023) busca possibilitar que servidores de recurso sinalizem ao cliente que um evento de autenticação associado ao token de acesso da requisição atual não atende aos requisitos impostos pelo fluxo de concessão de autorização. Os autores também fornecem um mecanismo para que um cliente seja capaz de requisitar um certo tipo de autenticação ao servidor de autorização que processa sua requisição de autorização.

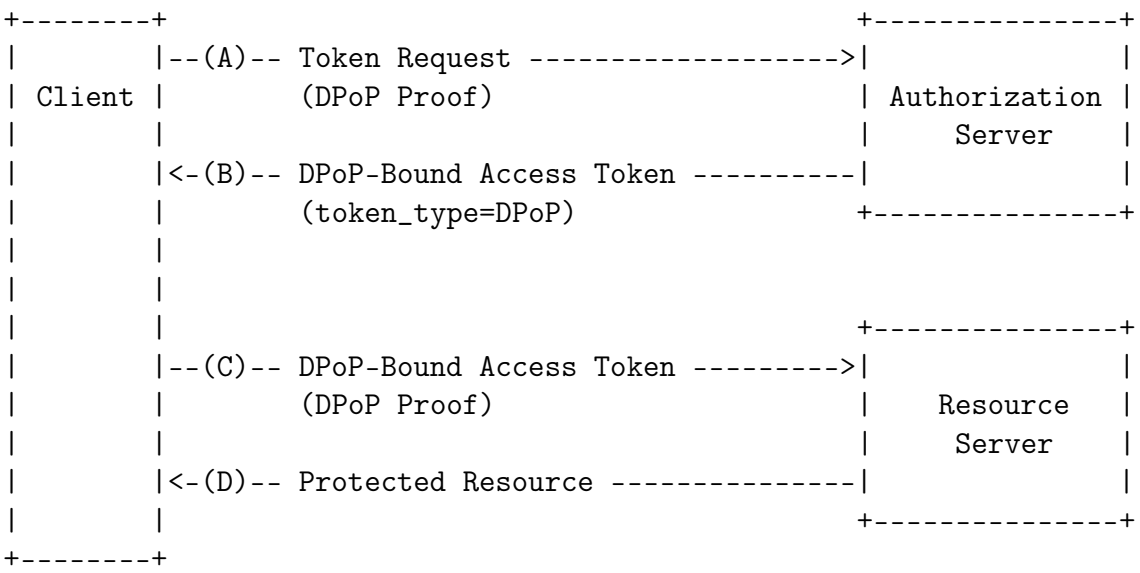
3 Desenvolvimento

As descrições e relatos dos procedimentos conduzidos para o desenvolvimento do trabalho são apresentados a seguir.

3.1 Identificação dos requisitos

A imagem a seguir, extraída do documento de (Fett *et al.*, 2023) que define o mecanismo, representa um fluxo de interações para concessão de acesso a um recurso protegido através da requisição, construção e utilização de um token DPoP.

Figura 3.1 – Fluxo DPoP



Fonte: (Fett *et al.*, 2023)

É possível identificar os seguintes requisitos com base nas interações definidas pelo mecanismo:

- Um cliente deve ser capaz de construir uma prova DPoP que será anexada ao cabeçalho HTTP da requisição por um token de acesso;
- Um cliente deve ser capaz de enviar uma requisição HTTP por um token de acesso ao servidor de autorização;
- Um servidor de autorização deve ser capaz de checar a prova DPoP presente no cabeçalho da requisição HTTP enviada pelo cliente;

- Um servidor de autorização deve ser capaz de vincular um token de acesso a chave pública de uma prova DPoP válida;
- Um cliente deve ser capaz de construir uma requisição de acesso a um recurso protegido contendo o token de acesso vinculado a uma prova DPoP;
- Um servidor de recurso deve ser capaz de checar se as credenciais contidas na requisição de acesso a um recurso protegido são válidas;

3.2 Elaboração dos componentes

As definições e o código fonte referentes à implementação dos componentes são apresentados nesta seção. As ferramentas utilizadas foram:

- Go 1.22;
- cristalhq/jwt v5;

O código fonte produzido para elaboração de DPoP Proof JWTs está disponível em https://github.com/thiagocamargodacosta/dpopjwt.

O código fonte produzido para elaboração das interações entre cliente e servidor está disponível em https://github.com/thiagocamargodacosta/dpopflow.

3.2.1 DPoP-Proof JWT

A estrutura de representação especificada foi elaborada como um JWT com campos adicionais. A implementação fornecida pela biblioteca `cristalhq/jwt/v5` foi adaptada para dar suporte a esse novo tipo de token.

O campo `Jwk` foi adicionado ao tipo `Header` para armazenar a chave pública do par de chaves utilizado na criação do DPoP-Proof JWT. A estrutura do *Header* e do *Payload* são apresentadas no anexos A.1 e B.1.

O tipo da reivindicação `Cnf`, presente no tipo `RegisteredClaims`, é definido pelo trecho de código presente no anexo C.1 e serve para armazenar o *hash*, no formato `base64url`, da chave pública usada para construção da prova DPoP.

3.2.2 Gerador de nonce

A cadeia de caracteres gerada deve seguir a regra de produção, na forma ABNF, a seguir. O prefixo `%x` indica que a codificação é em hexadecimal. O caractere `-` constrói um conjunto de caracteres onde o primeiro elemento é aquele à esquerda do `-` e o último elemento é aquele à

direita do `-`. Os caracteres entre o primeiro e último também fazem parte do conjunto. O caractere `/` indica uma alternativa.

Figura 3.2 – Regras de produção para o nonce

```
nonce   = 1*NQCHAR
NQCHAR = %x21 / %x23-5B / %x5D-7E
```

Fonte: (Fett *et al.*, 2023) e (Hardt, 2012)

O nonce gerado pelo servidor de autorização é produzido pela função `GenerateNonce` no anexo D.1 que recebe como parâmetro o tamanho desejado. A variável `nqchar` armazena os caracteres definidos na regra de produção. Um gerador de números pseudoaleatórios - `rand.Intn` - produz um número que será usado para selecionar o caractere em `nqchar` a ser adicionado no `nonce` gerado.

3.2.3 Método para confirmação de chave pública

A função `Jkt` no anexo E.1 foi elaborada para permitir que um servidor de recurso possa identificar se um token de acesso possui ligação com uma prova DPoP. Ela segue o método de *JWK Thumbprint confirmation* no qual o *hash* produzido é gerado a partir do algoritmo SHA-256 utilizando os campos - `crv`, `kty`, `x` e `y` - definidos no RFC 7638.

3.2.4 Rotas do servidor de autorização

A tabela a seguir contém as rotas expostas pelo servidor de autorização, os métodos suportados por cada rota e sua função.

Tabela 3.1 – Rotas expostas pelo servidor de autorização

Rota	**Método HTTP**	**Resultado de uma requisição com sucesso**
/nonce	POST	Retorna um nonce
/token	POST	Retorna um token de acesso

Fonte: De autoria própria

Para produzir um *nonce*, o servidor deverá receber uma requisição contendo o *hash* da chave pública do cliente utilizada para construção do DPoP-Proof JWT como valor do cabeçalho `jkt`. O *hash* tem como finalidade a identificação do usuário. A função `nonceHandler` no anexo F.1 implementa a lógica da rota `/nonce`.

Para retornar um token, o servidor deverá receber uma requisição contendo o DPoP-Proof JWT como valor do cabeçalho `DPoP`. A função `tokenHandler` no anexo G.1 implementa a lógica da rota `/token`.

3.2.5 Construtor de DPoP-Proof JWTs

A construção de um DPoP-Proof JWT necessita de um par de chaves criptográficas onde a chave pública é armazenada no *Header* seguindo o formato de JWK e a chave privada é utilizada para produzir o campo *Signature* do DPoP-Proof JWT. O construtor faz uso da função `GenerateNonce` para criar um *nonce* que será armazenado no *Payload* do DPoP-Proof JWT. O código no anexo H.1 implementa a funcionalidade de construção de DPoP-Proof JWTs.

3.2.6 Construtor de requisição por tokens ligados a DPoP

Para acessar um recurso protegido, o cliente deve apresentar um token que será obtido mediante o envio de uma requisição que contenha um DPoP-Proof JWT. O código no anexo I.1 constrói a requisição e aguarda pelo token de acesso.

3.3 Experimentação prática

O procedimento de experimentação consistiu na elaboração e execução de casos de teste para validação das funcionalidades construídas.

3.3.1 Validação do gerador de nonce

A função `CheckNonce`, contida no anexo J.1, busca validar o nonce gerado pela função `GenerateNonce`. Uma expressão regular capaz de descrever o conjunto de caracteres permitidos é construída na linha 5. O casamento do *nonce* fornecido como parâmetro com a expressão regular retorna se o padrão definido na expressão regular ocorre no *nonce* fornecido.

O conjunto de testes do anexo K.1 busca validar os seguintes cenários:

- A função `GenerateNonce` produz *nonces* que seguem as regras de produção definidas na especificação do mecanismo e,
- A função `CheckNonce` é capaz de sinalizar quando uma cadeia de caractere não pertence ao alfabeto especificado.

Um exemplo de *nonce* inválido é qualquer *nonce* gerado pela função `GenerateNonce` concatenado com um caractere não permitido e.g. aspas duplas.

A saída produzida pela execução do caso de teste é apresentada abaixo:

3.3.2 Validação do método para confirmação de chave pública

O procedimento de validação da função que calcula o `jkt` da chave pública foi feito a partir de um caso de teste que utiliza como entrada o *hash* fornecido como exemplo na figura 9

Figura 3.3 – Saída do caso de teste para validação das funções `GenerateNonce` e `CheckNonce`

```
$ go test -run TestNonce -v
=== RUN   TestNonce
--- PASS: TestNonce (0.00s)
PASS
ok      github.com/thiagocamargodacosta/dpopjwt 0.002s
```

Fonte: De autoria própria

da seção 6.1 do RFC 9449 e a chave pública contida no token presente no cabeçalho DPoP da requisição apresentada na figura 5 do mesmo documento.

O código elaborado para o caso de teste é apresentado no anexo L.1. A saída produzida pela execução do teste é apresentada a seguir.

Figura 3.4 – Saída do caso de teste para validação da função `CalculateJkt`

```
$ go test -run TestJkt -v
=== RUN   TestJkt
--- PASS: TestJkt (0.00s)
PASS
ok      github.com/thiagocamargodacosta/dpopjwt 0.002s
```

Fonte: De autoria própria

3.3.3 Validação do construtor de DPoP-Proof JWTs

O código a seguir foi elaborado para validar o construtor de DPoP-Proof JWTs. As funcionalidades de análise e verificação disponibilizadas pela biblioteca `cristalhq/jwt/v5` - `Parse` e `Verify` - foram utilizadas para checar se um DPoP-Proof JWT é válido.

O código elaborado para o caso de teste é apresentado no anexo M.1. A saída produzida após a execução do caso de teste é apresentada a seguir.

Figura 3.5 – Saída do caso de teste para validação do construtor de DPoP-Proof JWTs

```
$ go test -run TestDPoPProofBuild -v
=== RUN   TestDPoPProofBuild
    dpopjwt_test.go:10: Generate key pair
    dpopjwt_test.go:13: Craft DPoP JWT with example claims
    dpopjwt_test.go:16: Build verifier
    dpopjwt_test.go:23: Parse the token
    dpopjwt_test.go:32: Verify token signature
    dpopjwt_test.go:39: Get registered claims
    dpopjwt_test.go:47: Parse claims
--- PASS: TestDPoPProofBuild (0.00s)
PASS
ok      github.com/thiagocamargodacosta/dpopjwt        0.003s
```

Fonte: De autoria própria

3.3.4 Validação da rota /nonce do servidor de autorização

O caso de teste do anexo N.1 buscou validar a lógica da rota `/nonce` do servidor de autorização. Esperava-se que o servidor retornasse uma resposta com cabeçalho `DPoP_Nonce` com um *nonce* válido e com cabeçalho `Cache-Control` com o valor `"no-store"`. A saída produzida pela execução do caso de teste é apresentada abaixo.

Figura 3.6 – Saída do caso de teste para validação da rota /nonce

```
$ go test -run TestNonceHandler -v
=== RUN   TestNonceHandler
    handlers_test.go:92: Assemble POST request to /nonce
    handlers_test.go:100: Set required header value
    handlers_test.go:104: Create response recorder
    handlers_test.go:107: Send the request
    handlers_test.go:112: Evaluate the response
    handlers_test.go:116: Value in DPoP-Nonce header should be non-empty
    handlers_test.go:121: Value in DPoP-Nonce should be a valid nonce
    handlers_test.go:126: Value in Cache-Control should be no-store
    handlers_test.go:131: HTTP Status Code should be 201 (Created)
--- PASS: TestNonceHandler (0.00s)
PASS
ok      github.com/thiagocamargodacosta/dpopjwt 0.002s
```

Fonte: De autoria própria

3.3.5 Validação da rota /token do servidor de autorização

O caso de teste do anexo O.1 buscou validar a lógica da rota `/token` do servidor de autorização. Esperava-se que a resposta retornada fosse do tipo `application/json`, que o cabeçalho `Cache-Control` tivesse o valor `no-store`, que o corpo da resposta contivesse um conteúdo capaz de ser transformado para JSON e que o tipo do token presente no corpo da resposta fosse `dpop`. A saída produzida pela execução do caso de teste é apresentada abaixo.

Figura 3.7 – Saída do caso de teste para validação da rota /token

```
$ go test -run TestTokenHandler -v
=== RUN   TestTokenHandler
    handlers_test.go:16: Assemble POST request to /token
    handlers_test.go:24: Set the required headers
    handlers_test.go:29: Create response recorder
    handlers_test.go:32: Send the request
    handlers_test.go:38: Evaluate the response
    handlers_test.go:58: Read content of response body
    handlers_test.go:67: Expect content of body to be unmarshalable
    handlers_test.go:77: Expect response body to contain a 'token_type'
    ↪  key with value 'DPoP'
--- PASS: TestTokenHandler (0.00s)
PASS
ok      github.com/thiagocamargodacosta/dpopjwt 0.002s
```

Fonte: De autoria própria

4 Resultados

Como apenas algumas das funcionalidades requisitadas pelo mecanismo foram implementadas, têm-se que os requisitos do mecanismo foram parcialmente atendidos. Como consequência, o objetivo geral do trabalho foi parcialmente alcançado.

Em relação ao componente para geração de *nonces*, é possível aceitá-lo assumindo que os cenários elaborados endereçam todos os possíveis valores a serem enviados tanto pelo servidor, durante a requisição por um *nonce*, quanto pelo cliente durante o envio da prova DPoP construída e do token ligado à prova DPoP.

Quanto ao componente para obtenção do `jkt` de uma chave pública, é possível aceitá-lo para situações em que é permitido remover ocorrências do caractere = gerado pelo arredondamento do comprimento da cadeia de caracteres durante a codificação para `base64url`.

O caso de teste elaborado para validação do gerador de DPoP-Proof JWTs endereça somente a verificação da assinatura do token produzido. A análise - *parse* - realizada só valida a construção de uma estrutura do tipo `RegisteredClaims` por meio do desempacotamento - *unmarshaling* - da representação em `base64url` das reivindicações presentes no token produzido. É necessário validar cada um dos campos preenchidos pelo desempacotamento para aceitar ou recursar o token com base nas reivindicações recebidas. Dessa forma, se faz necessário recusar o componente que gera DPoP-Proof JWTs tendo em vista essas limitações.

A lógica das rotas `/nonce` e `/token` foram aceitas com base nos testes criados e são capazes de serem replicadas em implementações de clientes e servidores que desejem expôr as funcionalidades requisitadas pelo fluxo DPoP.

5 Considerações Finais

Este capítulo traz as considerações finais referentes ao trabalho.

5.1 Conclusão

Este trabalho fornece alguns dos componentes requisitados pelo mecanismo proposto por (Fett *et al.*, 2023) no RFC 9449. Os componentes foram sujeitos a validação por meio de testes não-automatizados. Em relação ao fluxo representado na figura 3.1, têm-se que é possível construir requisições com provas DPoP por tokens de acesso.

O construtor de DPoP-Proof JWTs é capaz de armazenar um *nonce* e o `jkt` da chave pública. Com isso, é possível identificar se um token de acesso está ligado a uma prova DPoP por meio do JWK *Thumbprint Confirmation Method* além de permitir que servidores de autorização limitem a validade de uma prova DPoP por meio de *nonces*.

A validação de um DPoP-Proof JWT não é possível de ser realizada de forma programática pela implementação elaborada. Apenas a verificação da assinatura do token e o desempacotamento das reivindicações podem ser realizados.

5.2 Trabalhos Futuros

Um refinamento da biblioteca adaptada para construção de DPoP-Proof JWTs é necessário para possibilitar a verificação programática de um token e suas reivindicações. Tal refinamento é uma das pendências necessárias para possibilitar a concessão de acesso a recursos protegidos por meio do fluxo descrito no RFC 9449. A outra, é interligar a biblioteca adaptada e um *Software Development Kit* (SDK) capaz de criar clientes e servidores de autorização e de recurso que seguem o padrão OAuth 2.0 para produzir clientes e servidores que suportem o fluxo de concessão de autorização por meio de DPoP-Proof JWTs.

Referências

AL-SHABI, M. A survey on symmetric and asymmetric cryptography algorithms in information security. **International Journal of Scientific and Research Publications (IJSRP)**, v. 9, n. 3, p. 576–589, 2019.

BERTOCCI, V.; CAMPBELL, B. **OAuth 2.0 Step Up Authentication Challenge Protocol**. RFC Editor, 2023. RFC 9470. (Request for Comments, 9470). Disponível em: <https://www.rfc-editor.org/info/rfc9470>.

BRADEN, R. T. **Requirements for Internet Hosts - Communication Layers**. RFC Editor, 1989. RFC 1122. (Request for Comments, 1122). Disponível em: <https://www.rfc-editor.org/info/rfc1122>.

BRAY, T. **The JavaScript Object Notation (JSON) Data Interchange Format**. RFC Editor, 2017. RFC 8259. (Request for Comments, 8259). Disponível em: <https://www.rfc-editor.org/info/rfc8259>.

FETT, D. *et al.* **OAuth 2.0 Demonstrating Proof of Possession (DPoP)**. RFC Editor, 2023. RFC 9449. (Request for Comments, 9449). Disponível em: <https://www.rfc-editor.org/info/rfc9449>.

FIELDING, R. Representational state transfer. **Architectural Styles and the Design of Netowork-based Software Architecture**, p. 76–85, 2000.

FIELDING, R. T.; NOTTINGHAM, M.; RESCHKE, J. **HTTP Semantics**. RFC Editor, 2022. RFC 9110. (Request for Comments, 9110). Disponível em: <https://www.rfc-editor.org/info/rfc9110>.

GRASSI, P. A.; GARCIA, M. E.; FENTON, J. L. **Digital identity guidelines: revision 3**. Gaithersburg, MD, 2017. NIST SP 800–63–3 p. Disponível em: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf>.

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>.

HEILMAN, E. *et al.* Openpubkey: Augmenting openid connect with user held signing keys. Cryptology ePrint Archive, Paper 2023/296, 2023. Disponível em: <https://eprint.iacr.org/2023/296>.

JONES, M. B. **JSON Web Key (JWK)**. RFC Editor, 2015. RFC 7517. (Request for Comments, 7517). Disponível em: <https://www.rfc-editor.org/info/rfc7517>.

JONES, M. B.; BRADLEY, J.; SAKIMURA, N. **JSON Web Signature (JWS)**. RFC Editor, 2015. RFC 7515. (Request for Comments, 7515). Disponível em: <https://www.rfc-editor.org/info/rfc7515>.

JONES, M. B.; BRADLEY, J.; SAKIMURA, N. **JSON Web Token (JWT)**. RFC Editor, 2015. RFC 7519. (Request for Comments, 7519). Disponível em: <https://www.rfc-editor.org/info/rfc7519>.

JONES, M. B.; HARDT, D. **The OAuth 2.0 Authorization Framework: Bearer Token Usage**. RFC Editor, 2012. RFC 6750. (Request for Comments, 6750). Disponível em: <https://www.rfc-editor.org/info/rfc6750>.

JONES, M. B.; SAKIMURA, N. **JSON Web Key (JWK) Thumbprint**. RFC Editor, 2015. RFC 7638. (Request for Comments, 7638). Disponível em: <https://www.rfc-editor.org/info/rfc7638>.

KAJAN, E. **Information technology encyclopedia and acronyms**. Springer Science & Business Media, 2002. Disponível em: <https://link.springer.com/book/10.1007/978-3-642-56262-4>.

LAMPSON, B. *et al.* Authentication in distributed systems: Theory and practice. **ACM Trans. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 10, n. 4, p. 265–310, nov 1992. ISSN 0734-2071. Disponível em: <https://doi.org/10.1145/138873.138874>.

MITALI, V. K.; SHARMA, A. *et al.* A survey on various cryptography techniques. **International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)**, v. 3, n. 4, p. 307–312, 2014.

MOODY, D. **Digital Signature Standard (DSS)**. Gaithersburg, MD, 2023. NIST FIPS 186–5 p. Disponível em: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf>.

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>.

SAKIMURA, N. *et al.* **Final: OpenID Connect Core 1.0**. 2014. S3 p. Disponível em: <https://openid.net/specs/openid-connect-core-1_0-final.html>.

SAMARATI, P.; VIMERCATI, S. C. de. Access control: Policies, models, and mechanisms. In: FOCARDI, R.; GORRIERI, R. (Ed.). **Foundations of Security Analysis and Design**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. p. 137–196. ISBN 978-3-540-45608-7. Disponível em: <https://doi.org/10.1007/3-540-45608-2_3>.

SHIREY, R. **RFC 4949: Internet security glossary, Version 2**. RFC Editor, 2007. Disponível em: <https://www.rfc-editor.org/rfc/rfc4949>.

STEEN, M. R. van; SIPS, H. *et al.* Computer and network organization: an introduction. onbekendPrentice Hall, 1995. Disponível em: <https://www.distributed-systems.net/index.php/books/computer-and-network-organization/>.

STEEN, M. van; TANENBAUM, A. **Distributed Systems**. 4. ed. [s.n.], 2023. Disponível em: <https://www.distributed-systems.net/index.php/books/ds4/>.

STEIN, P. **On the Internet, Nobody Knows You're a Dog**. 1993. 61 p. The New Yorker. Disponível em: <https://www.plsteiner.com/cartoons#/newyorker>.

WILSON, Y.; HINGNIKAR, A. **Solving Identity Management in Modern Applications: Demystifying OAuth 2, OpenID Connect, and SAML 2**. Berkeley, CA: Apress, 2023. ISBN 9781484282601 9781484282618. Disponível em: <https://link.springer.com/10.1007/978-1-4842-8261-8>.

Anexos

ANEXO A – Definição do header - cabeçalho - de um DPoP JWT

Figura A.1 – Definição do *header* - cabeçalho - de um DPoP JWT

```
type Header struct {
    Type        string    `json:"typ"`
    Algorithm   Algorithm `json:"alg"`
    Jwk         JWK       `json:"jwk"`
    ContentType string    `json:"cty,omitempty"`
    KeyID       string    `json:"kid,omitempty"`
}
```

Fonte: Adaptação da implementação de cristalhq/jwt/v5

ANEXO B – Definição do payload de um DPoP JWT

Figura B.1 – Definição do *payload* de um DPoP JWT

```
type RegisteredClaims struct {
    Jti string `json:"jti"`
    Htm string `json:"htm"`
    Htu string `json:"htu"`
    Iat *NumericDate `json:"iat"`
    Nonce string `json:"nonce,omitempty"`
    Cnf Cnf `json:"cnf,omitempty"`
}
```

Fonte: Adaptação da implementação de cristalhq/jwt/v5

ANEXO C – Definição da reivindicação cnf - confirmation - de um DPoP JWT

Figura C.1 – Definição da reivindicação `cnf` - *confirmation* - de um DPoP JWT

```
type Cnf struct {
    Jkt string `json:"jkt,omitempty"`
}
```

Fonte: De autoria própria

ANEXO D – Função que gera nonces

Figura D.1 – Função que gera nonces

```go
func GenerateNonce(length int) string {
    const nqchar =
    ↪  "!#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]
    ↪  ^_`abcdefghijklmnopqrstuvwxyz{|}~"
    nonce := make([]byte, length)
    for i := 0; i < length; i++ {
        index := rand.Intn(len(nqchar))
        nonce[i] = nqchar[index]
    }
    return string(nonce)
}
```

Fonte: De autoria própria om base nas regras de produção contidas no RFC 9449 e no apêndice A do RFC 6749

ANEXO E – Função que calcula o jkt de uma jwk

Figura E.1 – Função que calcula o `jkt` de uma `jwk`

```go
func Jkt(jwk JWK) (string, error) {

    jwkFields := JWK{
        Kty: jwk.Kty,
        Crv: jwk.Crv,
        X:   jwk.X,
        Y:   jwk.Y,
    }

    jwkFieldsJSON, err := json.Marshal(jwkFields)

    if err != nil {
        return "", err
    }

    sha256Hash := sha256.Sum256(jwkFieldsJSON)

    jkt := base64.URLEncoding.EncodeToString(sha256Hash[:])

    return strings.TrimRight(jkt, "="), nil
}
```

Fonte: De autoria própria seguindo a especificação do RFC 7638

ANEXO F – Função que implementa o handler da rota /nonce

Figura F.1 – Função que implementa o *handler* da rota /nonce

```go
func nonceHandler(w http.ResponseWriter, r *http.Request) {

    if r.Method != "POST" {
        log.Printf("error=\"Invalid Request\" path=%s method=%s",
        ↪  r.URL.Path, r.Method)
        http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        return
    }

    value := r.Header.Values("jkt")[0]

    if value == "" {
        http.Error(w, "No 'jkt' provided in request header",
        ↪  http.StatusBadRequest)
    } else {

        nonce := dpopjwt.GenerateNonce(24)

        nonces.Lock()
        nonces.m[value] = nonce
        nonces.Unlock()

        w.Header().Set("Cache-Control", "no-store")
        w.Header().Set("DPoP-Nonce", nonce)
        w.WriteHeader(http.StatusCreated)

    }

}
```

Fonte: De autoria própria

ANEXO G – Função que implementa o handler da rota /token

Figura G.1 – Função que implementa o handler da rota /token

```go
func tokenHandler(w http.ResponseWriter, r *http.Request) {

    if r.Method != "POST" {
        log.Printf("error=\"Invalid Request\" path=%s method=%s",
        ↪  r.URL.Path, r.Method)
        http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        return
    }

    w.Header().Set("Content-Type", "application/json")
    w.Header().Set("Cache-Control", "no-store")
    w.WriteHeader(http.StatusOK)

    response := AccessTokenResponse{
        AccessToken:  "Kz~8mXK1EalYznwH-LC-1fBAo.4Ljp~zsPE_NeO.gxU",
        TokenType:    "DPoP",
        Expires:      dpopjwt.NewNumericDate(time.Now()),
        RefreshToken: "Q..Zkm29lexi8VnWg2zPW1x-tgGad0Ibc3s3EwM_Ni4-g",
    }

    responseJSON, err := json.MarshalIndent(response, "", " ")

    if err != nil {
        http.Error(w, "Error while encoding response",
        ↪  http.StatusInternalServerError)
        return
    }

    w.Write(responseJSON)
}
```

Fonte: De autoria própria

ANEXO H – Função para gerar DPoP-Proof JWTs

Figura H.1 – Função para gerar DPoP-Proof JWTs

```go
func CreateExampleDPoPJWT(key *ecdsa.PrivateKey, jwk JWK) *Token {

    jkt, _ := Jkt(jwk)

    signer, _ := NewSignerES(ES256, key)

    claims := &RegisteredClaims{
        Jti:   uuid.NewString(),
        Htm:   "GET",
        Htu:   "https://server.example.com/token",
        Iat:   NewNumericDate(time.Now()),
        Nonce: GenerateNonce(24),
        Cnf: Cnf{
            Jkt: jkt,
        },
    }

    opts := []BuilderOption{
        WithJWK(jwk),
        WithTyp("dpop+jwt"),
    }

    builder := NewBuilder(signer, opts...)

    token, err := builder.Build(claims)

    if err != nil {
        log.Fatal("error while building token:", err)
        return nil
    }

    return token
}
```

Fonte: De autoria própria

ANEXO I – Função para gerar requisições por tokens ligados a DPoP

Figura I.1 – Função para gerar requisições por tokens ligados a DPoP

```go
func createDPoPBoundTokenRequest(token dpopjwt.Token, url string) (string, error) {

    client := &http.Client{}

    req, _ := http.NewRequest("POST", url, nil)

    req.Header.Set("Content-Type", "application/x-www-form-urlencoded")
    req.Header.Set("DPoP", token.String())

    resp, err := client.Do(req)

    if err != nil {
        log.Fatal("Error sending request:", err)
        return "", err
    }

    defer resp.Body.Close()

    if resp.StatusCode == 200 {

        buf := new(bytes.Buffer)
        _, err := buf.ReadFrom(resp.Body)

        if err != nil {
            log.Fatal("Error reading response body:", err)
            return "", err
        }

        return buf.String(), nil
    }

    return "", errors.New("unable to generate access token")
}
```

Fonte: De autoria própria

ANEXO J – Função para validação de um nonce com base nas regras de produção contidas no RFC 9449 e no apêndice A do RFC 6749

Figura J.1 – Função para validação de um *nonce*

```
1  func CheckNonce(nonce string) (bool, error) {
2
3      const pattern = `^(?:!|[\#\$\%\&\'\(\)\*\+\,\-\.\/]|[0-9]
       ↪  |[\:\;\<\=\>\?\@]|[A-Za-z]|[\[\]\^_\x60\{\\|\}\~])+$`
4
5      re := regexp.MustCompile(pattern)
6
7      if re.MatchString(nonce) {
8          return true, nil
9      }
10
11     return false, errors.New("no match")
12 }
```

Fonte: De autoria própria com base nas regras de produção contidas no RFC 9449 e no apêndice A do RFC 6749

ANEXO K – Conjunto de testes para validação das funções GenerateNonce e CheckNonce

Figura K.1 – Conjunto de testes para validação das funções `GenerateNonce` e `CheckNonce`

```go
func TestNonce(t *testing.T) {
    testCases := []struct {
        nonce string
        valid bool
        err   error
    }{
        {
            GenerateNonce(24),
            true,
            nil,
        },
        {
            GenerateNonce(24) + `"`,
            false,
            errors.New("no match"),
        },
        {
            GenerateNonce(12) + `"` + GenerateNonce(12),
            false,
            errors.New("no match"),
        },
        {
            `"` + GenerateNonce(24),
            false,
            errors.New("no match"),
        },
    }

    for _, tc := range testCases {

        valid, err := CheckNonce(tc.nonce)

        if valid != tc.valid && err != tc.err {
            t.Log("tc.nonce:", tc.nonce)
            t.Log("tc.valid:", tc.valid)
            t.Log("tc.err", tc.err)
            t.Log("valid", valid)
            t.Log("err", err)
            t.Fatal("unexpected behaviour")
        }
    }
}
```

Fonte: De autoria própria

ANEXO L – Conjunto de testes para validação da função Jkt

Figura L.1 – Conjunto de testes para validação da função `Jkt`

```go
func TestJkt(t *testing.T) {
    jwk := dpopjwt.JWK{
        Crv: "P-256",
        Kty: "EC",
        X:   "l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
        Y:   "9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    }

    res, _ := dpopjwt.Jkt(jwk)

    testCases := []JktTableTest{
        {
            jwk:    jwk,
            jkt:    res,
            expect: "0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I",
        },
    }

    for _, tc := range testCases {

        valid := tc.jkt == tc.expect

        if !valid {
            t.Log("tc.jwk:", tc.jwk)
            t.Log("tc.jkt:", tc.jkt)
            t.Log("tc.expect:", tc.expect)
            t.Fatal("unexpected result")
        }

    }
}
```

Fonte: De autoria própria

ANEXO M – Teste para validação do construtor de DPoP-Proof JWT

Figura M.1 – Teste para validação do construtor de DPoP-Proof JWT

```
1  func TestDPoPProofBuild(t *testing.T) {
2      t.Log("Generate key pair")
3      key, jwk, _ := CreateKey()
4      t.Log("Craft DPoP JWT with example claims")
5      token := createExampleDPoPJWT(key, jwk)
6      t.Log("Build verifier")
7      verifier, err := NewVerifierES(ES256, &key.PublicKey)
8      if err != nil {
9          t.Error("\tError while building verifier")
10     }
11     t.Log("Parse the token")
12     tokenBytes := token.Bytes()
13     newToken, err := Parse(tokenBytes, verifier)
14     if err != nil {
15         t.Error("\tError while parsing the token:", err)
16     }
17     t.Log("Verify token signature")
18     err = verifier.Verify(newToken)
19     if err != nil {
20         t.Error("\tFailed to verify token signature. Got:", err)
21     }
22     t.Log("Get registered claims")
23     var newClaims RegisteredClaims
24     errClaims := json.Unmarshal(token.Claims(), &newClaims)
25     if errClaims != nil {
26         t.Error("\tFailed to unmarshal claims. Got:", errClaims)
27     }
28     t.Log("Parse claims")
29     errParseClaims := ParseClaims(token.Bytes(), verifier, &newClaims)
30     if errParseClaims != nil {
31         t.Error("\tFailed to parse claims. Got:", errParseClaims)
32     }
33 }
```

Fonte: De autoria própria

ANEXO N – Teste para validação da rota /nonce do servidor de autorização

Figura N.1 – Teste para validação da rota `/nonce` do servidor de autorização

```go
func TestNonceHandler(t *testing.T) {

    t.Log("Assemble POST request to /nonce")
    req, _ := http.NewRequest("POST", "/nonce", nil)

    t.Log("Set required header value")
    req.Header.Set("jkt", "example-value")

    t.Log("Create response recorder")
    w := httptest.NewRecorder()

    t.Log("Send the request")
    nonceHandler(w, req)

    resp := w.Result()

    t.Log("Evaluate the response")
    nonce := resp.Header.Values("DPoP-Nonce")[0]
    cache_control := resp.Header["Cache-Control"][0]

    t.Log("Value in DPoP-Nonce header should be non-empty")
    if nonce == "" {
        t.Error("Expected non-empty string in DPoP-Nonce header\n")
    }

    t.Log("Value in DPoP-Nonce should be a valid nonce")
    if flag, _ := dpopjwt.CheckNonce(nonce); flag == false {
        t.Errorf("Received invalid nonce from %s", req.URL.Path)
    }

    t.Log("Value in Cache-Control should be no-store")
    if cache_control != "no-store" {
        t.Error("Expected no-store string in Cache-Control header\n")
    }

    t.Log("HTTP Status Code should be 201 (Created)")
    if resp.StatusCode != http.StatusCreated {
        t.Errorf("Expected response status: %v, Got: %v\n", http.StatusOK,
        ↪  resp.StatusCode)
    }
}
```

Fonte: De autoria própria

ANEXO O – Teste para validação da rota /token do servidor de autorização

Figura O.1 – Teste para validação da rota `/token` do servidor de autorização

```go
func TestTokenHandler(t *testing.T) {
    t.Log("Assemble POST request to /token")
    req, _ := http.NewRequest("POST", "/token", nil)
    t.Log("Set the required headers")
    req.Header.Set("Content-Type", "application/x-www-form-urlencoded")
    req.Header.Set("DPoP", "")
    t.Log("Create response recorder")
    w := httptest.NewRecorder()
    t.Log("Send the request")
    tokenHandler(w, req)
    resp := w.Result()
    t.Log("Evaluate the response")
    if resp.StatusCode != http.StatusOK {
        t.Errorf("Expected response status: %v, Got: %v\n", http.StatusOK,
        ↪ resp.StatusCode)
    }
    content_type := resp.Header.Values("Content-Type")[0]
    if content_type != "application/json" {
        t.Error("Expected response to be of type application/json\n")
    }
    cache_control := resp.Header.Values("Cache-Control")[0]
    if cache_control != "no-store" {
        t.Error("Expected no-store string in Cache-Control header\n")
    }
    defer resp.Body.Close()
    t.Log("Read content of response body")
    buf := new(bytes.Buffer)
    _, err = buf.ReadFrom(resp.Body)
    t.Log("Expect content of body to be unmarshalable")
    var contentJSON map[string]interface{}
    err = json.Unmarshal(buf.Bytes(), &contentJSON)
    if err != nil {
        t.Error("Error while unmarshaling response body")
    }
    t.Log("Expect response body to contain a 'token_type' key with value
    ↪ 'DPoP'")
    value, exists := contentJSON["token_type"]
    if !exists {
        t.Error("Response should contain 'token_type'")
    }
    if strings.ToLower(value.(string)) != "dpop" {
        t.Errorf("Value in token_type is not dpop. Got %s",
        ↪ value.(string))
    }
}
```

Fonte: De autoria própria