



# Segurança Autenticação

---

Prof. Victor Farias

V 1.3

# Introdução

---

# Introdução

- Nosso sistema precisa de um sistema de autenticação
- No sistema de matrícula, o aluno se loga usando matrícula e senha
- Para acessar os endpoints do sistema, o usuário deve estar logado
- Vamos elencar alguns de problemas de autenticação e segurança

# Problema 1

## Comunicação

---

# Problema 1 - Comunicação

- Como receber a senha?
  - Vamos receber a senha via endpoint POST
  - Se tiver alguém na linha **man-in-the-middle?**
  - Informação é enviada em texto plano no HTTP
- Qualquer um pode escutar pode escutar a rede
  - Sniffers!

# Possível Solução

- Protocolo **HTTPS**

- Comunicação criptografada
- Criptografia assimétrica
- Protocolos SSL/TLS

- Em breve, chrome vai marcar todas as páginas HTTP como inseguras
- Algoritmo de busca do google está dando prioridade no SEO para páginas HTTPS
- Não impede de obter metadados - sites acessados, tempos de acesso, nome de arquivos...

# Problema 2

## Armazenando Senhas

---

## Problema 2 - Armazenando Senhas

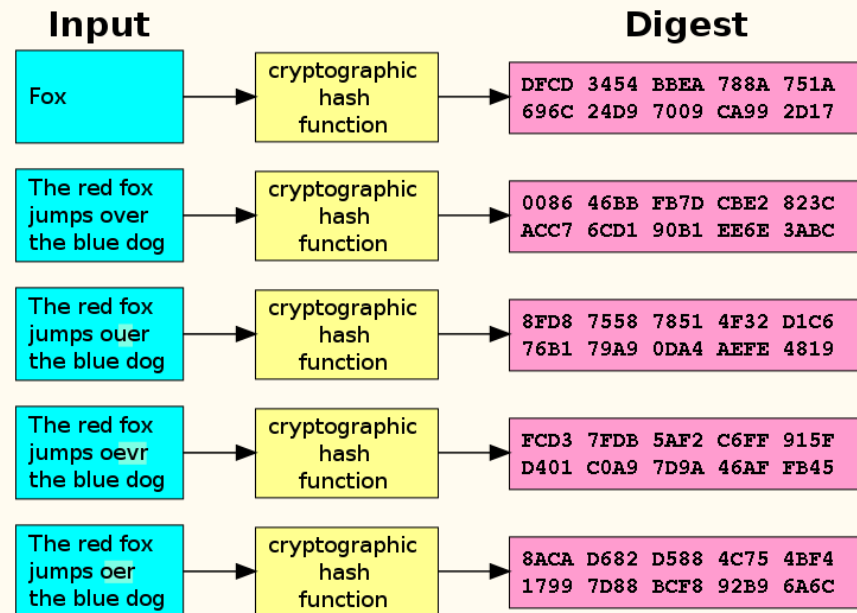
- Como guardar a senha em banco?
- Se guardarmos a limpa ...
  - Algum funcionário mal-intencionado pode olhar as senhas dos clientes no banco
  - Se alguém conseguir invadir o banco, o invasor terá todas as senhas facilmente



# Solução

## ● Funções HASH Criptográficas - MD5, SHA-1, SHA-3, Bcrypt ...

- Função hash é um função que recebe dados de tamanho variável e retorna um dado de tamanho fixo
- Esse retorno é uma cadeia de caracteres que chamamos de **assinatura hash**

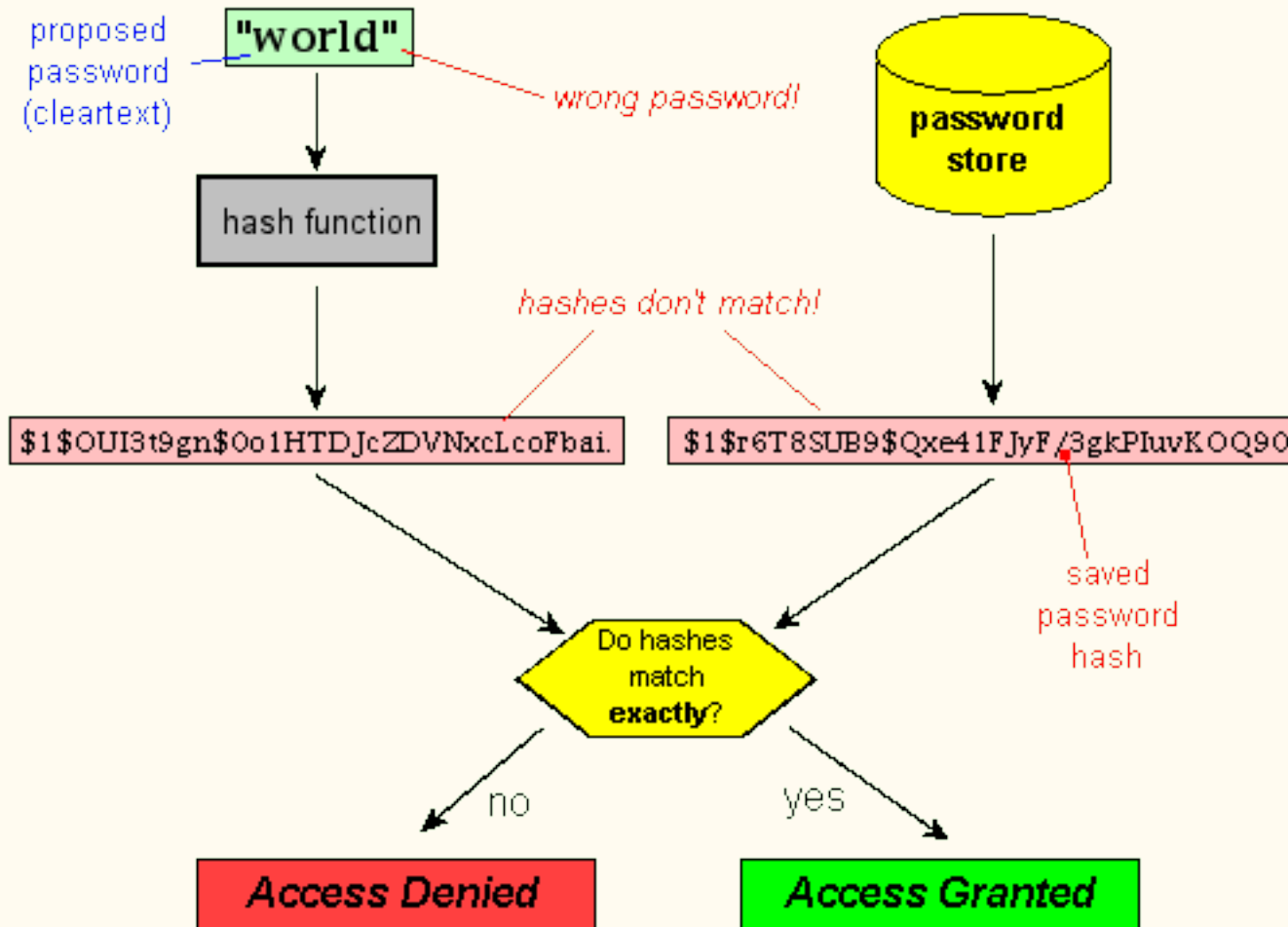


Fonte:  
[https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function)

# Hash

- Propriedade importante das funções **hash**
  - A partir da assinatura, não é possível obter o dado original
  - Ao produzir a assinatura, se perde informação
- Assim, não guardaremos a senha em banco
  - Guardamos apenas a assinatura hash da senha
- Desse modo, mesmo que um invasor tenha posse das assinaturas hash, não é possível obter a senha original
- **Importante:** Não usar **MD5** e **SHA-1**
  - Eles já foram quebrados
  - Já é possível obter as senhas a partir da assinatura hash

# Fluxo - Backend



# BCrypt no Node

- Usaremos o **BCrypt** para hashear nossas senhas
- Instalação:

```
npm install --save bcrypt
```

# Como usar

## ● Para criar hash

- Função **bcrypt.hashSync**(data, salt)
- **data** é o dado a ser hasheado
- **salt** representa um inteiro usado para criar uma string que será concatenada com o dado (valor 10, por exemplo)
- Retorna Hash

## ● Para comparar dois hashes


- Função **bcrypt.compareSync**(hash1, hash2)
- Retorna **true** caso sejam iguais ou **false**, caso contrário

# Refatorando Schema de Aluno

```
// app/models/aluno.js
...
  senha: {
    type: String,
    required: true
  }
...
```

# Inserindo Usuário com Senha no Banco

```
// app/controllers/alunos.js
module.exports.adicionarAluno = function(req, res){
  let aluno = new Aluno({
    nome: req.body.nome,
    matricula: req.body.matricula,
    senha: bcrypt.hashSync (req.body.senha, 10)
  });
  let promise = Aluno.create(aluno);
  promise.then(
    function(aluno) {
      res.status(201).json({nome:aluno.nome, matricula:aluno.matricula});
    },
    function(erro){
      res.status(500).json(erro);
    }
  );
};
```

- Tomar cuidado para não enviar senha de volta.
  - Modificar também os outros arquivos.
- 

# Rota - Adicionando Verificação de Credenciais

```
// app/route/alunos.js
const controller = require('../controllers/alunos.js');
const auth = require('../controllers/auth.js')

module.exports = function(app){
  app.post('/api/alunos/signin', auth.logar)
    .post('/api/alunos', controller.inserirAluno)
    .get('/api/alunos', controller.listaAlunos)
    .get('/api/alunos/:id', controller.obterAluno)
    .get('/api/alunos/:id/matriculas', controller.obterMatriculasDeAluno);
}
```



# Controller -Adicionando Verificação de Credenciais

```
// app/controllers/auth.js
const bcrypt = require('bcrypt');
const Aluno = require('../models/alunos.js');
module.exports.logar = function(req, res){
  function logar(user){
    if(!bcrypt.compareSync(req.body.senha, user.senha)){
      falhar();
    }else{
      res.status(200).send("Credenciais estão OK!")
    }
  }
  function falhar(){
    res.status(401).send('Invalid login');
  }
  Aluno.findOne({matricula:req.body.matricula}).exec().then(logar, falhar);
}
```

# Problema 3

## Autenticação

---

# Autenticação

- Como liberar acesso aos recursos apenas para quem tem as credenciais válidas?
- Enviar credenciais em toda requisição?
  - Má ideia
- Solução: Autenticação via **Tokens!**

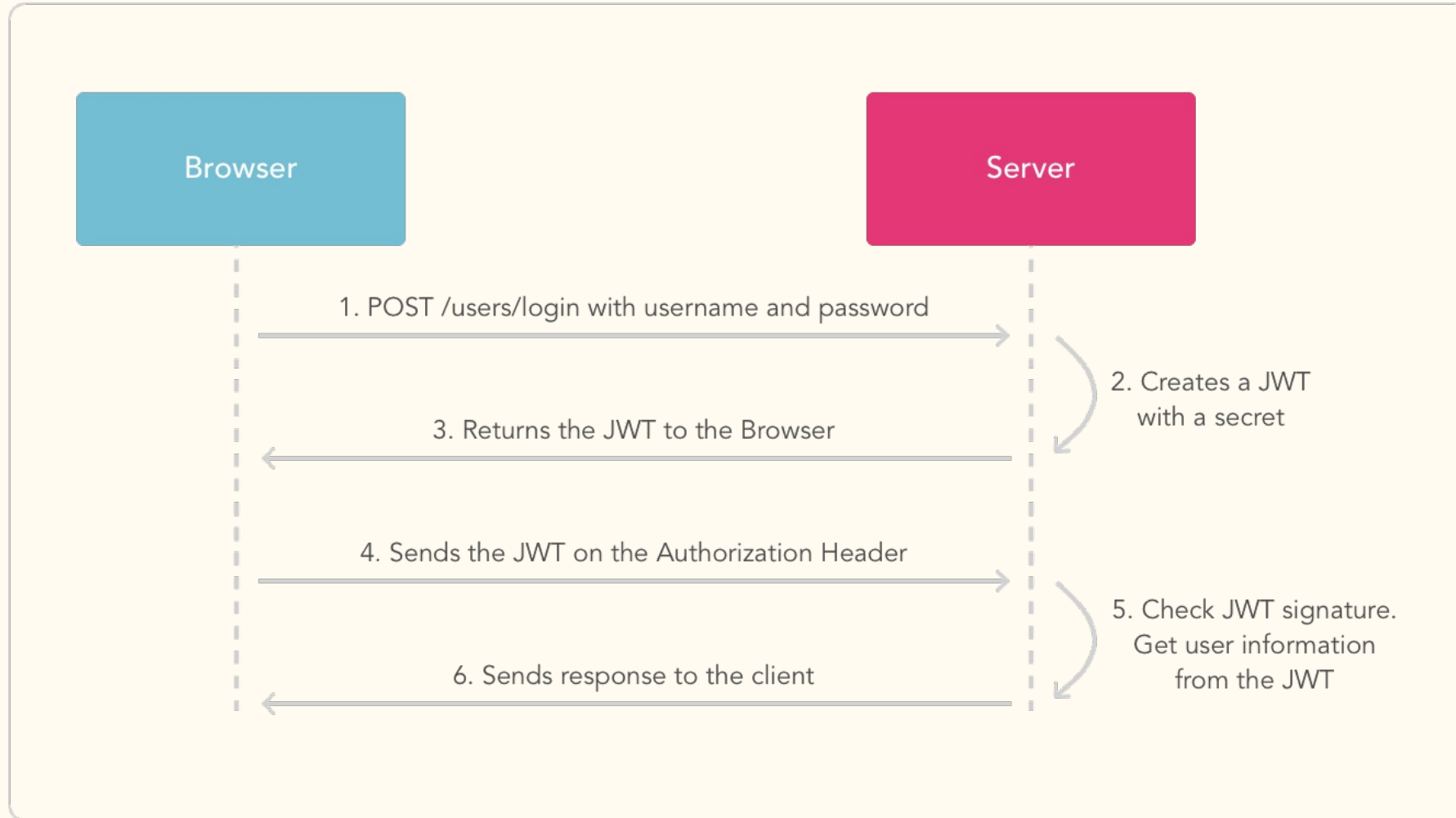
# Autenticação via Tokens

- Token serve para identificar uma aplicação
- Ao fazer o login, o servidor retorna um token para o cliente
  - Esse token contém um identificador da sessão, data de validade do token, id do usuário ...
- Sempre que formos acessar algum recurso no servidor, temos que passar também o token para mostrar que estamos logados
  - A partir do **token**, o servidor consegue saber qual é o usuário logado

# Json Web Token

- Json Web Token (JWT)
  - Padrão (RFC 7165)
  - Criação e transmissão segura de objetos JSON via token
- Um JWT é dividido em 3 partes
  - Header - informações como algoritmo de criptografia
  - Payload
  - Signature - informações para validar token
- No **payload**, é possível armazenar qualquer objeto
  - Inclusive dados do usuário

# Json Web Token - Fluxo



# Json Web Token - NodeJS

## **Instalação**

```
npm install --save jsonwebtoken
```

# Como usar

## ● Criar token

- Função **jwt.sign**(payload, secretOrPrivateKey)
- **payload** é os dados que vão ser embutidos no token
- **secretOrPrivateKey** é a chave/senha privada que só o servidor pode conhecer
- Retorna token

## ● Validar token

- Função **jwt.verify**(token, secretOrPublicKey)
- **Token** a ser validado
- **secretOrPublicKey** é a chave que foi usada para criar o token
- Retorna true se token é válido ou false, caso contrário



# Como usar

## ● Decodificar token

- Função `jwt.decode(token )`
- Recebe token a ser decodificado
- Retorna objeto representando payload
- obs: não valida token!

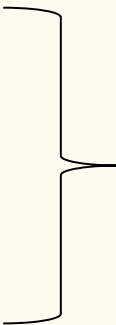
# Criando Token - Logar V2

```
// app/controllers/auth.js
module.exports.logar = function(req,res){
  function logar(user){
    if(!bcrypt.compareSync(req.body.senha, user.senha)){
      falhar();
    }else{
      let token = jwt.sign({user: user}, 'secret');
      res.status(200).json({
        message: "Logado",
        token: token,
        userId: user._id
      })
    }
  }
  function falhar(){
    res.status(401).send('Invalid login');
  }
  Aluno.findOne({matricula:req.body.matricula}).exec().then(logar, falhar);
}
```

# Verificando Token em todos endpoints - Rota

```
// app/route/alunos.js
const controller = require('../controllers/alunos.js');
const auth = require('../controllers/auth.js');
```

```
module.exports = function(app){
  app.post('/api/alunos/singin', auth.logar)
    .post('/api/alunos', controller.inserirAluno)
    .use('/api/alunos/', auth.checar)
    .get('/api/alunos', controller.listaAlunos)
    .get('/api/alunos/:id', controller.obterAluno)
    .get('/api/alunos/:id/matriculas', controller.obterMatriculasDeAluno);
}
```


- 
- O casamento das rotas com a rota pedida é feito sequencialmente.
  - Logo, a rota /api/alunos vai executar o controller auth.checar.
  - O mesmo vale para /api/alunos/:id onde é o controller auth.checar que é executado primeiro

Obs: Atenção à ordem das rotas!

# Verificando Token em todos endpoints - Controller

```
// app/controllers/auth.js
let jwt = require('jsonwebtoken');
module.exports.checar = function (req, res, next) {
  jwt.verify(req.headers.token, 'secret', function (err, decoded) {
    if (err) {
      return res.status(401).json({
        title: 'Not Authenticated',
        error: err
      });
    }
    next();
  })
};
```

Executa o próximo controller que casar com a rota pedida



# Perguntas?

Prof. Victor Farias