

## 4. DESENVOLVIMENTO E IMPLEMENTAÇÃO DA SOLUÇÃO

### 4.1. Visão Geral da Arquitetura

---

A solução proposta, denominada *Secure Vault*, foi desenvolvida seguindo o paradigma de **Arquitetura Zero-Knowledge** (Conhecimento Zero). Diferente de sistemas tradicionais de armazenamento em nuvem, onde a segurança reside na confiança no provedor do serviço, esta aplicação transfere a responsabilidade criptográfica inteiramente para o lado do cliente (*Client-Side*).

O sistema foi estruturado em três camadas principais:

- **Interface do Cliente (Frontend):** Responsável pela interação com o usuário e, crucialmente, pela execução dos algoritmos criptográficos utilizando a CPU do dispositivo local.
- **Servidor de Aplicação (Backend):** Atua como uma API RESTful para gerenciamento de requisições, autenticação de hashes e persistência de dados cifrados.
- **Banco de Dados (Database):** Armazena os metadados e os *blobs* binários criptografados.



Syntax error in text  
mermaid version 10.9.5

Figura 1: Arquitetura do Sistema Secure Vault

### 4.2. Tecnologias e Ferramentas Utilizadas

---

Para a implementação do protótipo funcional, optou-se por uma *stack* moderna baseada em JavaScript, garantindo compatibilidade e performance:

- **Frontend:** Desenvolvido em **React.js** com a ferramenta de build **Vite**. A escolha deve-se à reatividade necessária para manipular estados de arquivos em memória sem recarregar a página.

- **Criptografia:** Utilizou-se a **Web Crypto API**, uma interface nativa dos navegadores modernos recomendada pela W3C. O uso de uma API nativa elimina a necessidade de bibliotecas de terceiros pesadas.
- **Backend:** Implementado em **Node.js** com framework **Express**.
- **Banco de Dados:** Utilizou-se o **MongoDB Atlas** (NoSQL).

## 4.3. Implementação da Criptografia e Segurança

---

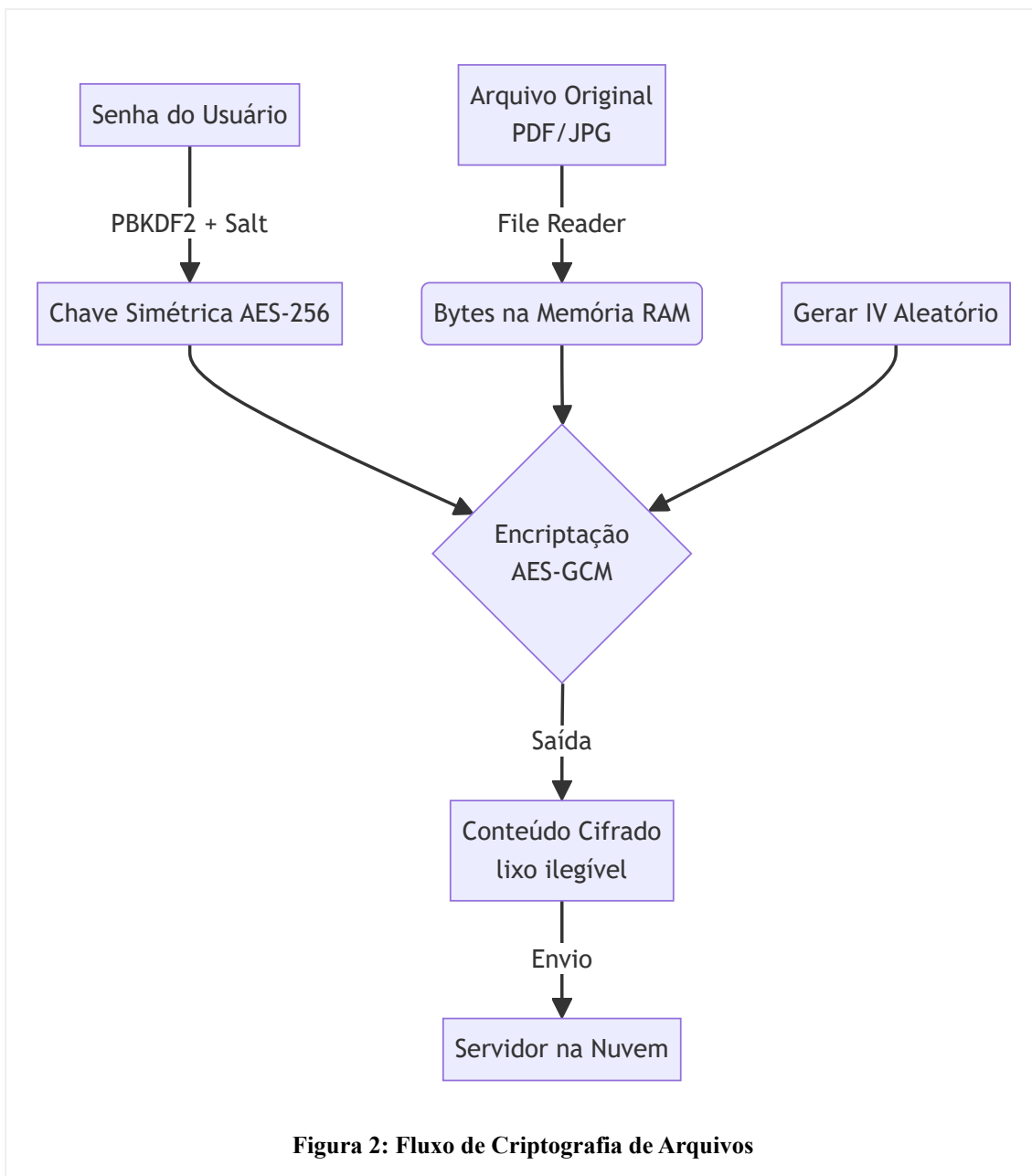
O núcleo de segurança da aplicação baseia-se em dois processos criptográficos distintos: a Derivação de Chaves e a Cifragem Simétrica.

### 4.3.1. Derivação de Chaves (PBKDF2)

Como senhas humanas raramente possuem entropia suficiente, implementou-se o algoritmo **PBKDF2** com **SHA-256**. O processo utiliza 100.000 iterações combinadas com um *Salt* único por usuário. A chave resultante de 256 bits é mantida exclusivamente na memória volátil (RAM).

### 4.3.2. Cifragem de Arquivos (AES-GCM)

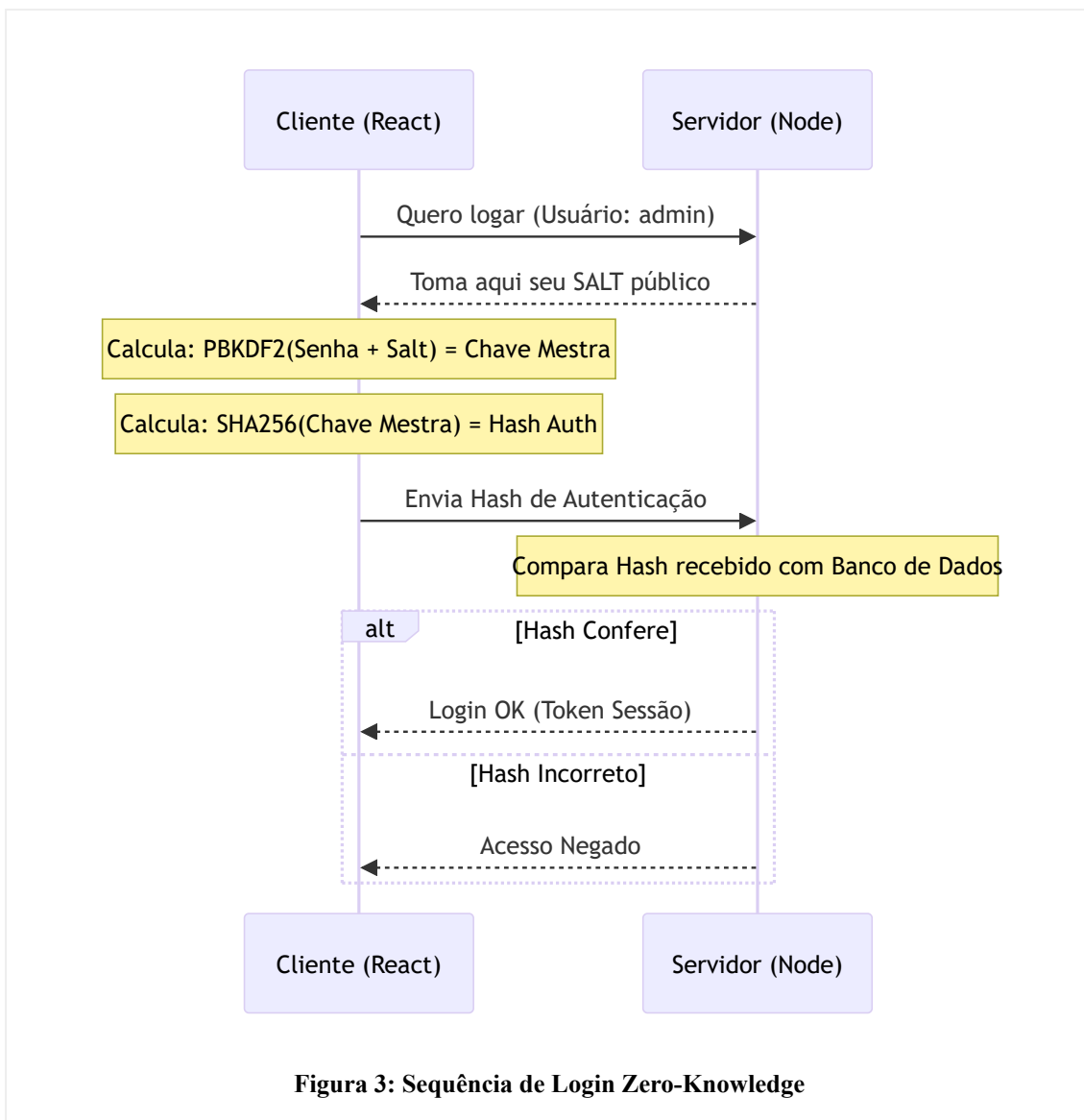
Para garantir a confidencialidade e integridade, optou-se pelo algoritmo **AES-GCM**. O fluxo segue as etapas ilustradas abaixo:



#### 4.4. Mecanismo de Autenticação Zero-Knowledge

Um dos maiores desafios foi criar um login onde o servidor pudesse validar o usuário sem conhecer a senha. Foi desenvolvida uma arquitetura de "Prova de Conhecimento":

O servidor armazena apenas um *Hash de Autenticação*, que é derivado da chave mestra. Como funções de hash são unidirecionais, o servidor consegue validar que o usuário possui a chave mestra, mas não consegue descobrir qual é essa chave.



## 4.5. Funcionalidades Adicionais

Além da criptografia central, foram implementadas camadas adicionais de proteção:

- **Logout Destrutivo:** Ao clicar em "Sair", a aplicação limpa ativamente as variáveis de estado da memória e substitui o histórico de navegação, impedindo o uso do botão "Voltar".
- **Kit de Emergência:** Geração automática de um documento PDF no lado do cliente contendo as credenciais de acesso para armazenamento físico seguro (*Cold Storage*), visto que não há recuperação de senha via servidor.