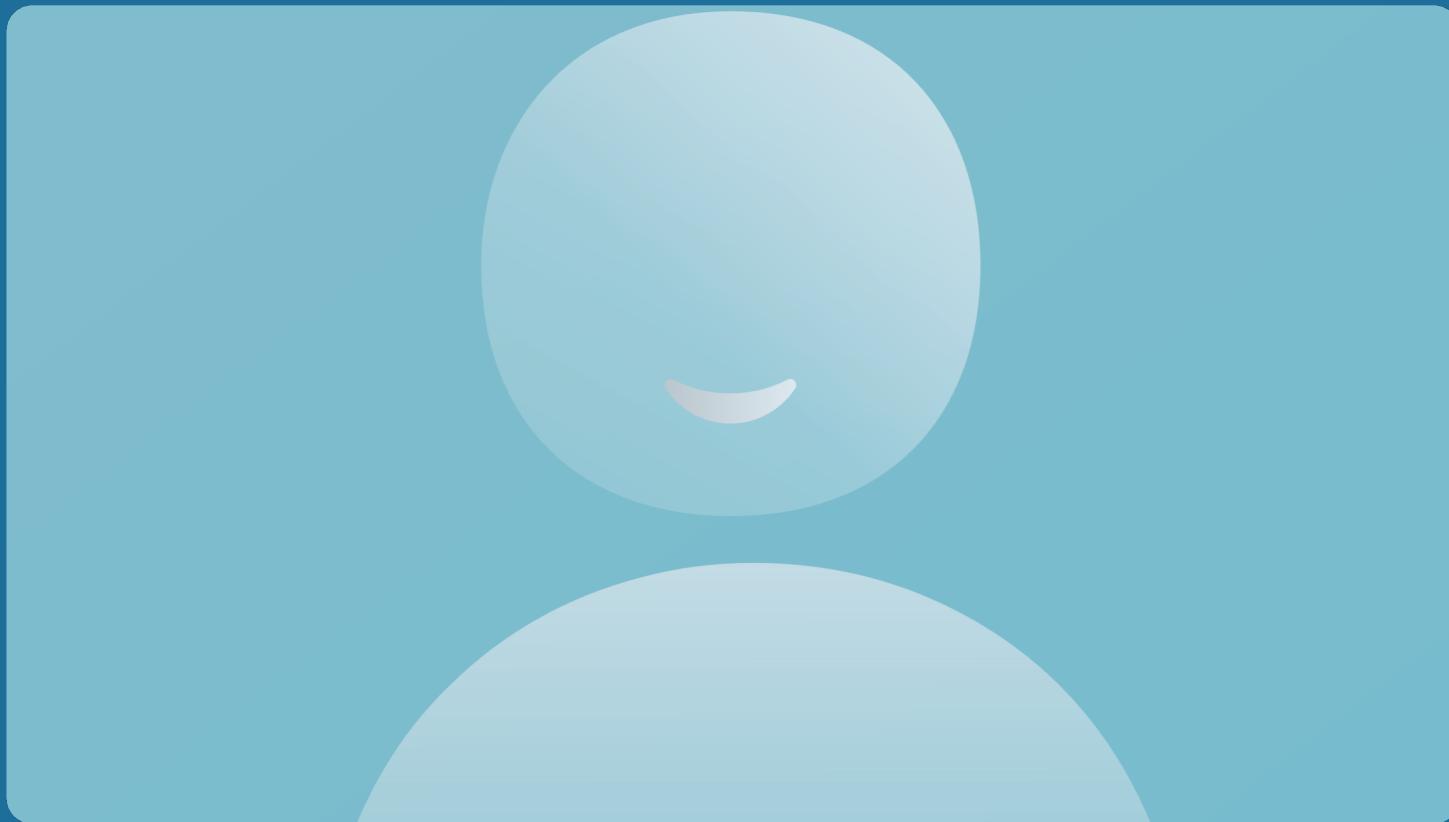


APIs e Web Services



Tópicos

- Unidade 1 - Introdução
 - Introdução ao Mundo das APIs
 - Protocolo HTTP
- Unidade 2 - Web APIs
 - Aspectos arquiteturais de APIs
 - Estilos: SOAP, REST, GraphQL, Webhooks e Websockets
- Unidade 3 - Segurança em APIs
 - Fundamentos de Segurança
 - Autenticação no Protocolo HTTP
 - Tecnologias e Frameworks: JWT, Oauth e CORS
- Unidade 4 - Gerenciamento de APIs
 - API Lifecycle Management (ALM)
 - Arquitetura de Microserviços e API Gateway
 - Boas práticas no desenvolvimento de APIs

APIs & Web Services

Informações Gerais

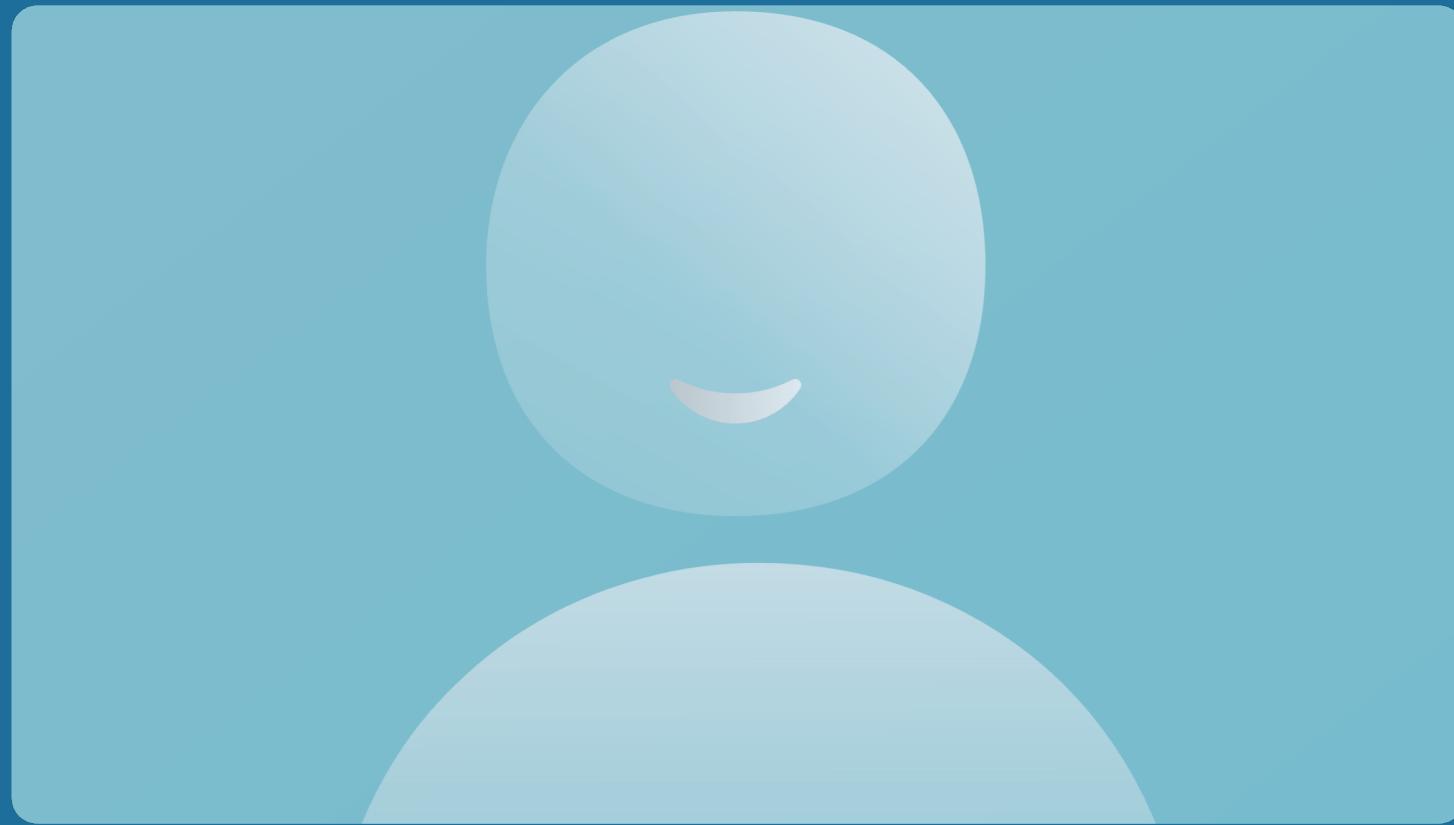
Ementa

Evolução das APIs. Gestão do ciclo de vida das APIs. Melhores práticas no projeto de API ferramentas para documentação de APIs. Mecanismos de segurança: autenticação, a vulnerabilidades. Abordagens arquiteturais de APIs: RESTful, GraphQL, WebSockets, WebF Streaming.

Bibliografia

- [Designing Web APIs](#). Brenda Jin, Saurabh Sahni, Amir Shevat. O'Reilly Media, Inc. (2018)
- [Mastering API Architecture](#). James Gough, Daniel Bryant, Matthew Auburn. O'Reilly Media, Inc. (2022)
- [API Design Patterns](#). John J. (JJ) Geewax. Manning Publications (2021)
- [Designing APIs with Swagger and OpenAPI](#). Josh Ponelat, Lukas Rosenstock. Manning Publications (2022)
- [Microservice APIs](#). Jose Haro. Manning Publications (2023)

Fundamentos de Segurança



Autenticação vs Autorização



Autenticação

Verifica se o usuário é quem diz ser

Autorização

Verifica as permissões de acesso do usuário

Fontes:

- [Authentication vs. Authorization \(okta\)](#)



170



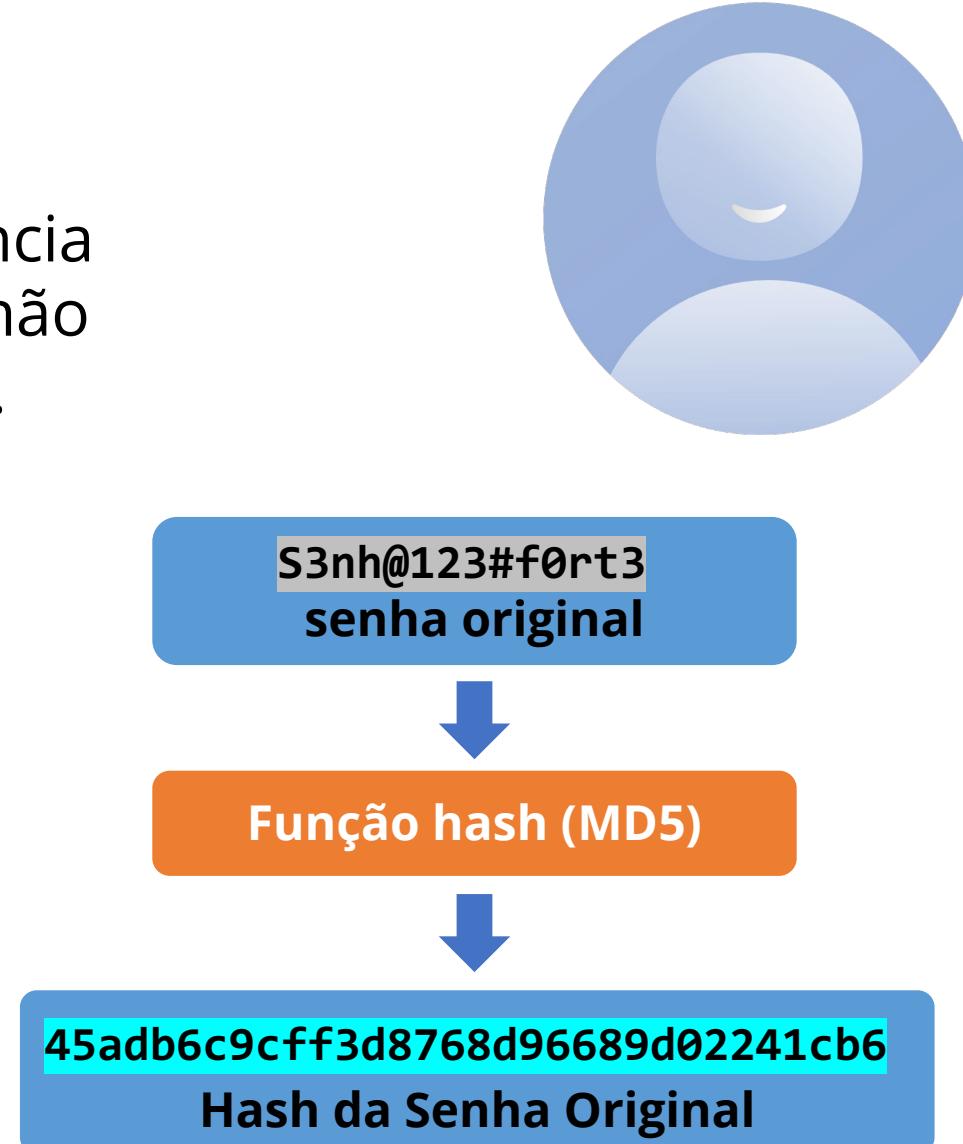
Prof. Rommel Vieira Carneiro

Message Digest – Função Hash

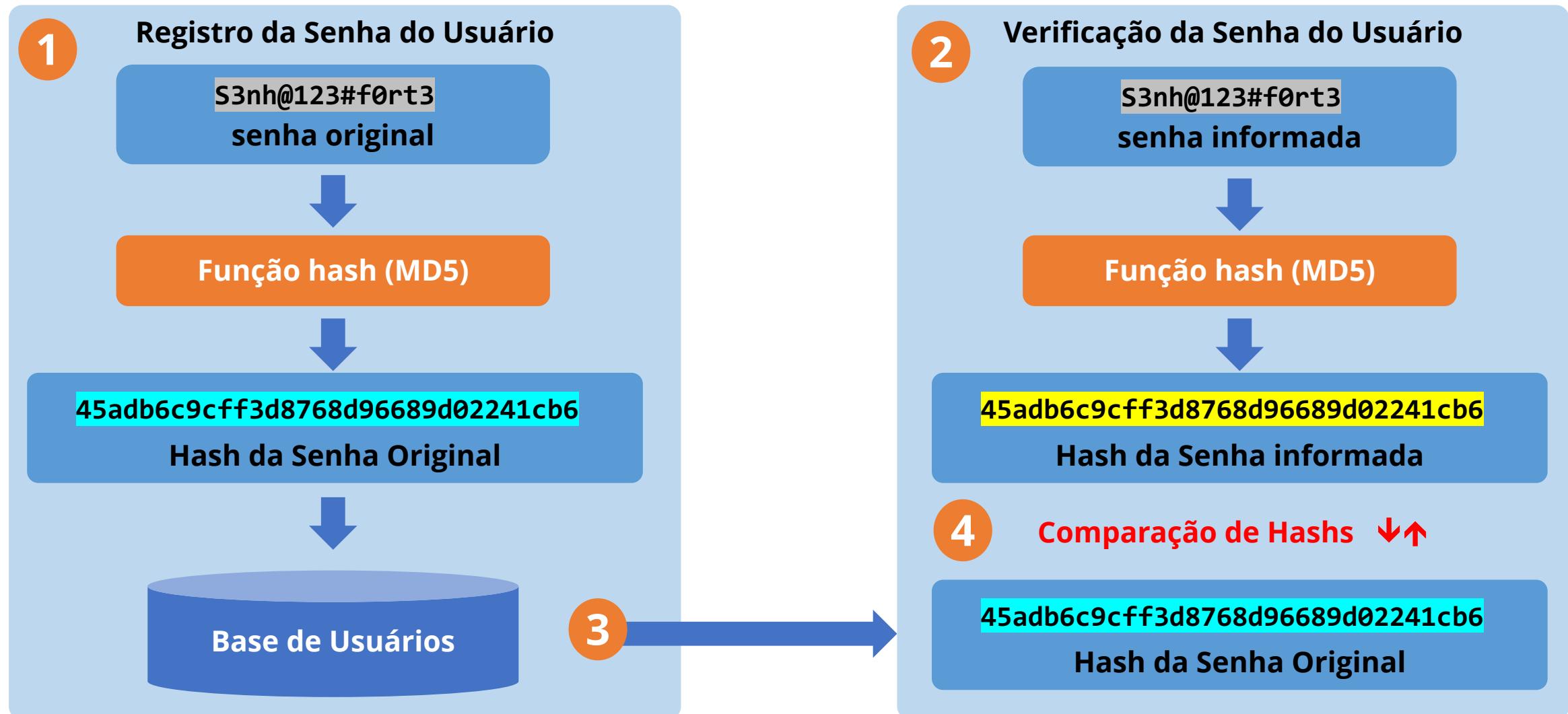
Função hash: algoritmo que produz uma sequência diferentes para cada entrada e de tamanho fixo não reversíveis (não restauram a mensagem original).

Algoritmos

- **Message Digest Algorithm (MD5)**
 - Resumo de mensagem de 128bits
 - Documentação RFC-1321
- **SHA-1**
 - Resumo de mensagem de 160bits
 - Padrão no EUA



Senhas seguras com Hash



Codificação Base64

Método utilizado para converter dados binários em texto e vice-versa.



```
iVBORw0KGgoAAAANSUhEUgAAABAAAAAQCAYAAAAf8/9hAAAABGdBTUEAAK/INwWK6QAAABl0RVh0U29mdHdhcmUAQWRvYmUgSW1hZ2VSZWfkeXHJZTwAAANdSURBVHjaYvz//z8DJQAggBh9fecyfPr8i4GP15vh+/cfDL9//WVgYeVk+P37H80/v/8Zvv34K8TMzJKiqa5Yd0f00e/Hjjc7MDD8fMjAwAw2ACCAWHCZ/PPnLx4mJtYKJSXJJG8fTUlTY2WGbbS VGS5fXVH1+d0VdJg6gADCMODvv39Mf3/8rlNSls60d9AU s30QZZASY2M4d+UPg42zPIPZ1tCQvXuu1DEw/HvJwMDEABBA YAP+/fvP80f3X67//xlrlBTl0mzstYStrIEaJVkZ+HkZGC5e+M7w5z8g7QkA40vf4jQoYOLin//vlsGtI4BIIBABo hxcbHnSUqJpFvZao lYWMozSIozM7CxMTBwsTMwvHv7h+H1278M8kocDMx/GBicnbQYDAwDok+f6m0B6v0EEEAsgoI8 J+3t1RRs7LQY5GTZGNhYgQ5jQnj p7q3vDKLCQJfwMDIwACNMUQHkjDJs2ewZ//796wdIICYjYx8W/TMNNmevXgHVP ye4c2r7wzfvv8FxgAjw70nvxg+fvzLoKLKzcDCwsjADAx4Lk4GBgEBccb9B+4ovH1zdhFAADH//cuu50Hjb6SgLs7A zSHNAAwHhk8ffjM8evid4d27nwzqmnwMfHwsYFexAD3MDKR1ZJgYXr8XEzx0YMdLgABifvfu7W1RQc140ztddh40Vg YxYU4GWVluBkVFHgZFJW4GTK5IRDEDKS4uiCHPnjEwnDrzkvH0qV2MAAEEdBYvg65u3Kqjxz79f/3m//nz/7/fw0k P378///r1///v3///v3/9g8BYoPnvWnf9mZg1/2NnVzzMyMvsDBBDQVDEGDg4l47rabd8/f/7///2H//8/APGXL/ ///kD0QgycNbMG/9tbet+s7GpngM6KB6I0UAuAwggBi5uaWDSFWIwMkzf c+Xyj/+fyM0vn717//c2Tf/W1tV/2Zm VjwFVB+LnvgAAoiBg10CgY1dmIGH18i+r+f4b5DGly9BNl7+b21d8YuFRek0UF0UQgdIPy0cBxBALhx80sD4/c/w7z /bwc1b1p9mZW013LBh7e8DB5ed/fv3YR9QzXog/oMrzwAEECMnpxDYRGBQgaJQ6+8/j sI/v58fAgoux64R5IK/DOBU BQQAAQYAP5FRv1nW25oAAAAASUVORK5CYII=
```

Base64 é muito utilizado na Internet pois vários protocolos de aplicação aceitam apenas dados em texto (email, http). É usado em conjunto com o padrão MIME.

Codificação Base64

Senhas são codificadas via Base64 para garantir que os caracteres a serem enviados na comunicação são strings ASCII válidas



Rommel : Senha@123



Um9tbWVsO1N1bmhhQDEyMw==

IMPORTANTE

Embora seja uma codificação, Base64 não é uma forma de criptografia e deve ser considerado como dados sendo passados de forma aberta.

Internet X.509 Public Key Infrastructure (PKI)

Conceitos

- **ICP - Infraestrutura de Chaves Públicas**

Estrutura que abrange um conjunto de entidades AC Raiz, ACs, ARs, certificados para prover requisitos de segurança em comunicação de sistemas.

- **AC Raiz - Autoridade Certificadora Raiz**

Estabelece a cadeia hierárquica de certificados. Estabelece a política de certificados, controla certificados das ACs e mantém a lista de certificados revogados

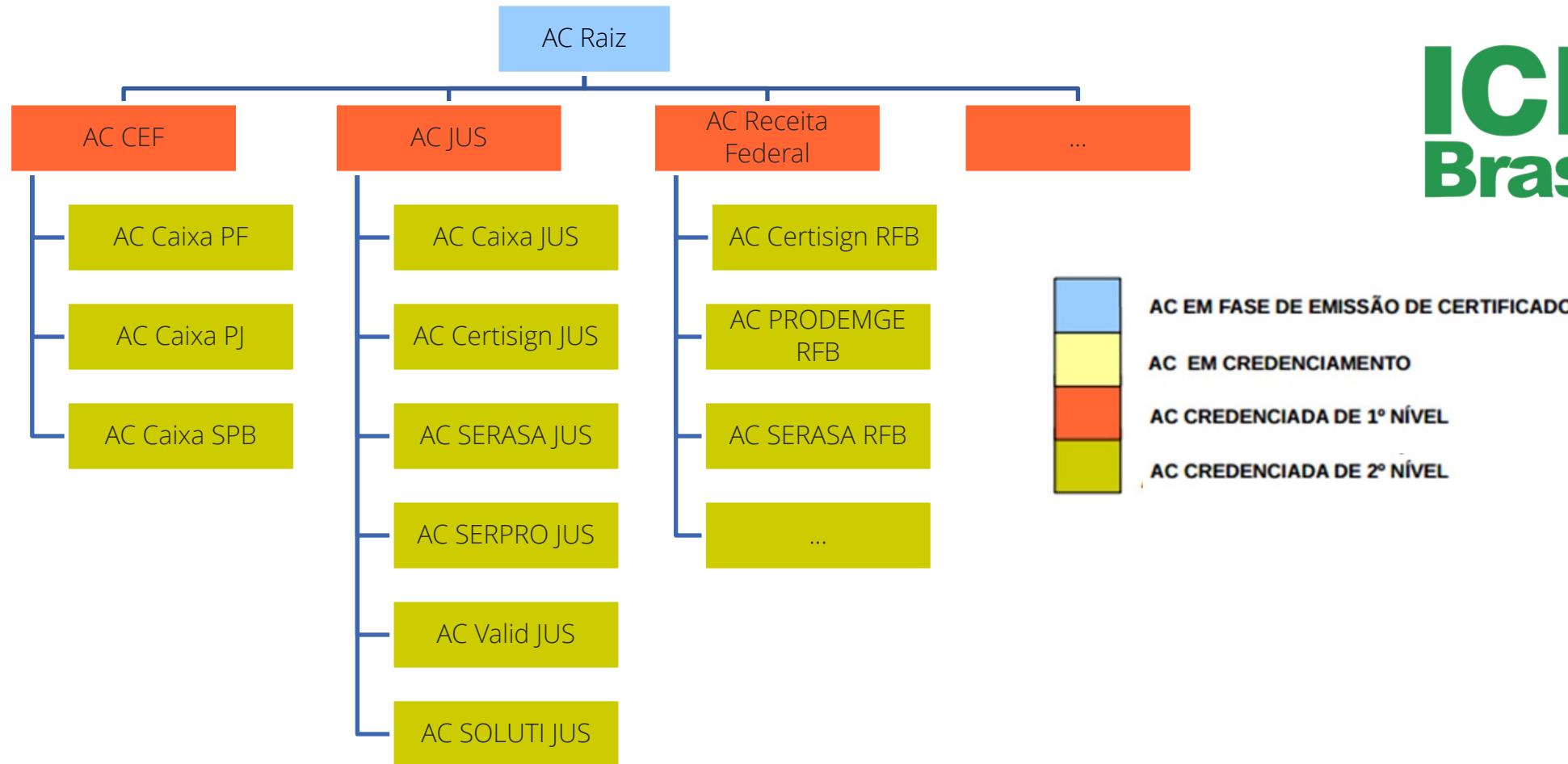
- **AC - Autoridade Certificadora**

Responsável por emitir, distribuir, renovar e gerenciar certificados digitais.

- **Certificado Digital**

Arquivo eletrônico contendo a identidade digital de uma entidade reconhecida pela ICP. Contem: nome da entidade, período de validade, chave pública, nome e assinatura da entidade que assinou o certificado, número de série.

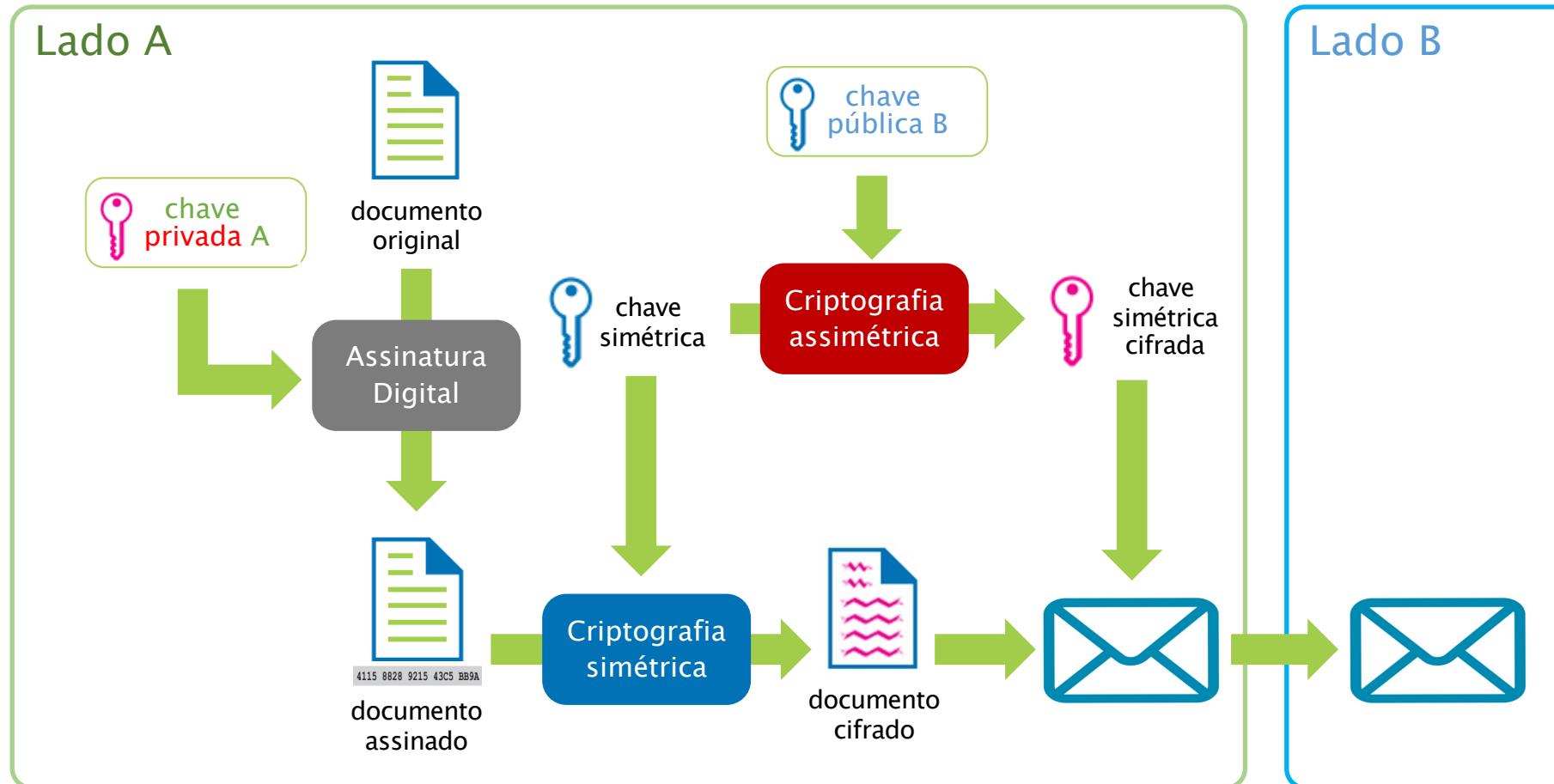
Internet X.509 Public Key Infrastructure (PKI)



Fonte: [Estrutura da ICP – Brasil](#)

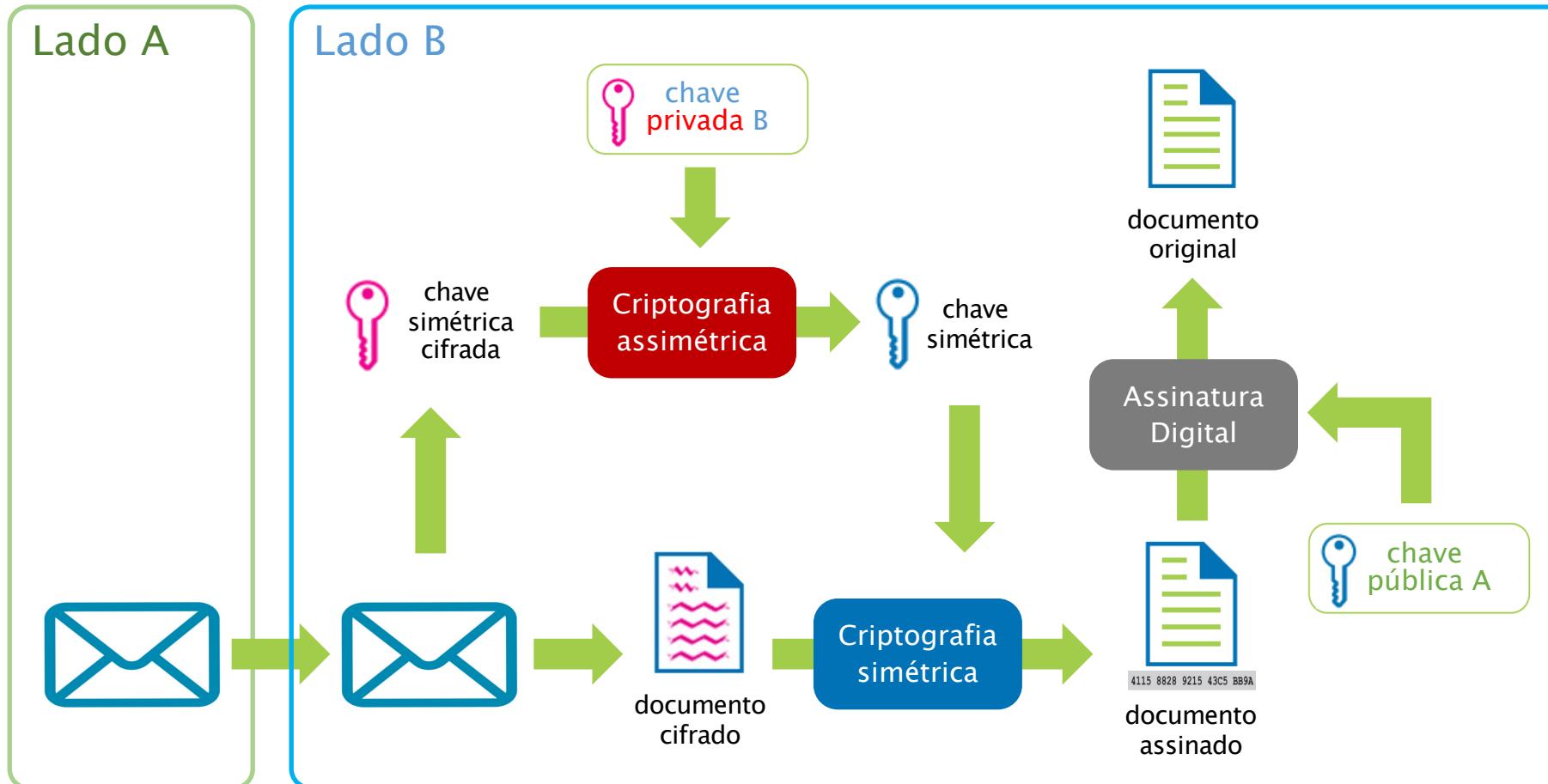
Internet X.509 Public Key Infrastructure (PKI)

Certificação Digital – Envio de Mensagem



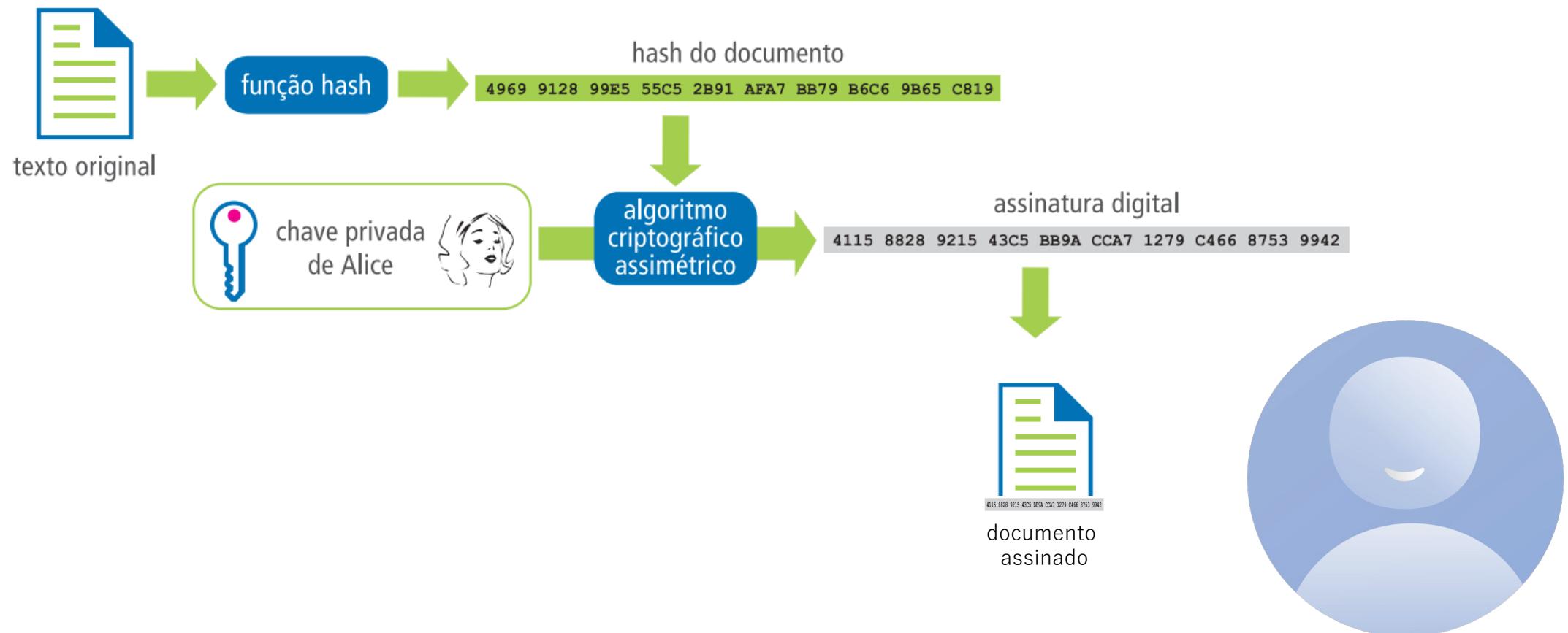
Internet X.509 Public Key Infrastructure (PKI)

Certificação Digital – Recebimento de Mensagem



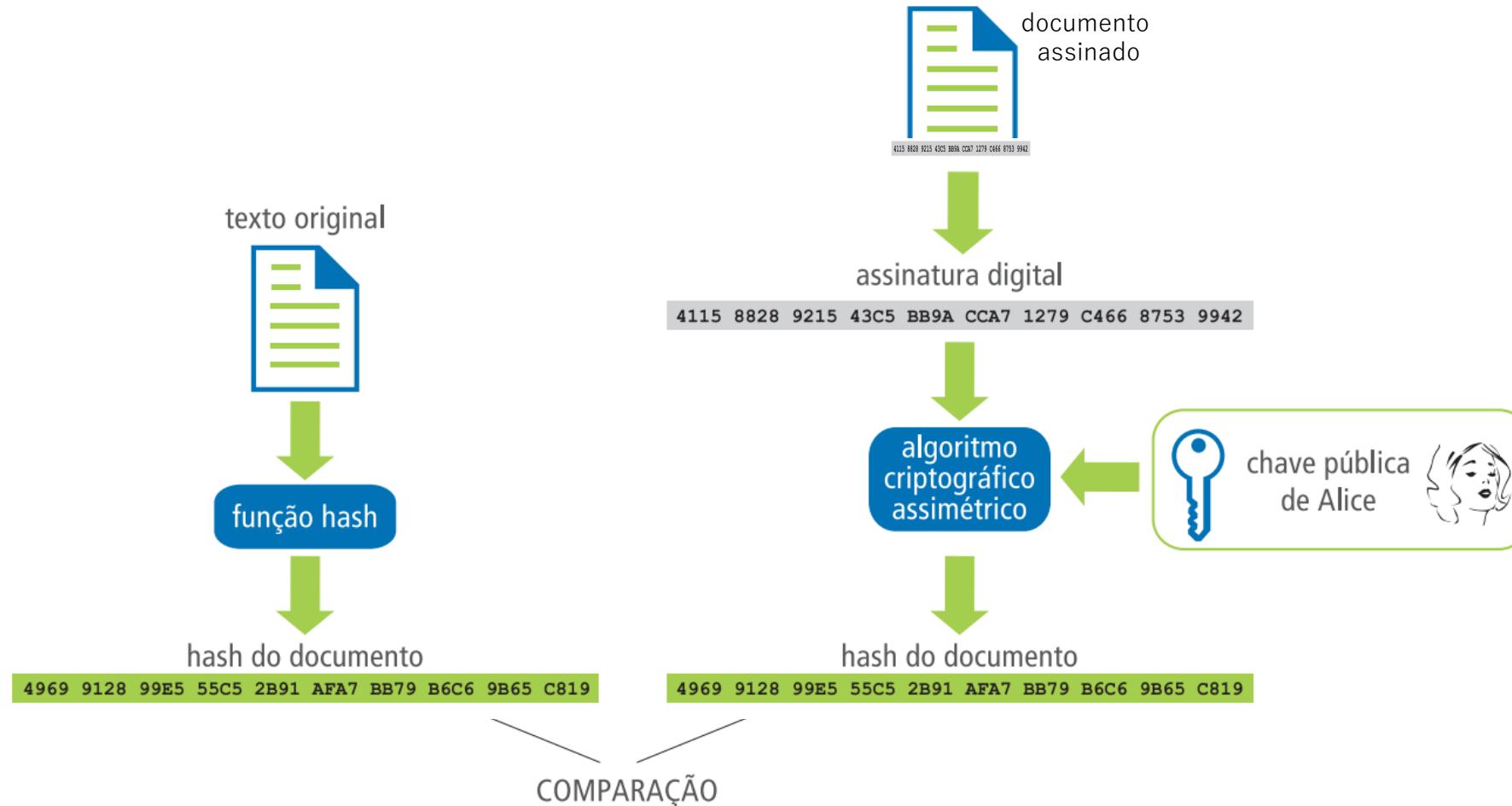
Internet X.509 Public Key Infrastructure (PKI)

Certificação Digital - Assinatura Digital



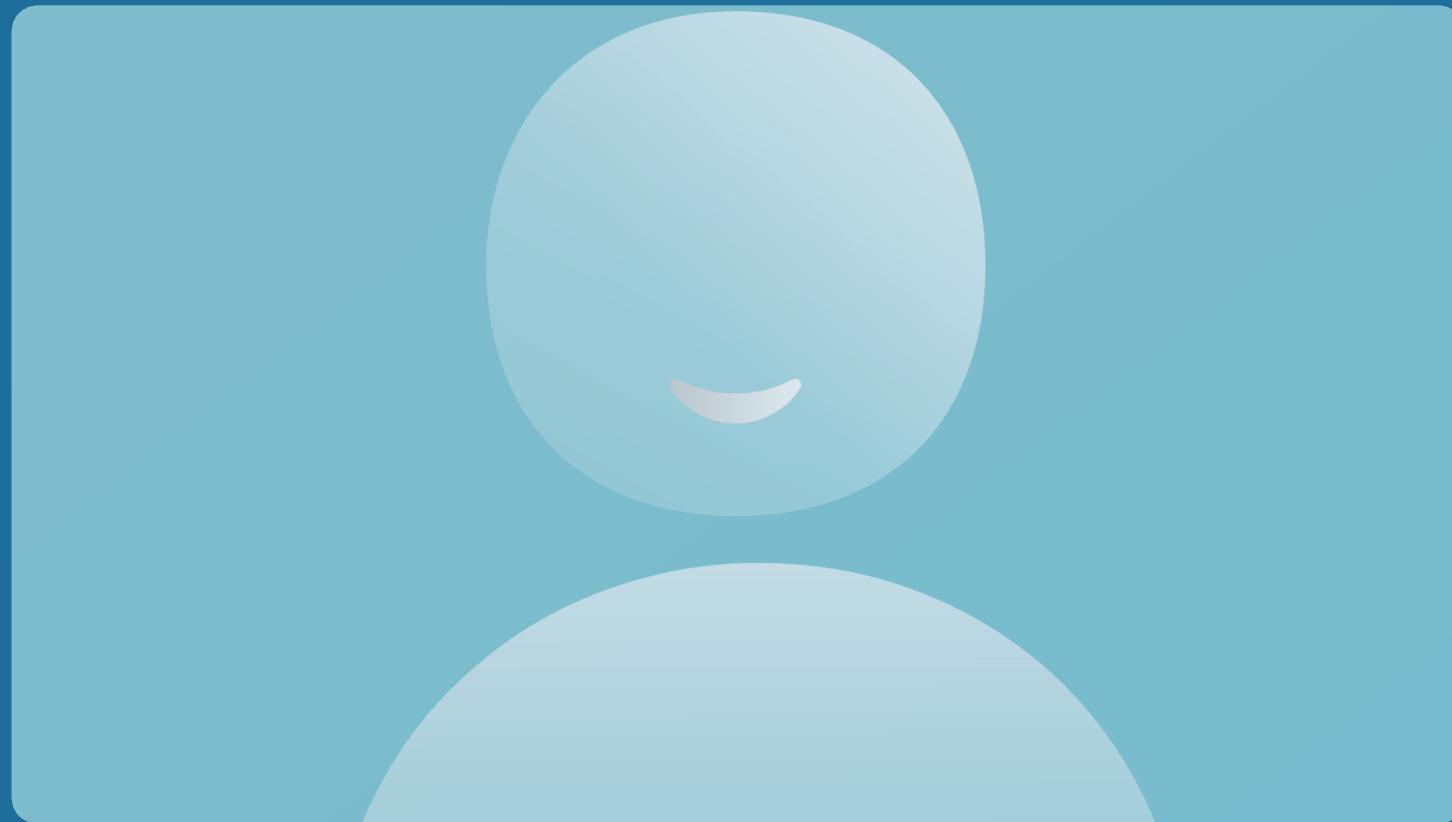
Internet X.509 Public Key Infrastructure (PKI)

Certificação Digital – Verificação de Assinatura Digital



Obrigado!

Autenticação no protocolo HTTP



Autenticação HTTP

Processo para verificar a identidade do usuário de uma aplicação Web.



Esquemas de Autenticação

- Usuário Anônimo + Forms da aplicação (Login)
- Autenticação Basic
- Autenticação Digest
- Autenticação Bearer (Token Authentication)

Fonte: RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication (<https://tools.ietf.org/html/rfc2617>)



183



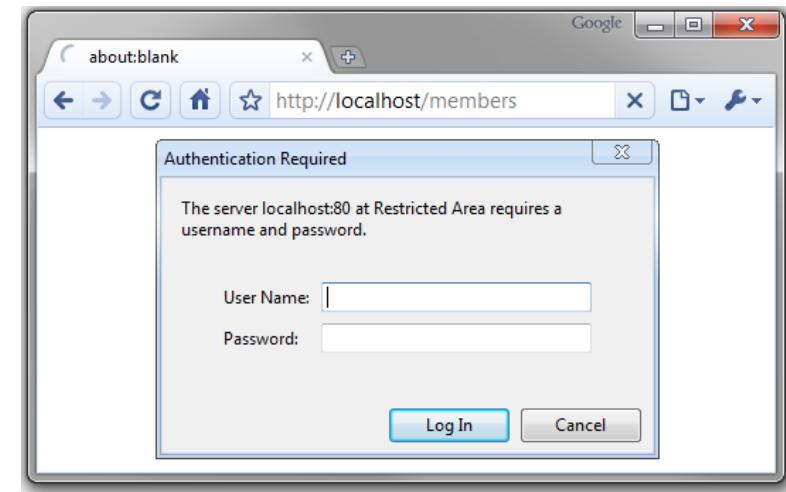
Prof. Rommel Vieira Carneiro

Autenticação HTTP – Basic



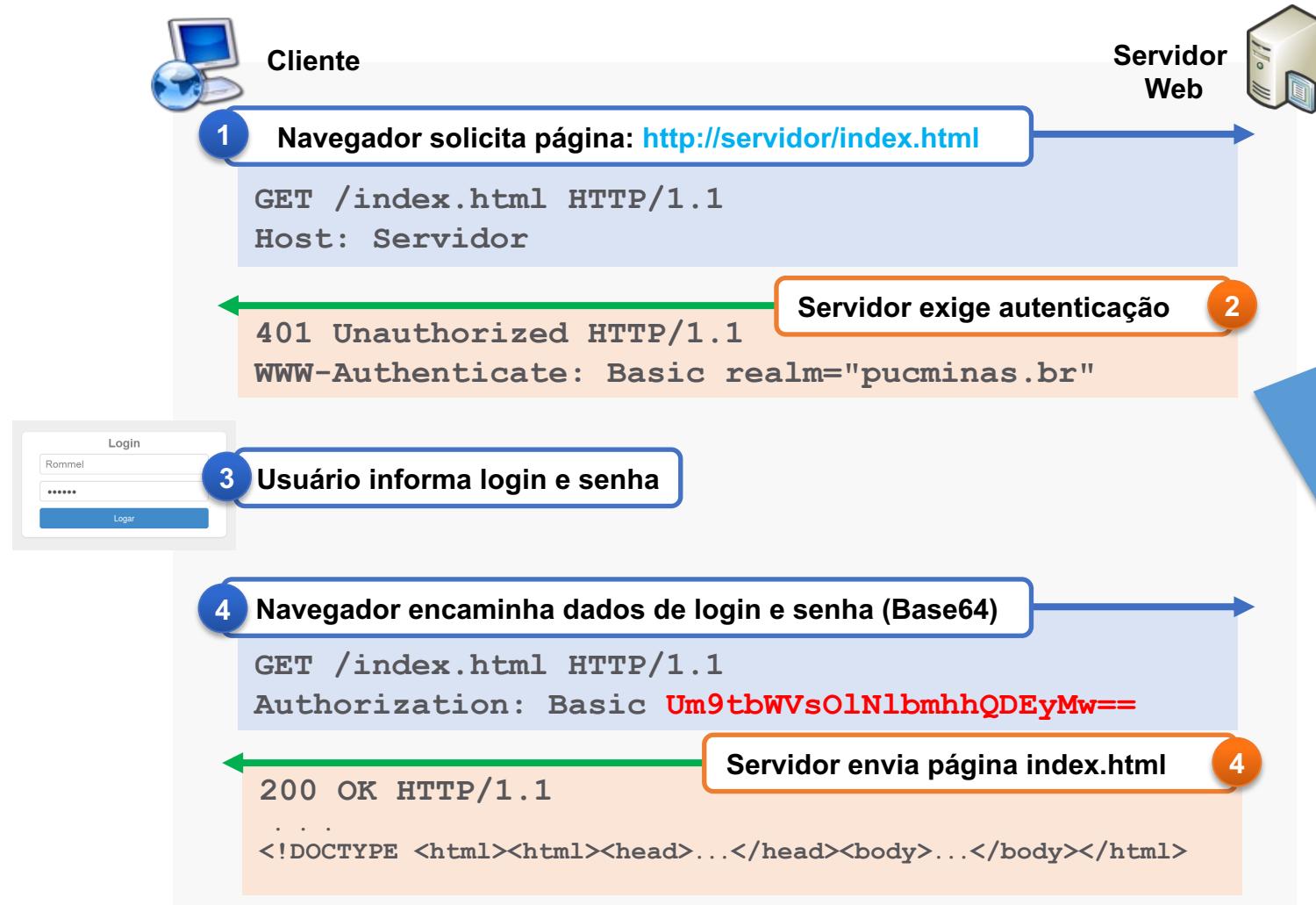
Tela de login exibida pelo próprio browser e envio de *string* codificada em Base64 com informação de usuário e senha.

IMPORTANTE: Recomenda-se utilizar apenas com conexões HTTPS.



Fonte: RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication (<https://tools.ietf.org/html/rfc2617>)

Autenticação HTTP – Basic

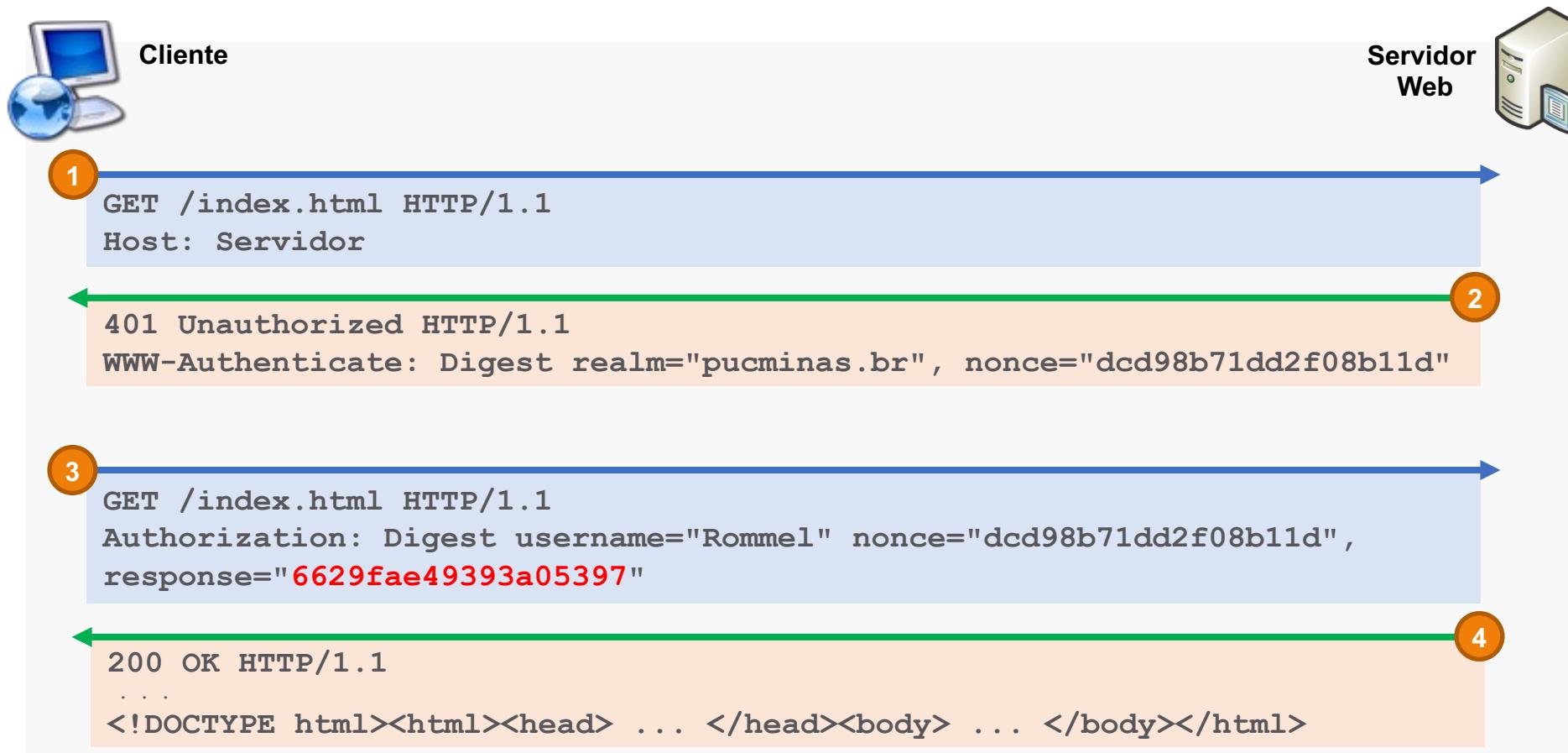


Informações

- **WWW-Authenticate**: Cabeçalho da resposta que exige o envio de dados de autenticação
- **Realm**: Definição do espaço protegido pela autenticação
- **Authorization**: cabeçalho da requisição que leva os dados de autenticação do cliente.
- **Base64**: algoritmo de codificação de dados para a Internet.

Autenticação HTTP – Digest

Cliente e servidor não trocam informações de senha, apenas o **hash**.

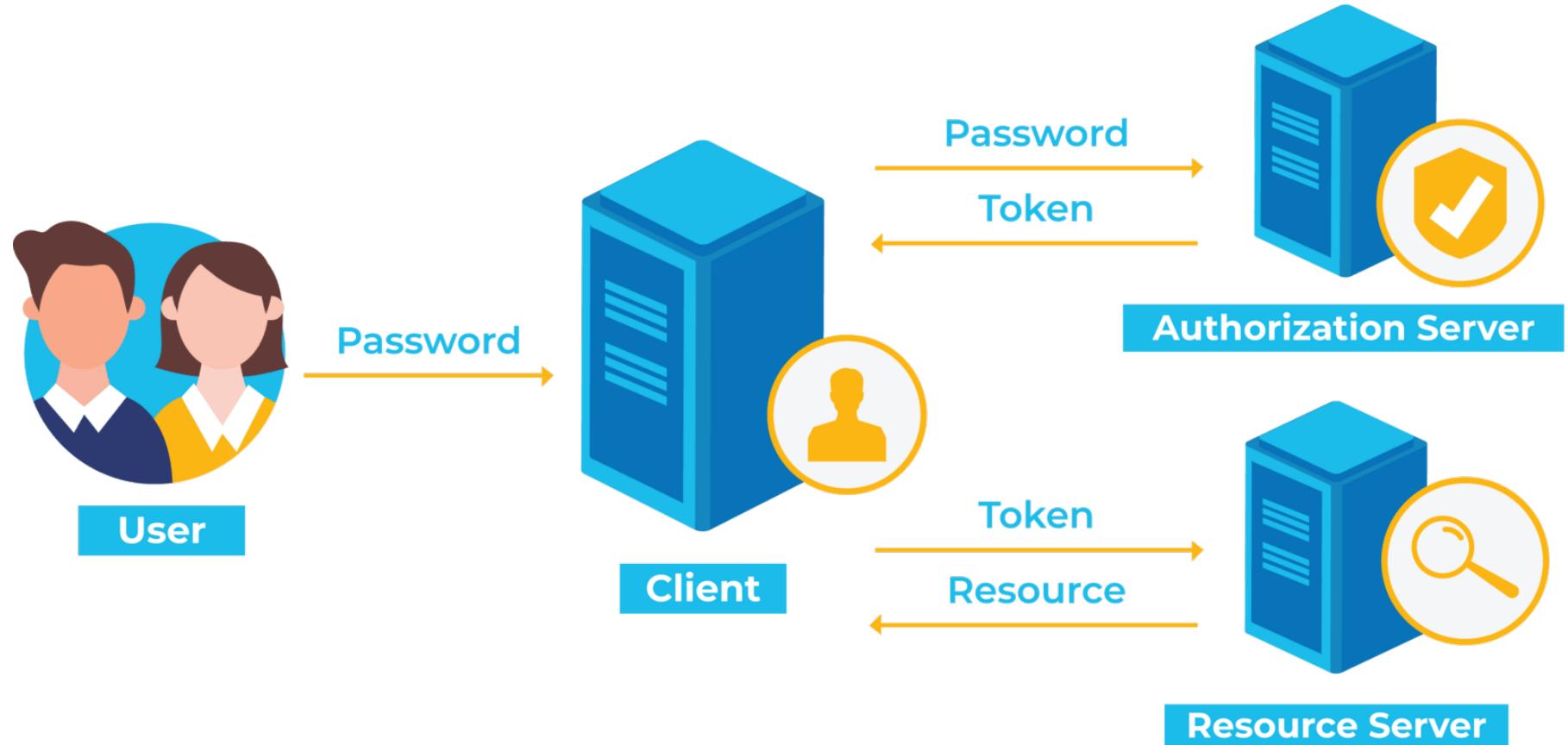


186



Autenticação HTTP – Bearer

Token Authentication



Fonte: [What Is Token-Based Authentication?](#) (Okta)

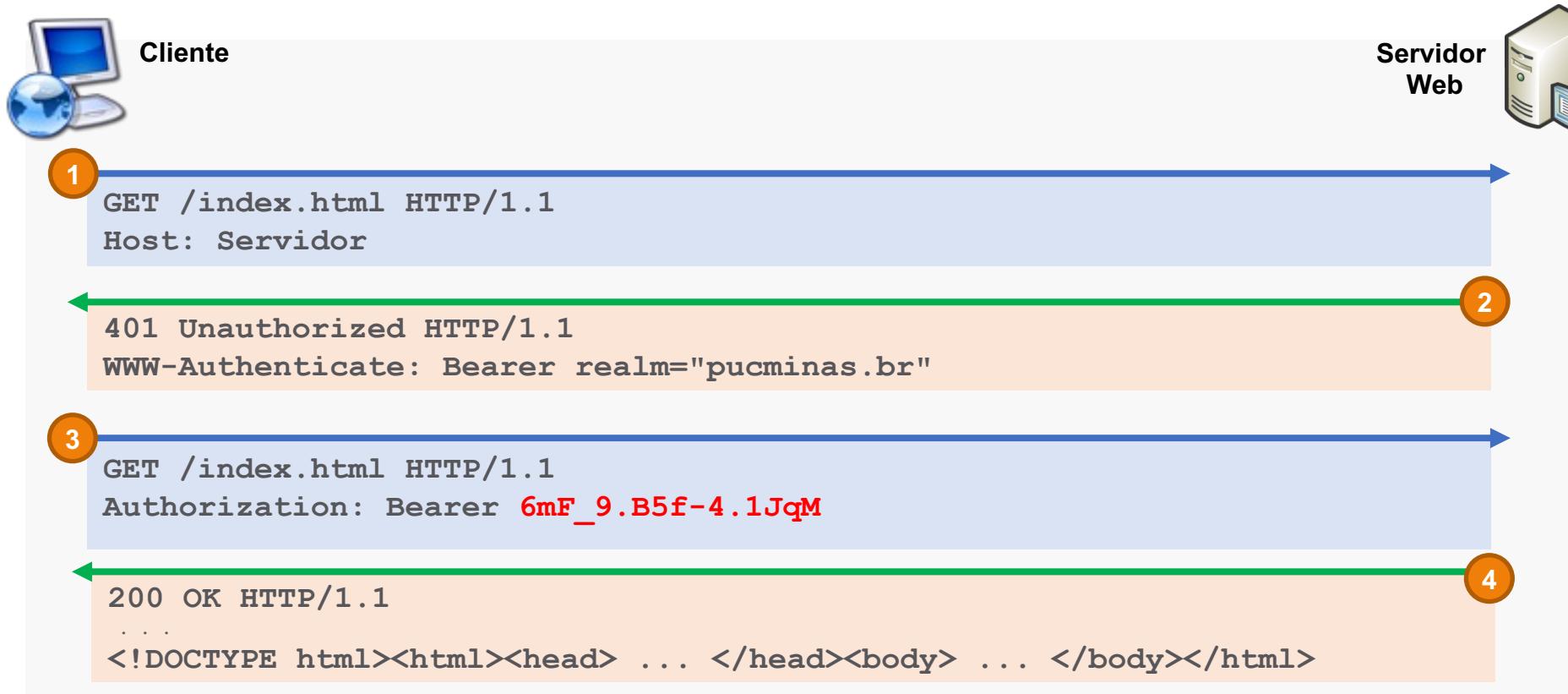
okta

Autenticação HTTP – Bearer

Token Authentication

Cliente e servidor trocam uma token previamente acordada.

IMPORTANTE: Recomenda-se utilizar apenas com conexões HTTPS.

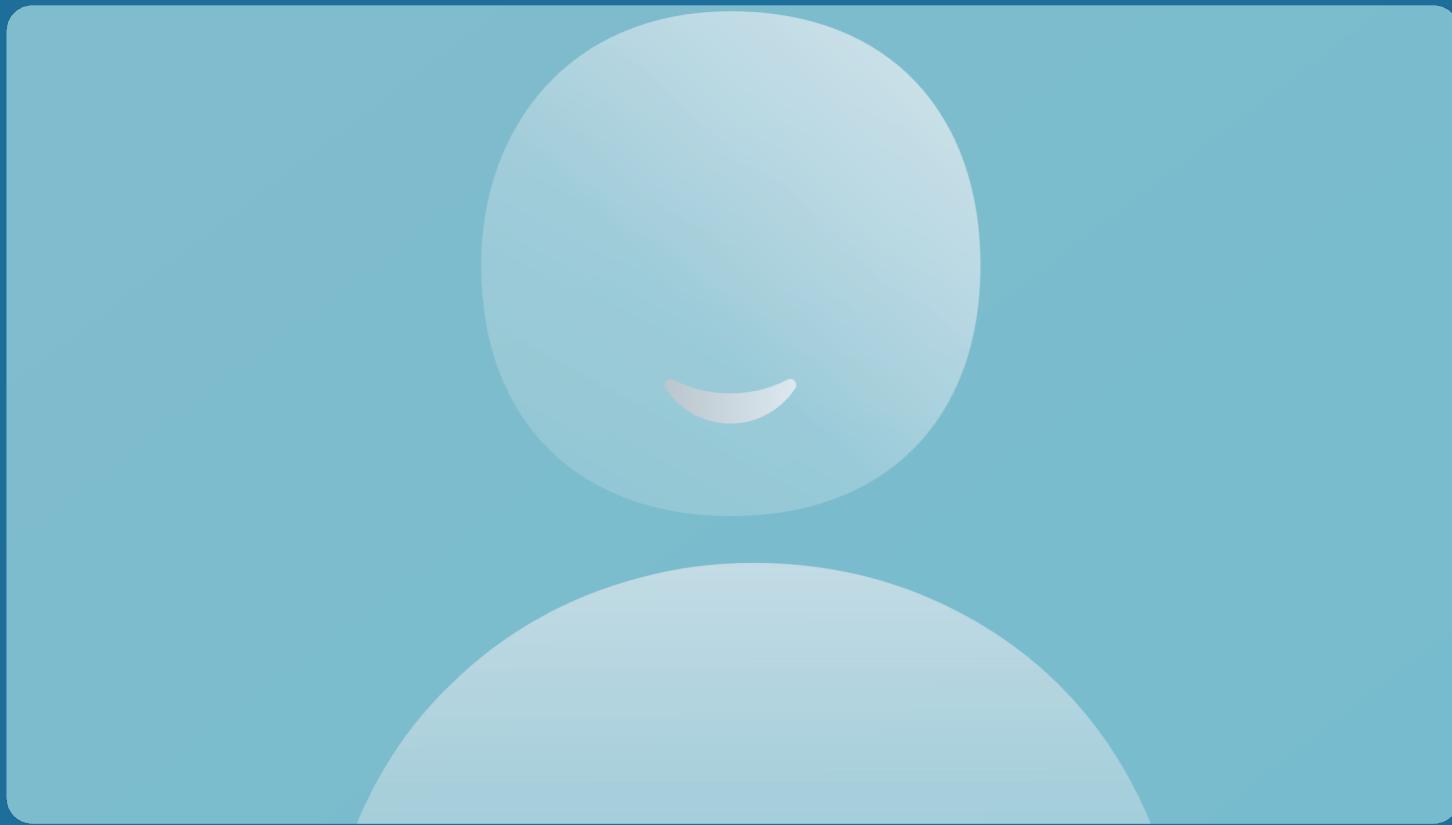


188



Prof. Rommel Vieira Carneiro

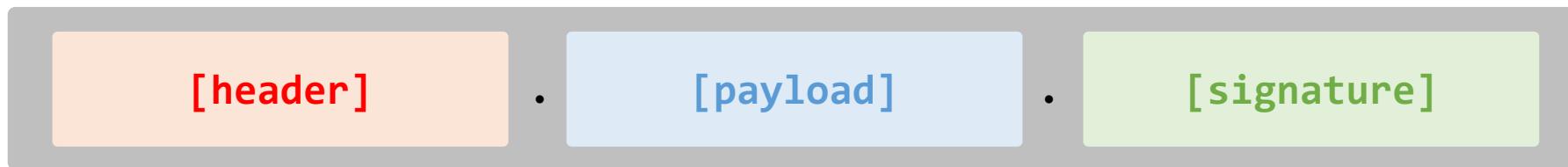
JSON Web Token (JWT)



JSON Web Token (JWT)

Padrão que define uma forma compacta e segura de transferência de informações baseada em tokens no formato JSON ([RFC-7519](#)), utilizado em mecanismos de autenticação e autorização.

Estrutura básica da token JWT



[header]

Informações sobre o tipo da token e algoritmo de criptografia

```
base64enc ({  
  "alg": "HS256",  
  "typ": "JWT"  
})
```

[payload]

Dados (claims) que podem ser reservados, públicos ou privados

```
base64enc ({  
  "id": "HS256",  
  "roles": "ADMIN"  
})
```

[signature]

Assinatura da token que evita alterações

```
HMACSHA256 (   
  base64enc(header) + "." +  
  base64enc(payload),  
  secretKey  
)
```

JSON Web Token (JWT)

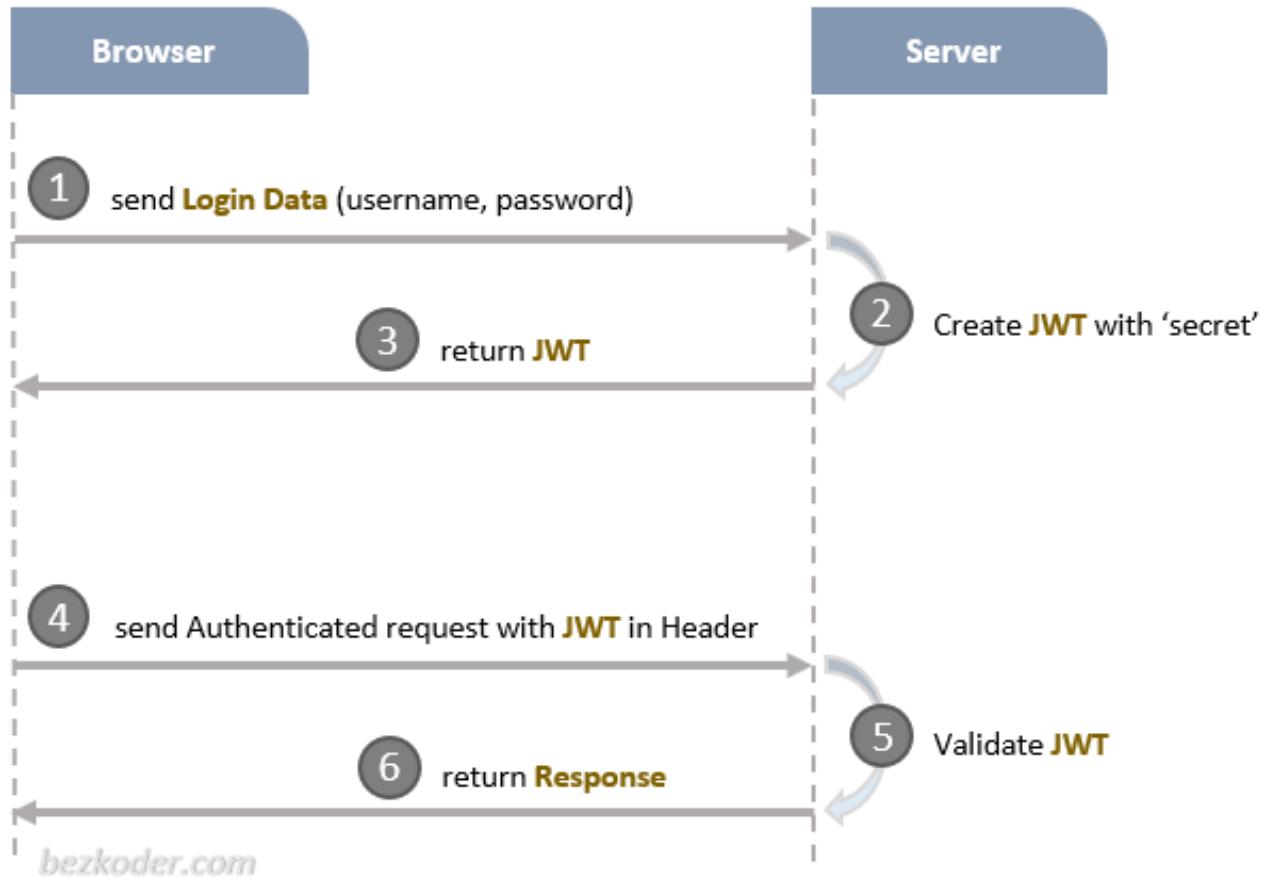
Campos reservados (claims)

- Iss (Issuer) – Emissor da token JWT
- Sub (Subject) – Assunto da token
- Aud (Audience) – Identifica o público para o qual a token foi emitida
- Exp (Expiration Time) – Horário de expiração da token, a partir do qual não é mais aceita
- Iat (Issued At) – Horário em que a token foi emitida



Fonte: [Wikipedia JSON Web Token](#)

JSON Web Token (JWT)



Fonte: [Node.js Express: JWT example | Token Based Authentication & Authorization](https://www.bezkoder.com/node-js-express-jwt-example/)

JSON Web Token (JWT)

The screenshot shows the jwt.io interface. On the left, under 'Encoded', there is a large text area containing a long, encoded JWT string. On the right, under 'Decoded', the token is broken down into its components:

- HEADER: ALGORITHM & TOKEN TYPE**

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```
- PAYOUT: DATA**

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```
- VERIFY SIGNATURE**

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret base64 encoded
```

At the bottom left, there is a green checkmark icon followed by the text 'Signature Verified'. At the bottom center, there is a blue button labeled 'SHARE JWT'.

JWT.io
Testes com tokens JWT

JSON Web Token (JWT)



Gerando uma chave forte em JavaScript com Node.js

```
>> node -e "console.log(require('crypto').randomBytes(256).toString('base64'));"  
  
//Exemplo de chave secreta gerada  
GdLOU8owaDRw3kF23gicjzWHLsICP5pP365G/8Jd1Vm90qaphvQFRJAtZOFU2BuAj+y0bat3ImenitbJp  
ZF+9toj9+4L7A9XZW98kxLo+oGJhXJwlwc78bvJaHH7zzx1VFBZ9ipGoIz57xqlhqZE7buH1nPjRGOSH  
QuqZ7XxZ1/0dk7Veib182Sbox3G8gCI38FIBmiJM4quWDv3MZOIP1ElQlWgVBJH/y6AxZkxSMan8DyZ9x  
a94pKxwXFMQPZrbD0+rhnEC0rfi5kvCxVLM/w3rB3EHWBHNNaQ941QKpIMjqAg8iIlLUegfZ4tJs49tGV  
fYgwtBKAKXwI6YuCIw==
```

JSON Web Token (JWT)

```
require('dotenv').config()
const JWT = require('jsonwebtoken')
const express = require("express")
const app = express()

app.get("/", function(req, res){
  JWT.sign({ userID: 'abc123' },
    process.env.SECRET_KEY,
    { algorithm: 'HS256' },
    (err, token) => {
      if (err) console.error('Erro: ' + err);
      else console.log (token)
    })
  res.send ('Hello JWT')
});

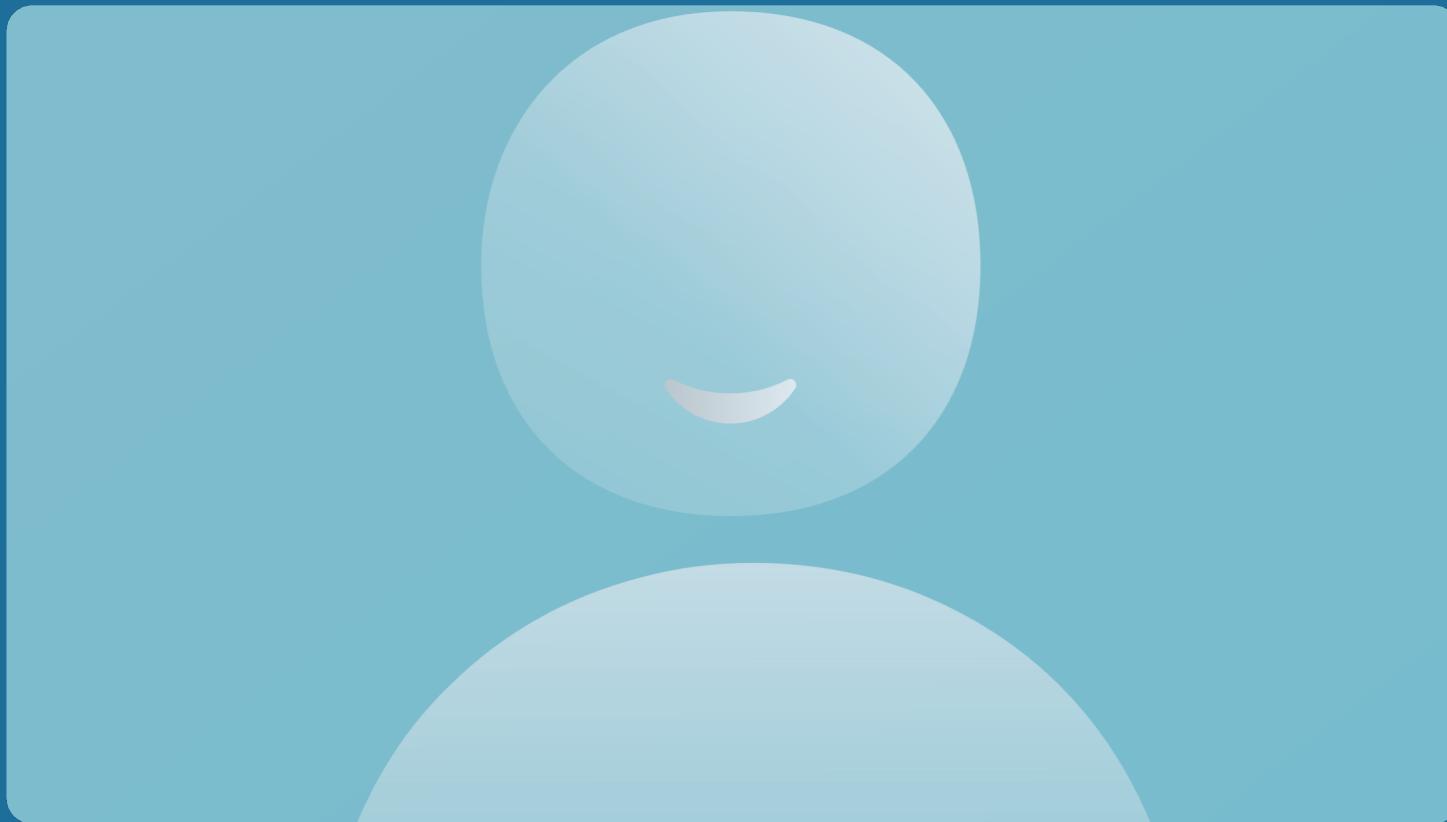
app.listen(3000);
```

Criação de token com informação do usuário (id)

Instalação JSONWebToken
npm install jsonwebtoken



Open Authorization (OAuth)



OAuth - Open Authorization



Framework aberto definido pelo IETF (RFC 6749) com foco na autenticação e autorização de recursos na Web.

Características

- Evita exposição de senhas
- Baseado no uso de tokens e Autenticação Bearer
- Facilita a interoperabilidade (Web, Mobile, Server)
- Controla a validade e o escopo do acesso concedido



OAuth - Papéis



**Dono do
Recurso**



**Aplicação
Cliente**



Papéis no OAuth 2.0

O mecanismo de autorização e autenticação
OAuth define quatro papéis.



**Servidor de
Autorização**



**Servidor de
Recursos**



OAuth - Papéis



Dono do Recurso

Entidade que possui recursos na rede e pode ser solicitado a autorizar o acesso a estes recursos protegidos. **Ex: Usuário final**



Aplicação Cliente

Aplicações utilizadas para acessar recursos na rede. Podem ser confidenciais ou públicos. **Ex: aplicações móveis e sites na Web**



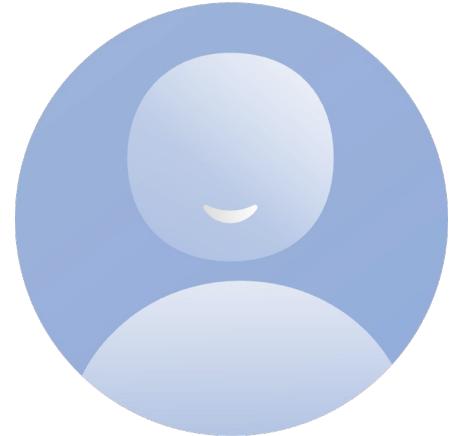
Servidor de Autorização

Sistema que controla a geração de tokens de acesso para as aplicações cliente. **Ex: Google Accounts**



Servidor de Recursos

Sistema que mantém recursos na Internet e pode fornecer tais recursos por autorização do dono. **Ex: Google Photos**



OAuth - Fluxo Geral



- 1 - requisita autorização
- 2 - autoriza acesso
- 3 - requisita token
- 4 - envia token
- 5 - encaminha token
- 6 - envia recurso



Dono do Recurso



Servidor de Autorização



Servidor de Recursos



OAuth – Access Token

- O Access Token é uma **credencial para acesso** a um recurso protegido.
- Trata-se de uma **string em formato específico** de acordo com a aplicação em questão
- Uma Access Token é obtida de acordo com o **tipo de autorização**
- A Access Token substitui a necessidade de **usuário e senha**



OAuth - Tipos de Autorização



1

Código de Autorização (*Authorization Code*)

Aplicação Cliente é uma aplicação Web ou nativa e mantém uma chave secreta.
Ex: Site X acessa seus dados no facebook

2

Autorização Implícita (*Implicit Grant*)

Aplicação Cliente é baseada no browser e não pode manter uma chave secreta.
Ex: Aplicações SPA (Single Page Web)

3

Credenciais do Usuário (*Resource Owner Password Credentials*)

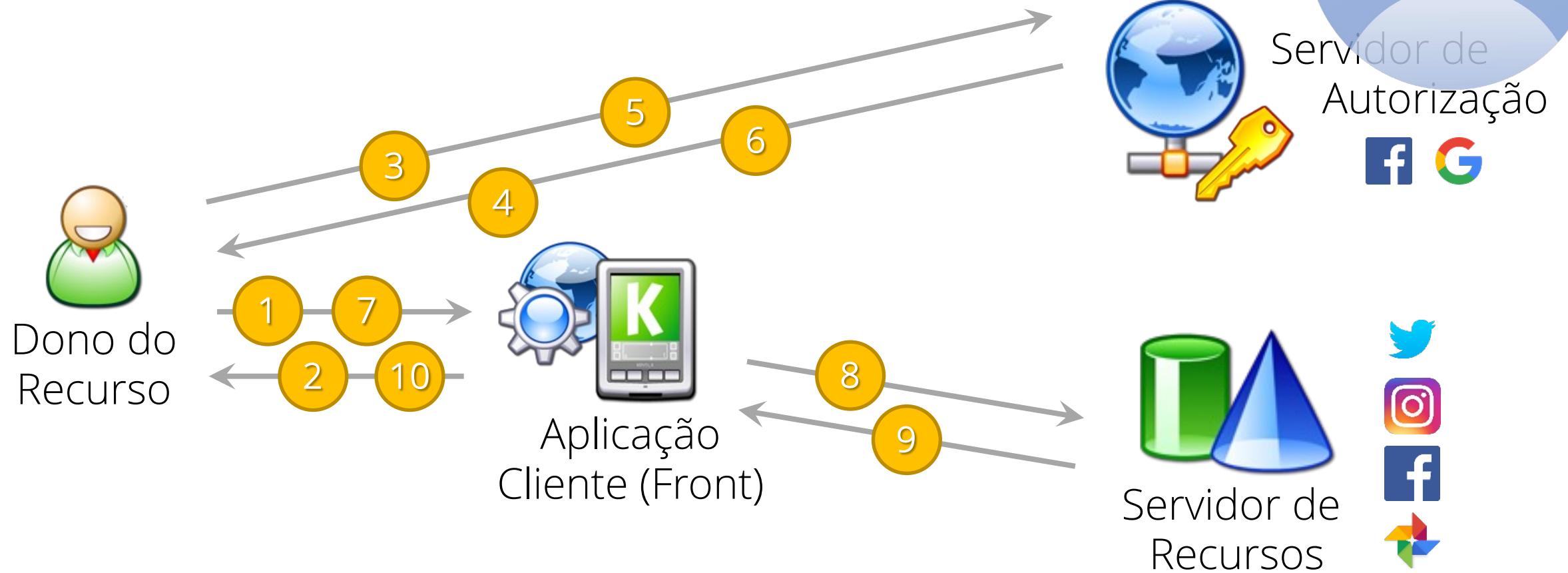
Aplicação Cliente é próxima do Servidor de Autorização e requer usuário e senha, normalmente, ambos feitos pela mesma empresa. Ex: Aplicativo "Gerenciador de Negócios" do facebook

4

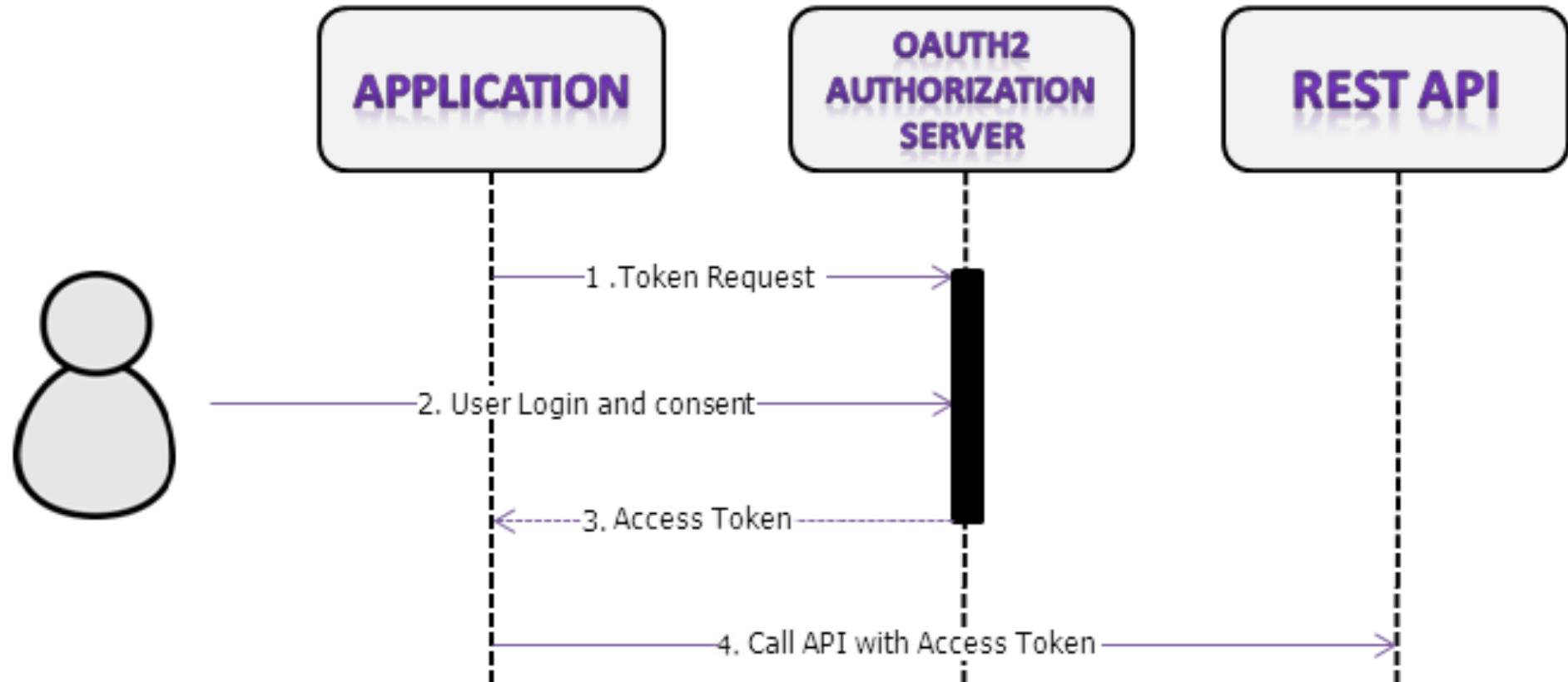
Credenciais do Cliente (*Client Credentials*)

Aplicação Cliente é a proprietária dos recursos e não o usuário final. Ex: Cloud Azure acessando dados em storage interno

OAuth - Fluxos - Autorização Implícita

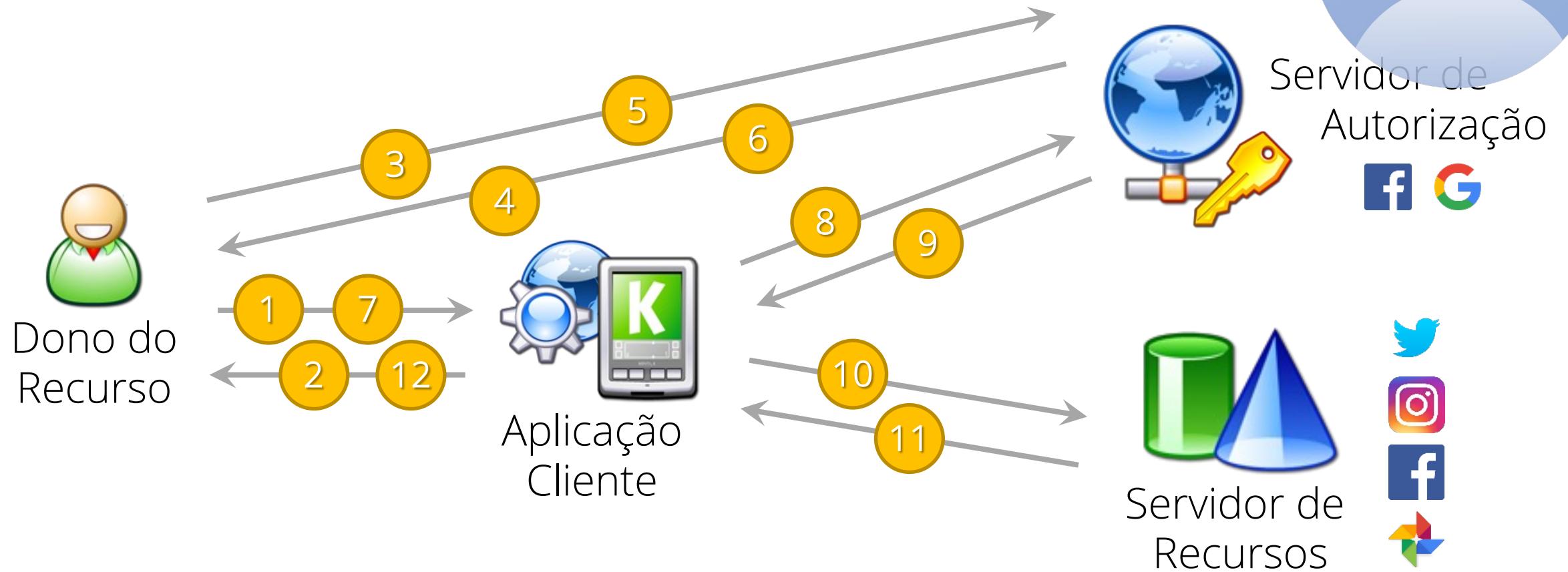


OAuth - Fluxos - Autorização Implícita

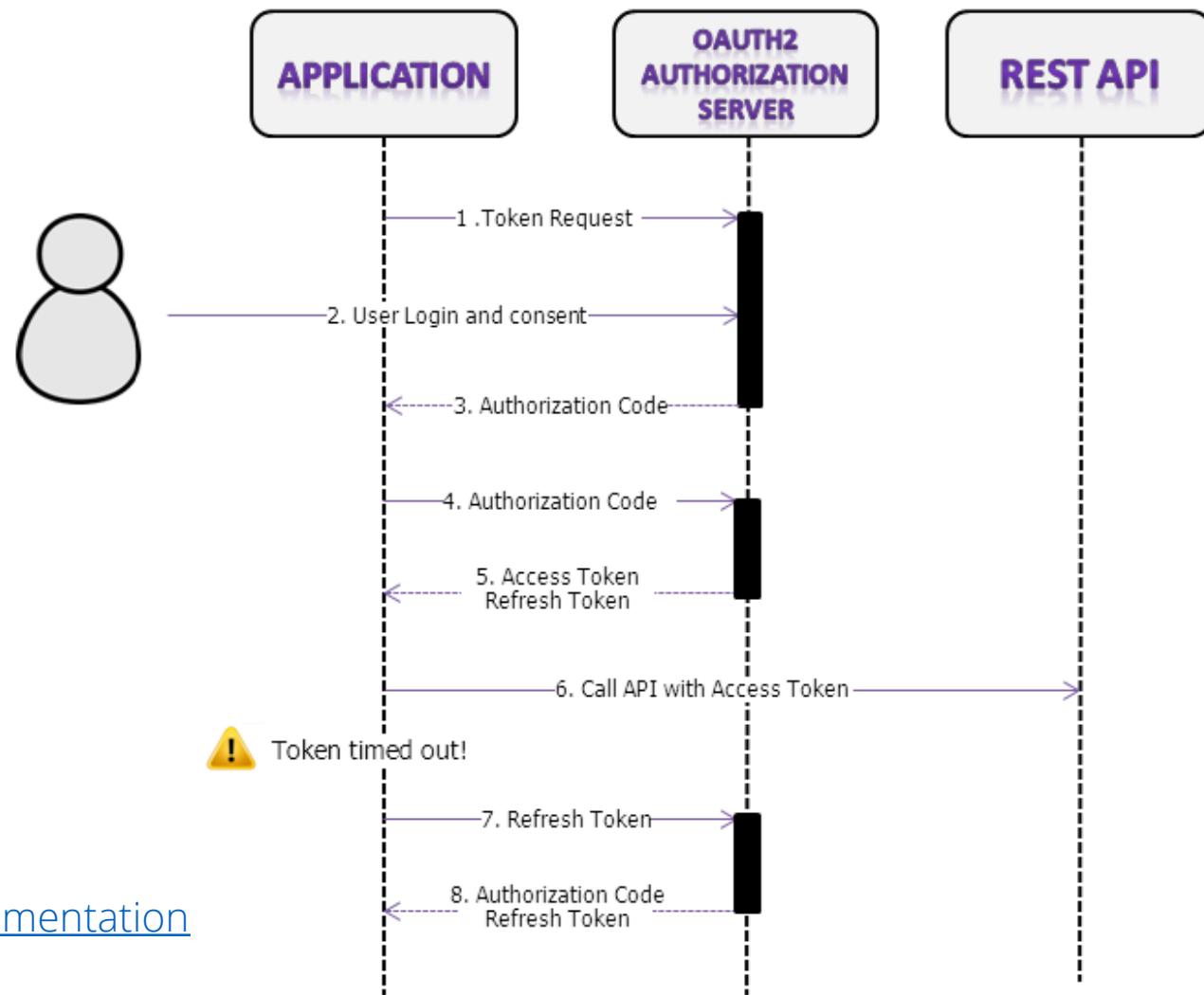


Fonte: [Qeo Native Documentation](#)

OAuth - Fluxos - Código de Autorização

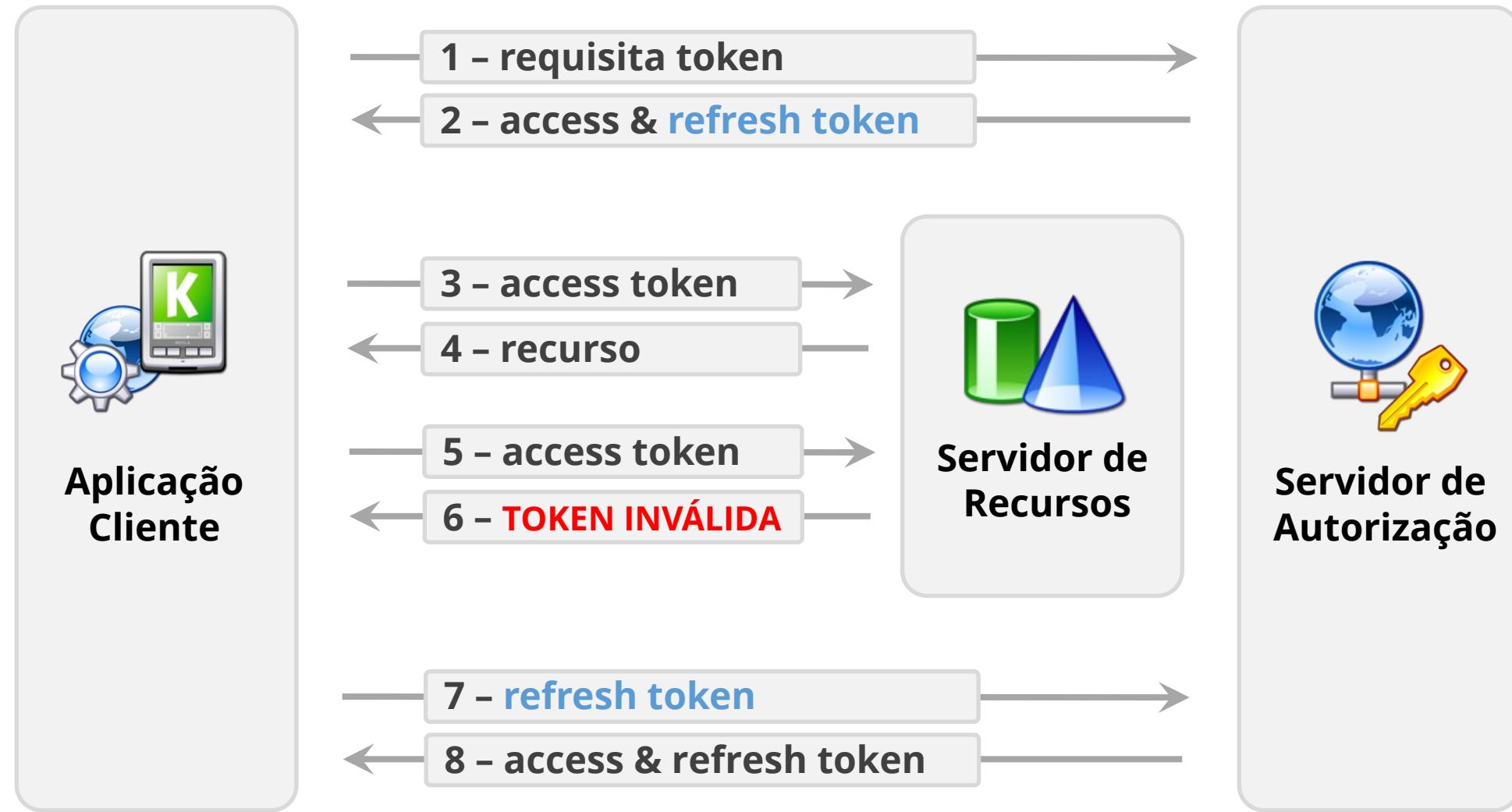


OAuth - Fluxos - Código de Autorização

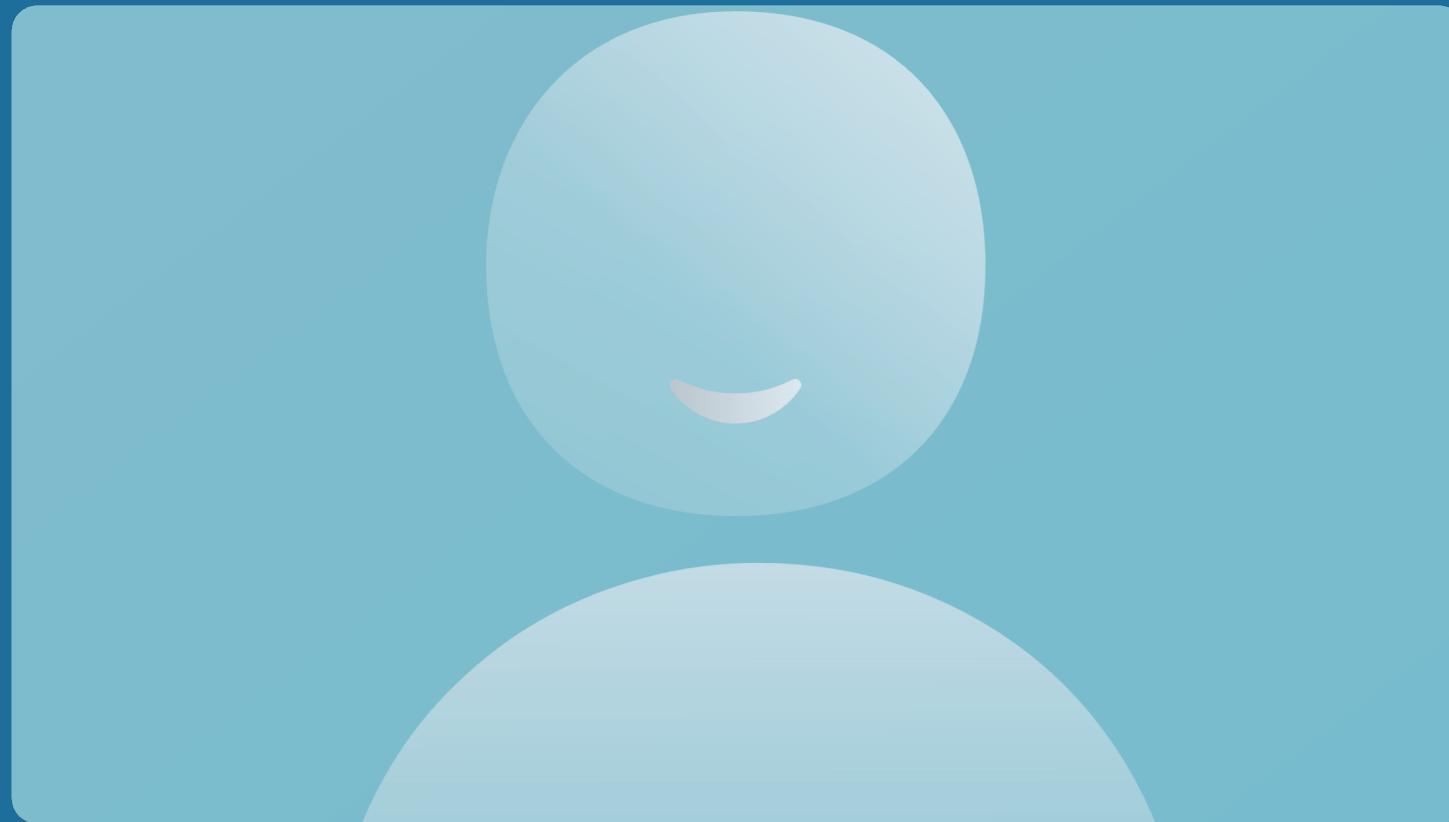


Fonte: [Qeo Native Documentation](#)

OAuth - Refresh token



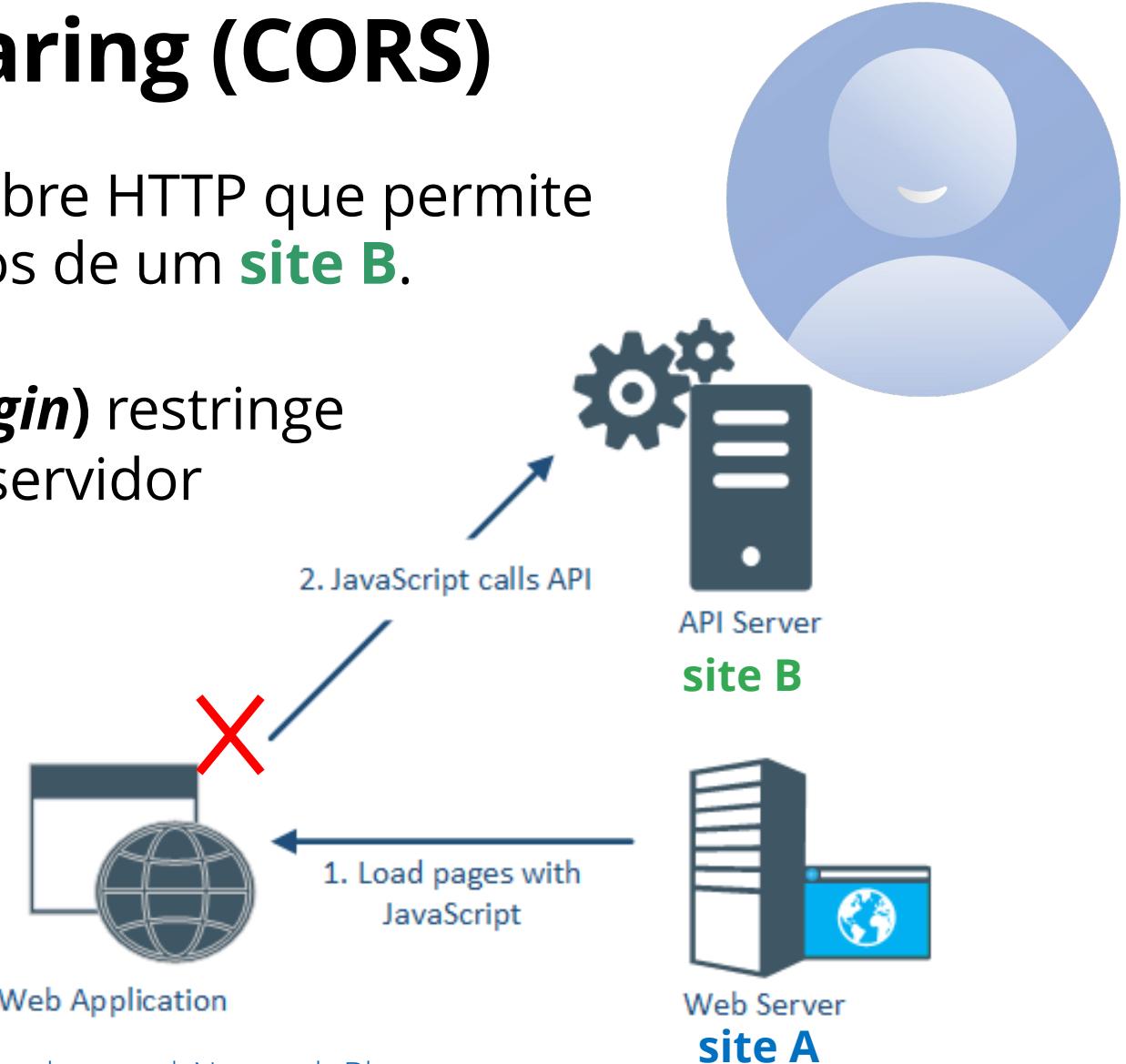
Cross Origin Resource Sharing (CORS)



Cross-Origin Resource Sharing (CORS)

CORS é um mecanismo de segurança sobre HTTP que permite páginas de um **site A** acessarem recursos de um **site B**.

A **política de mesma origem (same-origin)** restringe sites de acessarem recursos apenas no servidor em que está hospedado, evitando que códigos maliciosos usem chamadas AJAX para repasse de informações em uma sessão aberta.



Fontes:

- [Fetch Standard](#)
- [How to enable your API with flexible support for CORS with no code changes | Nevatech Blog](#)

Cross-Origin Resource Sharing (CORS)

O CORS define dois tipos de requisições:

Requisição Simples

Métodos: GET, HEAD e POST

Cabeçalhos permitidos:

- Accept
- Accept-Language
- Content-Language
- Content-Type (*)
- Connection
- User-Agent

(*) Apenas para valores como application/x-form-urlencoded, multipart/form-data, text/plain)

➤ Requisições permitidas

Requisição Preflighted

Métodos: GET, HEAD, POST, PUT, DELETE

Cabeçalhos permitidos:

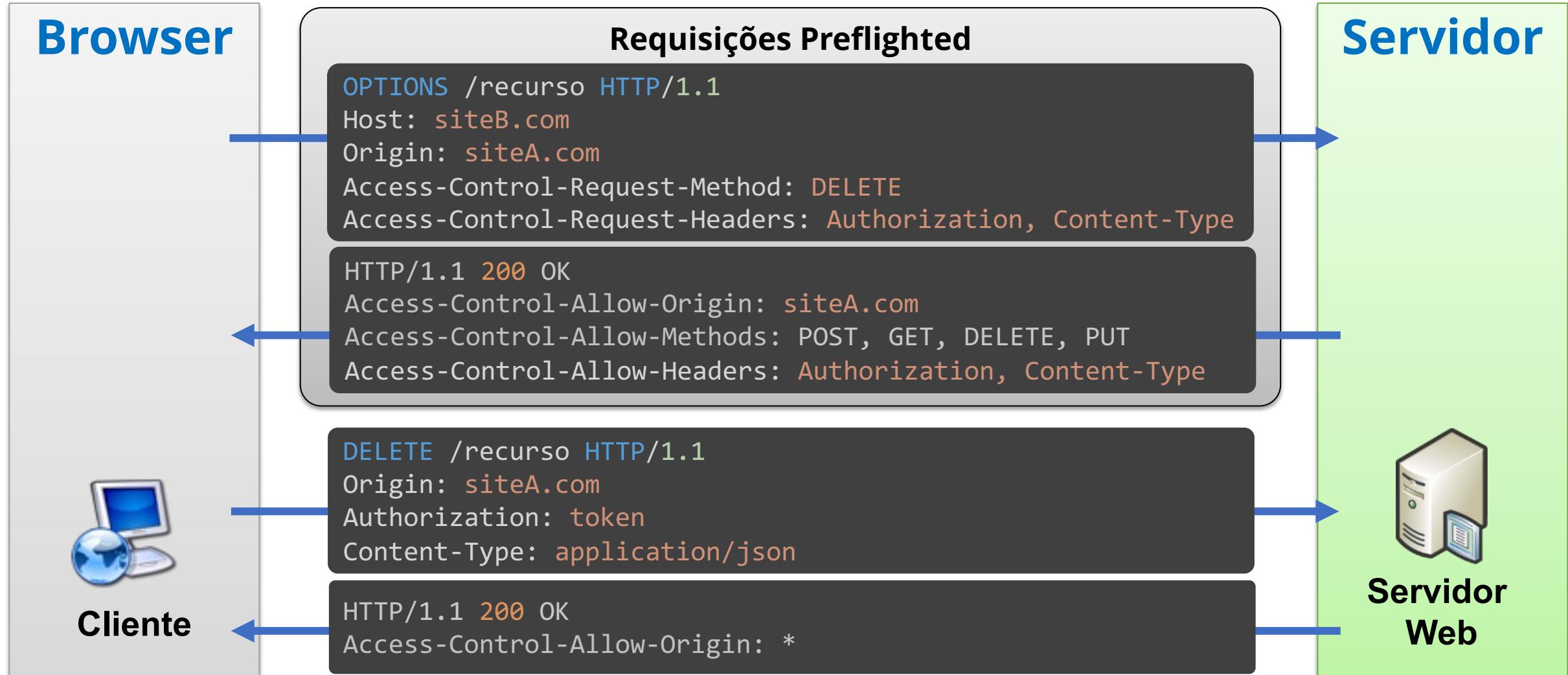
- Demais cabeçalhos

➤ Requer verificação de acesso via método OPTIONS

Cross-Origin Resource Sharing (CORS) - Simples



Cross-Origin Resource Sharing (CORS) - Preflighted



Cross-Origin Resource Sharing (CORS)

É possível desabilitar o controle de ***same-origin*** feito pelo browser enquanto estiver fazendo testes de desenvolvimento.

Para isso, localize uma extensão, ou utilize configurações específicas do browser.

Execute o Chrome com o seguinte parâmetro

```
chrome --disable-web-security --user-data-dir
```



Abra a página **about:config**

Localize e altere a opção **security.fileuri.strict_origin_policy** para false



Configuração do CORS em APIs com Node.js

Módulo cors

Trata a negociação HTTP para chamadas cross-origin.

Funcionalidades

- Habilita CORS no servidor todo ou em rotas específicas
- Altamente configurável

Instalação

```
npm install cors
```

Fonte: [CORS NPM Package](#)

```
const express = require('express');
const cors = require('cors');
const app = express();

// Configuração do CORS para permitir site específico
const corsOptions = {
  origin: 'https://www.exemplo.com/client',
  methods: 'GET,PUT,POST,DELETE',
};

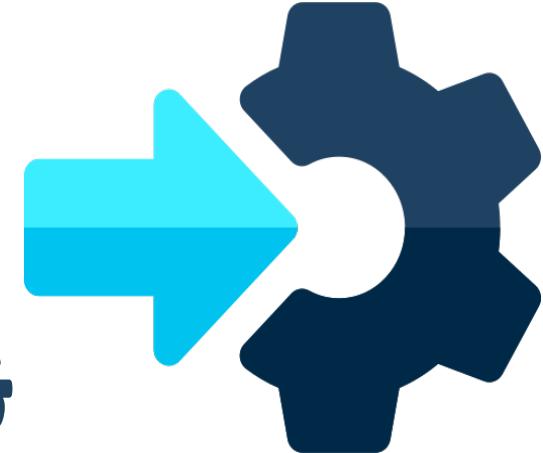
// Aplicando as configurações CORS
app.use(cors(corsOptions));

// Rotas de exemplo para as operações CRUD
app.get('/recurso', (req, res) => {});
app.post('/recursote', (req, res) => {});
app.put('/recursote', (req, res) => {});
app.delete('/recursote', (req, res) => {});

app.listen(3000, () => { console.log('Server ok'); });
```

Obrigado!

APIs &
Web Services



Prof. Rommel Vieira Carneiro

in [/rommelcarneiro](https://www.linkedin.com/in/rommelcarneiro)



<http://rommelcarneiro.me/>