Cibersegurança e Defesa Cibernética

3° Entrega

Objetivo: Definir uma criptografia a ser implementada em código e justificar a escolha.

Optamos por utilizar a criptografia SHA-3 como algoritmo de hash em nossa aplicação. Essa decisão foi tomada com base em critérios de segurança, desempenho e robustez frente aos desafios atuais da cibersegurança.

SHA-3 é a mais recente versão da família de algoritmos SHA, padronizada pelo NIST (National Institute of Standards and Technology) em 2015. Diferente das versões anteriores (SHA-1 e SHA-2), que utilizam a arquitetura Merkle–Damgård, o SHA-3 é baseado em um novo tipo de construção chamada sponge function (função esponja), por meio do algoritmo Keccak, o que garante uma estrutura interna mais segura e resistente a ataques criptográficos modernos, como colisões, ataques de extensão de comprimento e ataques por pré-imagem.

Além disso, SHA-3 não substitui SHA-2, mas o complementa. Ele é uma alternativa segura e eficiente, principalmente em aplicações onde se deseja diversidade de algoritmos ou onde SHA-2 já não é considerado suficiente, como em sistemas críticos, autenticação de dados sensíveis e registros imutáveis.

No nosso projeto, o SHA-3 é usado para gerar um hash único e irreversível das coordenadas geográficas de origem e destino. Essa abordagem é útil para garantir a integridade dos dados e impedir modificações ou falsificações sem a necessidade de armazenar os dados originais em texto claro.

Justificativas técnicas:

- Alta resistência a colisões e outros ataques conhecidos;
- Padrão moderno e seguro, reconhecido internacionalmente;
- Independente de SHA-2, ideal para aplicações com alta exigência de segurança;
- Flexibilidade no tamanho de saída (SHA3-224, SHA3-256, SHA3-384, SHA3-512) – utilizamos SHA3-256 por ser o mais comum;
- Implementação eficiente com suporte em várias bibliotecas modernas como CryptoJS.

src > Components > InputForm.jsx > InputForm

```
29  function InputForm({ onCompare }) {
117   const handleOriginSelect = (suggestion) => {
118     const coords = `${suggestion.lat},${suggestion.lon}`;
119     const hash = CryptoJS.SHA3(coords, { outputLength: 256 }).toString();
120
121     const dadosOrigem = {
122       tipo: 'origem',
123       endereco: formatAddress(suggestion),
124       coordenadas: coords,
125       hash_sha3_256: hash,
126       Distancia: '',
127       price: '',
128       DiaSemana: new Date().getDay().toString(),
129     };
130
131     const encrypted = CryptoJS.AES.encrypt(JSON.stringify(dadosOrigem), 'secret-key').toString();
132
133     fetch('http://localhost:4000/save', {
134       method: 'POST',
135       headers: { 'Content-Type': 'application/json' },
136       body: JSON.stringify({ data: encrypted })
137     })
138       .then(response => {
139         if (!response.ok) throw new Error("Erro ao salvar no servidor");
140         return response.json();
141       })
142       .then(data => console.log("Origem salva com sucesso:", data))
143       .catch(err => {
144       });
145
146     setEnderecosJson((prev) => {
147       const updatedJson = { ...prev };
148       if (!updatedJson['criptografia']) {
149         updatedJson['criptografia'] = [];
150       }
151       updatedJson['criptografia'].push(encrypted);
152       return updatedJson;
153     });
154
155     setOrigin(formatAddress(suggestion));
156     setOriginSuggestions([]);
157   };
```

Função: Realiza a criptografia dos dados de origem antes de enviá-los para o servidor.

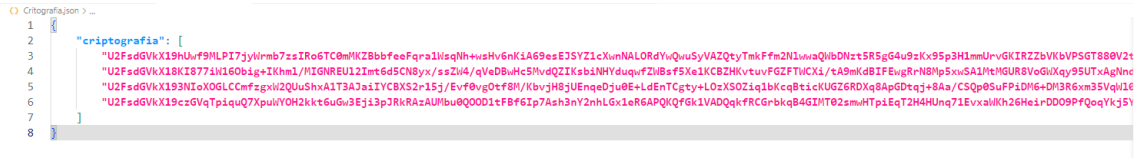
- A função handleOriginSelect é chamada quando uma sugestão de local é selecionada.
- O campo coords (coordenadas geográficas) é criptografado com SHA-3 (256 bits).
- O dado criptografado é enviado via POST para o servidor em http://localhost:4000/save.
- O resultado é salvo no estado React (setEnderecosJson) com o campo "criptografia" armazenando os textos criptografados.

server.js > app.post('/save') callback

```
const express = require('express');
const fs = require('fs');
const path = require('path');
const app = express();
const port = 4000;

app.use(express.json());

app.post('/save', (req, res) => {
  const { data } = req.body;

  if (data) {
    const filePath = path.join(__dirname, 'enderecos.json');

    // Garante que os dados sejam salvos como string JSON + quebra de linha
    const line = JSON.stringify(data) + '\n';

    fs.appendFile(filePath, line, (err) => {
      if (err) {
        console.error('Erro ao escrever no arquivo:', err);
        res.status(500).json({ error: 'Erro ao salvar dados' });
      } else {
        res.status(200).json({ message: 'Dados salvos com sucesso' });
      }
    });
  } else {
    res.status(400).json({ error: 'Dados não fornecidos' });
  }
});

app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${port}`);
});

```

Função: Recebe e armazena os dados criptografados enviados pelo front-end.

- Utiliza o framework Express.js para criar uma API simples em localhost:4000.
- A rota POST /save salva os dados criptografados no arquivo enderecos.json.
- Utiliza fs.appendFile() para adicionar cada nova entrada no arquivo local.
- Garante persistência dos dados para análise futura, mantendo a confidencialidade via criptografia.



Função: Armazena os textos criptografados pelo sistema.

- Contém um array chamado "criptografia" com os dados criptografados.
- Cada string do array representa uma informação sensível (como coordenadas, endereço, etc.) convertida para uma forma segura.
- Essas informações não podem ser interpretadas diretamente, garantindo confidencialidade dos dados.