Plano de Implementação Completa

SNE Vault Protocol

Infraestrutura Soberana para Processamento de Sinais de Mercado

SNE Labs
Documentação Técnica e Arquitetural

23 de dezembro de 2025

Resumo

Este documento apresenta um plano técnico completo e institucional para implementação do **SNE Vault Protocol**, uma infraestrutura física soberana que combina edge computing, blockchain e criptografia de nível enterprise. O protocolo implementa uma arquitetura dual-kernel (**SNE Radar** para inteligência e **SNE Pass** para custódia), com prova criptográfica de uptime via ASIC, mecanismos de zeroization física, e governança on-chain através de SNIPs (SNE Improvement Proposals).

O documento serve tanto como **plano de implementação técnica** quanto como **material de estudo avançado**, cobrindo desde a arquitetura de hardware até a implementação de smart contracts, passando por segurança criptográfica, escalabilidade, observability e compliance. Cada seção inclui explicações conceituais, exemplos de código, diagramas de arquitetura e considerações de segurança.

Pré-requisitos: Conhecimento avançado em criptografia aplicada, blockchain (Ethereum/Scroll L2), arquitetura de sistemas distribuídos, hardware de segurança (HSM/TPM/-TEE) e programação de sistemas de baixo nível.

Sumário

1 Introdução e Visão Geral

1.1 Objetivo do Protocolo

O **SNE Vault Protocol** visa eliminar a dependência de infraestrutura centralizada (cloud) para processamento de sinais de mercado, transferindo a execução para **edge nodes físicos** operados por terceiros, mas com garantias criptográficas de integridade e disponibilidade.

1.2 Princípios Fundamentais

1. **Soberania Física**: Hardware como "refinaria de dados", eliminando latência institucional e pontos únicos de falha.
2. **Segregação Rigorosa**: Separação física e lógica entre módulo de inteligência (SNE Radar) e módulo de custódia (SNE Pass), garantindo que comprometimento de um não afete o outro.
3. **Prova Criptográfica**: Proof of Uptime (PoU) via ASIC/BitAxe prova existência física do hardware, mitigando ataques Sybil.
4. **Zeroization Física**: Autodestruição instantânea via Tamper-Detection Line em caso de violação física, protegendo chaves criptográficas.
5. **Governança On-Chain**: SNIPs permitem evolução do protocolo através de votação ponderada por uptime comprovado.

1.3 Componentes Principais

SNE Radar Módulo de inteligência responsável por ingestão de dados de mercado, processamento via NTE (Motor de Inferência Determinístico) e geração de sinais de execução. Executa exclusivamente em RAM volátil.
SNE Pass Módulo de custódia que gerencia chaves criptográficas no Secure Element (TPM/-TEE), realiza handshakes on-chain e deriva chaves de sessão via HKDF.
SNE Box Hardware físico (Tier 1/2/3) contendo Edge Controller (ARM/x86), BitAxe ASIC (PoU), Secure Element e Tamper-Detection Line.
SNE Relayer Serviço que recebe proofs PoU do hardware, verifica assinaturas off-chain, agrupa em batches e submete Merkle roots ao contrato on-chain.
SNE Scroll Passport Frontend Web3 que fornece componentes reutilizáveis (Wallet, Balance, Gas) e API proxy para dados on-chain.
SNE Vault Dashboard Interface de visualização para licenças, status de nós, PoU, Merkle roots e governança SNIPs.

2 Repositórios do Ecossistema

O ecossistema SNE Vault é distribuído em múltiplos repositórios, cada um com responsabilidades específicas:

Repositório	Função e Conteúdo Detalhado
`SNE-Labs/SNE-Labs`	**Frontend Dashboard e Documentação Técnica** • Dashboard React/TypeScript para visualização de licenças, keys, boxes • Componentes de visualização de PoU, Merkle roots, status de nós • Interface de governança SNIPs com votação ponderada • Documentação técnica completa (Docs.tsx) • Stack: Vite, React, TypeScript, Viem, Wagmi, TanStack Query

`SNE-Labs/SNE-Scroll-Contracts`	**Componentes Web3 e API Proxy** • Componentes reutilizáveis: WalletConnect, BalanceDisplay, GasTracker • API Proxy com cache Redis e rate limiting • Endpoints: `/api/sne/licenses`, `/api/sne/nodes/:id/status`, etc. • Stack: Express, Redis, Viem, Wagmi
`SNE-Labs/SNE-Relayer`	**Relayer para Agregação de PoU** • Recebe payloads HTTP dos nós SNE (hardware) • Verifica assinaturas EK off-chain (antes de gastar gas) • Agrupa proofs em batches (máx 100 por transação) • Constrói Merkle trees e submete ao `SNEPoURegistry` • Gerencia estratégias de gas (IMMEDIATE/OPTIMISTIC/BATCHED) • Stack: Node.js, Express, Viem, MerkleTreeJS
`Scroll L2`	**Blockchain Layer 2 - Registry On-Chain** • `SNELicenseRegistry` (ERC-721): gestão de licenças NFT • `SNEPoURegistry`: submissão de Merkle roots, contestação, cálculo de uptime • `SNEGovernance`: SNIPs, votação ponderada, execução de propostas • `SNEAttestationRegistry`: registro de EKs (Endorsement Keys)
`4LFR3Dv1/SNE-V1.0-CLOSED-BETA`	**Smart Contracts Funcionais** • Contratos Solidity auditados e deployados • ABIs para integração frontend/backend • Scripts de deploy e migração • Endereços de contratos (mainnet/testnet)
`4LFR3Dv1/ordm-testnet`	**Ambiente de Testes e Validação** • Testnet para validação antes de produção • Testes E2E com hardware simulado • Validação de fluxos completos

3 Arquitetura do Ecossistema Completo

3.1 Camada 1: Hardware Físico (Edge Nodes)

O **SNE Box** é um dispositivo físico heterogêneo composto por múltiplos módulos especializados:

3.1.1 Módulo de Inteligência (SNE Radar)

- **Edge Controller**: Processador ARM/x86 com suporte a AVX-512 para processamento vetorial de alta performance.
- **Ingestão de Dados**: Conectividade via WebSocket, satélite (Starlink) ou conexão dedicada para receber feeds de mercado em tempo real.
- **NTE (Motor de Inferência Determinístico)**: Processa tensor de estado V_t composto por:

$$V_t = [O_t, T_t, U_t, B_t, P_t, D_t] \tag{1}$$

onde:
– O_t: Osciladores (RSI, Williams%R, CCI)
– T_t: Tendência (EMA, ADX, PSAR)

  - U_t: Volume (MFI, OBV)
  - B_t: Volatilidade (Keltner, Donchian)
  - P_t: Padrões (Wedges, Head&Shoulders)
  - D_t: DOM (Bid/Ask Ratio, Liquidity)
- **Execução Volátil**: Pesos do modelo (θ) são descriptografados apenas em RAM após handshake on-chain. Zero persistência em disco.
- **Função de Confluência**:

$$y_t = \text{Softmax}\left(\sum w_i \cdot \text{Score}_i(V_t)\right) \tag{2}$$

com pesos canônicos: $w_{\text{MTF}} = 3.0$, $w_{\text{DOM}} = 2.5$, $w_{\text{ZONES}} = 2.0$, $w_{\text{SENT}} = 1.5$, $w_{\text{VOL}} = 1.0$.

3.1.2 Módulo de Resiliência (BitAxe ASIC)

- **BitAxe ASIC**: Circuito integrado dedicado para geração contínua de Proof of Uptime.
- **Hash Contínuo**: Computa hashes SHA-256 de forma contínua, provando que o hardware está operacional e dedicado.
- **Heartbeat**: Gera payload PoU periodicamente:

$$\text{payload}_{\text{PoU}} = H(\text{challenge}_{L2} || \text{ID}_{\text{node}} || t || \text{PCRs} || \text{ctr}) \tag{3}$$

- **Assinatura EK**: Proof assinado pela Endorsement Key do Secure Element:

$$\text{proof}_{\text{PoU}} = \text{Sign}_{\text{EK}}(\text{payload}_{\text{PoU}}) \tag{4}$$

3.1.3 SNE Pass (Secure Element)

- **Secure Element/TPM**: Armazena K_{root} (chave raiz) que nunca sai do hardware.
- **HKDF**: Deriva chaves de sessão via HKDF-Expand:

$$K_{\text{enc}} = \text{HKDF-Expand}(\text{HKDF-Extract}(K_{\text{root}}, \text{salt}), \text{info}, 32) \tag{5}$$

- **TPM Quotes**: Gera quotes de atestação para verificação de integridade:

$$\text{quote} = \text{Sign}_{\text{EK}}(H(\text{challenge} || \text{PCRs} || \text{nonce})) \tag{6}$$

- **Measured Boot**: Atualiza PCRs (Platform Configuration Registers) durante boot, garantindo integridade do firmware.

3.1.4 Tamper-Detection Line

- **Sensores de Continuidade Elétrica**: Linha de detecção física que circunda componentes críticos.
- **Kill Switch**: Ao detectar violação física, remove instantaneamente energia do Secure Element, tornando o dispositivo inerte.
- **Zeroization**: Múltiplas passagens de overwrite na memória DRAM para mitigar remanência.

3.2 Camada 2: Blockchain (Scroll L2)

Scroll L2 é uma Layer 2 compatível com EVM que oferece baixo custo de transação e alta throughput, ideal para registro de licenças, PoU e governança.

3.2.1 SNELicenseRegistry (ERC-721)

Contrato NFT para gestão de licenças:
- **Função Principal**: `checkAccess(nodeID)` valida se operador tem direito de ativar o nó.
- **Handshake On-Chain**: Valida assinatura do operador e libera descriptografia de θ em RAM.
- **Revogação**: Licenças podem ser revogadas on-chain, invalidando acesso do nó.
- **Eventos**: `LicenseMinted`, `LicenseRevoked`, `HandshakeValidated`.

3.2.2 SNEPoURegistry

Gerencia Proof of Uptime e Merkle roots:
- **submitMerkleRoot**: Relayers submetem roots agregados de múltiplos proofs.
- **contestRoot**: Permite contestação de roots inválidos durante janela de 24h.
- **getUptime**: Calcula uptime acumulado de um nó baseado em proofs válidos.
- **Zeroization Events**: Registra eventos `ZeroizationOccurred` sem expor segredos.

3.2.3 SNEGovernance

Governança on-chain via SNIPs:
- **submitSNIP**: Propor mudanças no protocolo.
- **vote**: Votação ponderada por poder de voto:

$$P_{\text{voto}}(\text{nodeID}) = \alpha \int_{t_0}^{t_{\text{atual}}} \text{Proof}_{\text{uptime}}(t)\, dt + \beta \cdot \text{Bond}(\text{nodeID}) \tag{7}$$

- **executeSNIP**: Executar propostas aprovadas após timelock.
- **contestSNIP**: Processo de disputa on-chain.

3.3 Camada 3: Relayer (Componente Crítico)

O **SNE Relayer** é o "carteiro" do ecossistema, recebendo proofs do hardware e agregando-os eficientemente on-chain.

3.3.1 Arquitetura do Relayer

```
apps/sne-relayer/
          src/
                verifier.ts      // Verifica assinaturas EK off-chain
                batcher.ts       // Agrupa proofs em batches (m x 100)
                gas-manager.ts   // Gerencia estrat gias de gas
                merkle-builder.ts // Constr i Merkle trees
                server.ts        // HTTP server (Express)
                index.ts         // Entry point
          tests/
                verifier.test.ts
                batcher.test.ts
                gas-manager.test.ts
          Dockerfile
```

Listing 1: Estrutura do Relayer

3.3.2 Fluxo Completo do Relayer

1. **Recebimento**: Hardware envia `POST /api/pou/submit` com payload PoU.
2. **Verificação Off-Chain**: `verifier.ts` valida:

- Challenge válido (emitido pelo contrato)
- EK registrada (on-chain ou repositório público)
- Assinatura EK válida (sem gastar gas)
- Rate limiting (1 proof/minuto por nó)
- Timestamp válido (não muito antigo, não futuro)

3. **Batching**: `batcher.ts` agrupa proofs:
   - Máximo 100 proofs por batch
   - Timeout: 5 minutos ou quando atingir 100
   - Constrói Merkle tree automaticamente
4. **Gas Management**: `gas-manager.ts`:
   - Estratégia `OPTIMISTIC`: Aguarda gas baixo antes de submeter
   - Estratégia `IMMEDIATE`: Submete imediatamente
   - Estratégia `BATCHED`: Agrupa múltiplos roots em uma transação
5. **Submissão**: Chama `submitMerkleRoot` no `SNEPoURegistry`.

3.3.3 Exemplo de Código: Verificador

```
export async function verifyPoUPayload(
  payload: PoUPayload
): Promise<{ valid: boolean; reason?: string }> {
  // 1. Verificar challenge v lido
  const challengeValid = await verifyChallenge(
    payload.nodeId,
    payload.challenge
  );
  if (!challengeValid) {
    return { valid: false, reason: 'Challenge inv lido ou expirado' };
  }

  // 2. Verificar EK registrada
  const ekRegistered = await verifyEKRegistration(payload.ekPub);
  if (!ekRegistered) {
    return { valid: false, reason: 'EK n o registrada' };
  }

  // 3. Verificar assinatura EK (off-chain, sem gastar gas)
  const isValidSignature = await verifyEKSignature(
    payload.ekPub,
    payload.ekSignature,
    payload.payloadHash
  );
  if (!isValidSignature) {
    return { valid: false, reason: 'Assinatura EK inv lida' };
  }

  return { valid: true };
}
```

Listing 2: Verificador de Assinaturas

3.4 Camada 4: Frontend e Visualização

3.4.1 SNE Vault Dashboard

Interface React/TypeScript que visualiza todo o ecossistema:

- **Visualização de Licenças**: Lista NFTs ERC-721, status (active/revoked), nodeId associado.
- **Status de Nós**: Mostra status operacional (active/violated/inactive), uptime, último heartbeat, PCR hash, zeroization events.

- **PoU e Merkle Roots**: Visualiza proofs de uptime, Merkle roots publicados, janela de contestação.
- **Governança SNIPs**: Interface para visualizar, votar e executar SNIPs, com poder de voto calculado.
- **Handshake Status**: Visualiza status de handshake (read-only), último desafio, sessão ativa.

3.4.2 SNE Scroll Passport

Fornece componentes Web3 reutilizáveis:
- **WalletConnect**: Conexão de wallet (MetaMask, WalletConnect, Injected).
- **BalanceDisplay**: Exibição de saldo ETH + tokens com cache (5min TTL).
- **GasTracker**: Preços de gas em tempo real (on-demand).
- **API Proxy**: Endpoints com cache Redis e rate limiting.

4 Fases de Implementação Detalhadas

4.1 Fase 0: Preparação e Análise (Semana 1)

4.1.1 Objetivos

Mapear todos os componentes, extrair ABIs dos contratos e configurar estrutura base do projeto.

4.1.2 Tarefas Principais

1. **Análise do Repositório Beta**:
   - Clonar `4LFR3Dv1/SNE-V1.0-CLOSED-BETA-`
   - Extrair ABIs de todos os contratos Solidity
   - Documentar endereços deployados (mainnet/testnet)
   - Identificar funções públicas e eventos
2. **Configuração de Estrutura**:
   - Decidir entre monorepo (recomendado) ou packages NPM
   - Criar estrutura de pastas conforme Apêndice A
   - Configurar `package.json` raiz e workspaces
3. **Alinhamento de Dependências**:
   - Verificar versões: `viem@^2.0.0`, `wagmi@^2.0.0`
   - Alinhar `@tanstack/react-query@^5.0.0`
   - Garantir compatibilidade React 18+

4.2 Fase 0.5: Security Requirements Specification (SRS)

4.2.1 Objetivos

Criar documento formal de requisitos de segurança, threat model e especificações de provisionamento.

4.2.2 Conteúdo do SRS

1. **Threat Model**:
   - **Nível 1**: Ataque remoto (malware, phishing)
   - **Nível 2**: Ataque físico temporário (roubo, apreensão)
   - **Nível 3**: Ataque físico com recursos de laboratório (decapsulation, micro-probing)
   - **Nível 4**: Comprometimento de supply chain (OEM, transporte)
2. **Armazenamento de Chaves**:
   - K_{root} NUNCA sai do Secure Element

- Chaves de sessão apenas em RAM volátil
- Derivação via HKDF com salt único por sessão
- Zero backup policy

3. **APIs HSM/TEE**:
   - TPM 2.0: `TPM2_HMAC`, `TPM2_Sign`, `TPM2_Quote`
   - Intel SGX: `sgx_create_enclave`, `sgx_seal_data`
   - ARM TrustZone: `TEE_GenerateRandom`, `TEE_AsymmetricSign`
4. **Provisionamento Seguro**:
   - Fabricante gera EK única
   - EK assinada por CA_HW (Certificate Authority do Hardware)
   - EK_pub + metadados registrados on-chain
   - Firmware assinado (HW_sign + SW_sign)
   - Measured boot atualiza PCRs
5. **Cadeia de Custódia**:
   - Log de transferência física (OEM → Distribuidor → Operador)
   - Verificação de integridade em cada etapa
   - RMA requer zeroization prévia
   - Certificado de destruição para dispositivos retornados
6. **Proteção contra Rollback**:
   - Firmware versioning obrigatório
   - PCRs incluem versão de firmware
   - Contrato on-chain mantém versão mínima aceita
   - Downgrade bloqueado por SE/TPM

4.3 Fase 0.6: Compliance e Regulação

4.3.1 Objetivos

Documentar requisitos de compliance, controles de exportação e normas de custódia.

4.3.2 Conteúdo

- **Controles de Exportação**:
  - ITAR (US): Pode aplicar se hardware contém criptografia militar
  - EAR (US): Export Administration Regulations
  - Wassenaar Arrangement: Controle de exportação de tecnologia criptográfica
- **Normas de Custódia**:
  - EU: MiCA (Markets in Crypto-Assets)
  - US: State-by-state (NY BitLicense, etc.)
  - UK: FCA regulations
- **Privacy (GDPR/LGPD)**:
  - Consentimento explícito
  - Direito ao esquecimento
  - Portabilidade de dados
  - Notificação de violações

4.4 Fase 1: Infraestrutura Base (Semanas 2–3)

4.4.1 Configuração Wagmi/Viem

```
import { createConfig, http } from 'wagmi';
import { scroll, scrollSepolia } from 'wagmi/chains';
import { injected, walletConnect } from '@wagmi/connectors';

```

```
const isTestnet = import.meta.env.VITE_NETWORK === 'testnet';
const chain = isTestnet ? scrollSepolia : scroll;

export const wagmiConfig = createConfig({
  chains: [chain],
  connectors: [
    injected(),
    walletConnect({
      projectId: import.meta.env.VITE_WALLETCONNECT_PROJECT_ID
    }),
  ],
  transports: {
    [chain.id]: http(
      isTestnet
        ? 'https://sepolia-rpc.scroll.io'
        : 'https://rpc.scroll.io'
    ),
  },
});
```

Listing 3: Configuração Wagmi

4.4.2 Types TypeScript Compartilhados

```
export interface SNELicense {
  tokenId: bigint;
  owner: Address;
  nodeId: string;
  status: 'active' | 'revoked' | 'expired';
  registeredAt: number;
  revokedAt?: number;
}

export interface NodeStatus {
  nodeId: string;
  status: 'active' | 'violated' | 'inactive';
  lastHeartbeat: number;
  uptime: number;
  pcrHash: string;
  merkleRoot?: string;
  zeroizationEvents: number;
}

export interface PoUData {
  nodeId: string;
  windowId: number;
  merkleRoot: string;
  lastProof: string;
  contestable: boolean;
  proofsCount: number;
  timestamp: number;
}
```

Listing 4: Types SNE

4.5 Fase 2: Integração com Passport (Semanas 4–5)

4.5.1 Remoção de Dados Hardcoded

Substituir arrays hardcoded e funções mock por leituras on-chain reais.

4.5.2 Integração de Componentes

- Importar `WalletConnect` do Passport
- Importar `BalanceDisplay` do Passport
- Importar `GasTracker` do Passport
- Compartilhar configuração Wagmi entre projetos

4.6 Fase 3: Serviços On-Chain (Semanas 6–8)

4.6.1 getLicensesForAddress

```
export async function getLicensesForAddress(
  address: Address
): Promise<SNELicense[]> {
  // Usar indexer se dispon vel (r pido)
  if (USE_INDEXER) {
    const response = await fetch(`${INDEXER_API}/licenses?addr=${address}`);
    if (response.ok) return await response.json();
  }

  // Fallback: getLogs com range limitado ( ltimos  100k blocos)
  const currentBlock = await publicClient.getBlockNumber();
  const fromBlock = currentBlock - 100000n;

  const logs = await publicClient.getLogs({
    address: contracts.SNELicenseRegistry,
    event: parseAbi([
      'event Transfer(address indexed from, address indexed to, uint256 indexed
    tokenId)'
    ]),
    args: { to: address },
    fromBlock,
    toBlock: currentBlock,
  });

  // Mapear logs para licenses (ownerOf, getLicenseStatus, getNodeId)
  return await Promise.all(logs.map(async (log) => {
    const tokenId = log.args.tokenId!;
    const owner = await publicClient.readContract({
      address: contracts.SNELicenseRegistry,
      abi: SNELicenseRegistry_ABI,
      functionName: 'ownerOf',
      args: [tokenId],
    });
    // ... mais campos
  }));
}
```

Listing 5: Buscar Licenças

4.6.2 getNodeStatus

```
export async function getNodeStatus(
  nodeId: string
): Promise<NodeStatus | null> {
  const [status, lastHeartbeat, uptime, pcrHash, merkleRoot, zeroizationCount] =
    await publicClient.multicall({
      contracts: [
        {
          address: contracts.SNEPoURegistry,
          abi: SNEPoURegistry_ABI,
```

```
          functionName: 'getNodeStatus',
          args: [nodeId],
        },
        // ... mais contratos
      ],
    });

  return {
    nodeId,
    status: statusMap[status.result as number],
    lastHeartbeat: Number(lastHeartbeat.result),
    uptime: Number(uptime.result),
    pcrHash: pcrHash.result as string,
    merkleRoot: merkleRoot.result as string,
    zeroizationEvents: Number(zeroizationCount.result),
  };
}
```

Listing 6: Status de Nó

4.7 Fase 3.5: Zeroization Proof Model (Semanas 8–9)

4.7.1 Modelo de Prova

O **ZeroizationCertificate** prova que zeroization ocorreu sem revelar segredos:

```
interface ZeroizationCertificate {
  nodeId: string;
  timestamp: number;
  tamperEventHash: string; // H(tamperLineState || timestamp || nonce)
  ekSignature: string;     // SignEK(zeroizationCertificate)
  merkleProof?: string;    // Prova de inclus o em Merkle tree
}
```

Listing 7: Zeroization Certificate

4.7.2 Fluxo

1. Tamper line violada → Kill switch ativado
2. SE gera nonce único: `nonce = CSPRNG_SE()`
3. SE computa: `tamperEventHash = H(tamperLineState || timestamp || nonce)`
4. SE assina: `certificate = SignEK(tamperEventHash)`
5. Certificate publicado on-chain (evento `ZeroizationOccurred`)
6. Nó marcado como `violated` no contrato

4.8 Fase 3.6: Indexer/Subgraph (Semanas 9–10)

4.8.1 Problema: getLogs("earliest") é Caro

Usar `getLogs("fromBlock: 'earliest'")` em produção é:

- **Lento**: Pode levar minutos para processar toda a história
- **Caro**: Consome muitos recursos do RPC
- **Inescalável**: Não funciona com muitos eventos

4.8.2 Solução: The Graph Subgraph

```
type License @entity {
  id: ID!
  tokenId: BigInt!
  owner: Bytes!
  nodeId: String!
  status: String!
  registeredAt: BigInt!
  revokedAt: BigInt
}

type NodeStatus @entity {
  id: ID!
  status: String!
  lastHeartbeat: BigInt!
  uptime: BigInt!
  pcrHash: String!
  merkleRoot: String
  zeroizationEvents: BigInt!
}
```

Listing 8: Schema do Subgraph

4.8.3 Alternativa: Custom Indexer

```
export class SNEIndexer {
  private lastProcessedBlock: number = 0;

  async start() {
    const checkpoint = await prisma.checkpoint.findUnique({
      where: { id: 'main' },
    });
    this.lastProcessedBlock = checkpoint?.lastBlock || 0;

    setInterval(() => this.processNewBlocks(), 12000);
  }

  async processNewBlocks() {
    const currentBlock = await client.getBlockNumber();

    // Processar em batches de 1000 blocos
    const batchSize = 1000n;
    let fromBlock = BigInt(this.lastProcessedBlock);

    while (fromBlock < currentBlock) {
      const toBlock = fromBlock + batchSize > currentBlock
        ? currentBlock
        : fromBlock + batchSize;

      await this.processBlockRange(fromBlock, toBlock);
      fromBlock = toBlock + 1n;
    }

    // Salvar checkpoint
    await prisma.checkpoint.upsert({
      where: { id: 'main' },
      update: { lastBlock: Number(currentBlock) },
      create: { id: 'main', lastBlock: Number(currentBlock) },
    });
  }
}
```

Listing 9: Custom Indexer

4.9 Fase 3.7: PoU Economic Model (Semanas 10–11)

4.9.1 Parâmetros do Modelo

- **Window Duration**: 1 hora (3600 segundos)
- **Contestation Window**: 24 horas após publicação do Merkle root
- **Submission Deadline**: 5 minutos antes do fim da janela
- **MAX_BATCH_SIZE**: 100 proofs por batch
- **MIN_BOND**: 1 ETH (exemplo)
- **SLASHING_PERCENTAGE**: 10% do bond

4.9.2 Gas Economics

- **Relayers pagam gas** para publicar Merkle roots
- **Incentivo**: Relayers recebem fee dos nós (off-chain) ou recompensa on-chain
- **Batching**: Múltiplos roots em uma transação (multicall)

4.9.3 Limites Anti-Spam

- **Rate Limit**: Máximo 1 proof por nó por minuto
- **Bonding**: Nós devem depositar bond para participar
- **Slashing**: Proofs inválidos resultam em slashing do bond

4.10 Fase 4: UI Completa do Dashboard (Semanas 9–11)

4.10.1 Componentes Principais

- **NodeStatusCard**: Visualiza status operacional, uptime, PCR hash, zeroization events
- **PoUDisplay**: Mostra Merkle roots, proofs count, janela de contestação
- **SNIPsList**: Lista SNIPs ativos, poder de voto, interface de votação
- **HandshakeStatus**: Status de handshake (read-only), último desafio, sessão ativa

4.11 Fase 4.5: SNE Relayer (Semanas 11–12) [CRÍTICO]

4.11.1 Implementação Completa

Ver seção **??** para detalhes completos do Relayer.

4.12 Fase 4.6: Governance Hardening (Semanas 12–13)

4.12.1 Cálculo de Poder de Voto

$$P_{\text{voto}}(\text{nodeID}) = \alpha \int_{t_0}^{t_{\text{atual}}} \text{Proof}_{\text{uptime}}(t)\, dt + \beta \cdot \text{Bond}(\text{nodeID}) \tag{8}$$

onde:

- α = peso do uptime (ex: 1.0)
- β = peso do bond (ex: 0.1)
- t_0 = timestamp de registro do nó
- t_{atual} = timestamp atual

4.12.2 Mitigação de Ataques Sybil

- **Cap por operador**: Máximo 10% do poder de voto total
- **Mínimo de uptime**: 30 dias para votar
- **Quorum mínimo**: 20% do poder de voto total para aprovar SNIP
- **Decay temporal**: 1% por mês

4.12.3 Processo de Disputa

1. Qualquer operador pode contestar um SNIP
2. Disputa on-chain com evidências
3. Votação de disputa (7 dias)
4. Se disputa válida: SNIP rejeitado, proposer perde bond
5. Se disputa inválida: Contestante perde bond

4.13 Fase 5: API Proxy no Passport (Semanas 13–14)

4.13.1 Endpoints

- `GET /api/sne/licenses?addr={address}`
- `GET /api/sne/nodes/:nodeId/status`
- `GET /api/sne/nodes/:nodeId/pou`
- `GET /api/sne/governance/snips`
- `GET /api/sne/merkle/roots`

4.13.2 Cache e Rate Limiting

- **Cache Redis**: TTLs por tipo de dado (licenças: 5min, status: 30s)
- **Rate Limiting**: 30 requisições por minuto por IP
- **Error Handling**: Fallback para leitura on-chain direta

4.14 Fase 6: Handshake Visualization (Semanas 14–15)

Visualização read-only do status de handshake (não executa handshake, apenas mostra status).

4.15 Fase 7: Testes e Validação (Semanas 16–17)

4.15.1 Testes Unitários

- Vitest para serviços TypeScript
- Mocks de Viem/Wagmi providers
- Fixtures para simular hardware

4.15.2 Testes de Integração

- Testes com Wagmi providers reais (testnet)
- Validação de fluxos completos

4.15.3 Testes E2E

- Playwright com fluxo de wallet
- Testes de UI completos

4.16 Fase 7.5: Testes On-Chain Avançados (Semanas 17–18)

4.16.1 Foundry/Hardhat

```
contract SNELicenseRegistryTest is Test {
    SNELicenseRegistry registry;
    address operator = address(0x1234);
    string nodeId = "node-001";

    function setUp() public {
        registry = new SNELicenseRegistry();
    }

    function testCheckAccess() public {
        uint256 tokenId = registry.mint(operator, nodeId);
        bool access = registry.checkAccess(nodeId);
        assertTrue(access);
    }
}
```

Listing 10: Teste Foundry

4.16.2 Fixtures para Simular Hardware

```
export const MOCK_EK_PRIVATE_KEY = '0x' + '1'.repeat(64);
export const MOCK_EK_ACCOUNT = privateKeyToAccount(MOCK_EK_PRIVATE_KEY);

export async function mockTPMQuote(
  challenge: string,
  pcrHash: string
): Promise<{ quote: string; ekPub: string }> {
  const payload = `${challenge}${pcrHash}`;
  const signature = await MOCK_EK_ACCOUNT.signMessage({ message: payload });
  return { quote: signature, ekPub: MOCK_EK_ACCOUNT.address };
}
```

Listing 11: Fixtures de Hardware

4.17 Fase 8: Deploy e Produção (Semanas 18–19)

4.17.1 Ambientes

- **dev**: Desenvolvimento local
- **testnet**: Scroll Sepolia para testes
- **mainnet**: Scroll L2 para produção

4.17.2 CI/CD

- Pipelines para build, tests, deploy canary
- Rollback automático em caso de falhas

4.18 Fase 8.5: Observability Completa (Semanas 19–20)

4.18.1 Métricas Prometheus

```
export const metrics = {
  apiRequests: new Counter({
    name: 'sne_api_requests_total',
    help: 'Total de requisi  es    API',
```

```
    labelNames: ['endpoint', 'method', 'status'],
  }),
  apiLatency: new Histogram({
    name: 'sne_api_latency_seconds',
    help: 'Lat ncia de requisi  es    API',
    buckets: [0.1, 0.5, 1, 2, 5, 10],
  }),
  activeNodes: new Gauge({
    name: 'sne_active_nodes',
    help: 'N mero de n s ativos',
  }),
};
```

Listing 12: Métricas

4.18.2 SLI/SLO Definitions

- **API Availability**: 99.9% (30 dias)
- **API Latency (p95)**: $<$ 500ms (1 hora)
- **On-Chain Read Success**: 99.5% (1 hora)
- **Cache Hit Rate**: $>$ 80% (1 hora)

4.18.3 Playbooks de Incidente

1. **Zeroization Detectado** (CRÍTICO): Isolar nó, verificar certificado, revogar licença
2. **Relayer Offline** (ALTA): Verificar status, restart, failover
3. **Gas Price Spike** (MÉDIA): Aguardar gas baixo, ajustar estratégia
4. **Ataque Sybil Detectado** (CRÍTICO): Identificar nós, aplicar slashing
5. **Indexer Desatualizado** (MÉDIA): Verificar lag, fallback, backfill

4.18.4 Plano de Pentest

- **Fase 1**: Reconnaissance (1 semana)
- **Fase 2**: Vulnerability Assessment (2 semanas)
- **Fase 3**: Exploitation (1 semana)
- **Fase 4**: Reporting (1 semana)
- **Critérios de Aprovação**: Zero vulnerabilidades críticas, zero altas não mitigadas

5 Serviços On-Chain — Exemplos Completos

5.1 getLicensesForAddress

Ver código completo na Fase 3.

5.2 checkLicenseAccess

```
export async function checkLicenseAccess(
  nodeId: string
): Promise<boolean | null> {
  try {
    const result = await publicClient.readContract({
      address: contracts.SNELicenseRegistry,
      abi: SNELicenseRegistry_ABI,
      functionName: 'checkAccess',
      args: [nodeId],
    });
```

```
    return result as boolean;
  } catch (error) {
    console.error('Error checking license access:', error);
    return null;
  }
}
```

Listing 13: Verificar Acesso

5.3 getPoUData

```
export async function getPoUData(nodeId: string): Promise<PoUData | null> {
  const [windowId, merkleRoot, lastProof, contestable, proofsCount, timestamp] =
    await publicClient.multicall({
      contracts: [
        {
          address: contracts.SNEPoURegistry,
          abi: SNEPoURegistry_ABI,
          functionName: 'getCurrentWindow',
          args: [],
        },
        // ... mais contratos
      ],
    });

  return {
    nodeId,
    windowId: Number(windowId.result),
    merkleRoot: merkleRoot.result as string,
    lastProof: lastProof.result as string,
    contestable: contestable.result as boolean,
    proofsCount: Number(proofsCount.result),
    timestamp: Number(timestamp.result),
  };
}
```

Listing 14: Dados de PoU

6 Zeroization — Modelo Detalhado

Ver Fase 3.5 para modelo completo de ZeroizationCertificate.

7 Governança SNIPs — Detalhamento

7.1 Lifecycle de um SNIP

1. **draft**: Proposta criada, aguardando submissão
2. **active**: Em votação (7 dias)
3. **passed**: Aprovado, aguardando timelock
4. **rejected**: Rejeitado pela comunidade
5. **executed**: Executado on-chain

7.2 Cálculo de Poder de Voto

Ver Fase 4.6 para fórmula completa e mitigação Sybil.

8 Checklist de Implementação Completo

Task	Status / Nota
Fase 0: Preparação	
Clonar e analisar repositório beta	[ ]
Extrair ABIs e documentar endereços	[ ]
Configurar monorepo / packages	[ ]
Alinhar dependências (Viem, Wagmi, React, TS)	[ ]
Fase 0.5: SRS	
Criar Threat Model (4 níveis)	[ ]
Documentar HSM/TEE APIs	[ ]
Definir cadeia de custódia	[ ]
Implementar Attestation Flow	[ ]
Fase 0.6: Compliance	
Documentar controles de exportação	[ ]
Normas de custódia por jurisdição	[ ]
Privacy (GDPR/LGPD)	[ ]
Fase 1: Infraestrutura Base	
Configurar Wagmi/Viem	[ ]
Criar types TypeScript	[ ]
Configurar TanStack Query	[ ]
Fase 2: Integração Passport	
Integrar componentes (Wallet, Balance, Gas)	[ ]
Remover dados hardcoded	[ ]
Criar hooks useSNEData	[ ]
Fase 3: Serviços On-Chain	
Implementar getLicensesForAddress	[ ]
Implementar checkLicenseAccess	[ ]
Implementar getNodeStatus	[ ]
Implementar getPoUData	[ ]
Implementar getMerkleRoots	[ ]
Implementar getSNIPs	[ ]
Implementar getVotingPower	[ ]
Fase 3.5: Zeroization	
Implementar ZeroizationCertificate	[ ]
Eventos on-chain ZeroizationOccurred	[ ]
Verificação de certificados	[ ]
Fase 3.6: Indexer	
Configurar The Graph Subgraph OU Custom Indexer	[ ]
Cursor-based pagination	[ ]
Checkpoints e processamento em batches	[ ]
Fase 3.7: PoU Economic Model	
Definir parâmetros (bond, slashing, janelas)	[ ]

Implementar contestability	[ ]
Gas economics (relayers pagam)	[ ]
Stress testing	[ ]
Fase 4: UI Completa	
Componente NodeStatusCard	[ ]
Componente PoUDisplay	[ ]
Componente SNIPsList	[ ]
Componente HandshakeStatus	[ ]
Dashboard principal (integra todos)	[ ]
Fase 4.5: SNE Relayer [CRÍTICO]	
Estrutura de pastas criada	[ ]
Verificador de assinaturas (verifier.ts)	[ ]
Batcher (batcher.ts) - agrupa 100 proofs	[ ]
Gas Manager (gas-manager.ts)	[ ]
HTTP Server (server.ts) - recebe payloads	[ ]
Rate limiting por nodeId	[ ]
Merkle tree builder	[ ]
Testes unitários	[ ]
Deploy do relayer em produção	[ ]
Monitoramento do relayer	[ ]
Fase 4.6: Governance Hardening	
Caps por operador (10%)	[ ]
Timelocks (7 dias)	[ ]
Mitigação Sybil	[ ]
Processo de disputa detalhado	[ ]
Fase 5: API Proxy	
Endpoints implementados	[ ]
Cache Redis configurado	[ ]
Rate limiting funcionando	[ ]
Error handling	[ ]
Fase 6: Handshake Visualization	
Service getHandshakeStatus	[ ]
Componente HandshakeStatus	[ ]
Fase 7: Testes	
Testes unitários (Vitest)	[ ]
Testes de integração	[ ]
Testes E2E (Playwright)	[ ]
Testes on-chain (testnet)	[ ]
Fase 7.5: Testes Avançados	
Foundry/Hardhat configurado	[ ]
Fixtures para simular hardware	[ ]
Testes com forking	[ ]
Simulação de reorgs	[ ]
Fase 8: Deploy	
Ambientes configurados (dev/testnet/mainnet)	[ ]

CI/CD pipelines	[ ]
Deploy Dashboard	[ ]
Deploy Passport API	[ ]
Deploy Relayer	[ ]
Fase 8.5: Observability	
Métricas Prometheus expostas	[ ]
SLI/SLO definitions	[ ]
Alertas configurados (PagerDuty)	[ ]
Dashboards Grafana	[ ]
Playbooks de incidente criados	[ ]
Plano de pentest aprovado	[ ]
Auditoria e Compliance	
Auditoria de contratos (empresa especializada)	[ ]
Pentest hardware/firmware	[ ]
Compliance documentado e aprovado	[ ]
E2E testnet com hardware real	[ ]

9 Riscos Críticos e Mitigações Detalhadas

9.1 Segurança de Chaves e Cadeia de Custódia

Risco: Comprometimento físico do SNE Box → comprometimento de PoU & assinatura de provas.

Mitigação:

- **HSM/TEE**: TPM 2.0, Intel SGX, ARM TrustZone para armazenamento seguro
- **Attestation**: TPM Quotes verificam integridade do firmware
- **Provisionamento Seguro**: EK assinada por CA_HW, registro on-chain
- **Cadeia de Custódia**: Log de transferência física, verificação em cada etapa
- **Key Rotation**: Política de rotação de chaves (se necessário)
- **Playbook de Incidente**: Resposta rápida a comprometimento

9.2 Zeroization e Evidência Auditável

Risco: Provar que zeroization ocorreu sem vazar segredos.

Mitigação:

- **ZeroizationCertificate**: Hash + assinatura EK (não revela K_{root})
- **Evento On-Chain**: `ZeroizationOccurred` registra ocorrência
- **Verificação**: Certificado verificável publicamente
- **Auditoria**: Histórico completo de zeroizations on-chain

9.3 Escalabilidade de Logs

Risco: Uso de `getLogs("fromBlock: 'earliest'")` é caro e lento.

Mitigação:

- **Indexer/Subgraph**: The Graph ou Custom Indexer Node.js
- **Pagination**: Cursor-based pagination para logs
- **Checkpoints**: Processamento incremental com checkpoints
- **Batching**: Processamento em batches de 1000 blocos
- **Fallback**: Leitura on-chain direta se indexer falhar

9.4 PoU Economics e Contestability

Risco: Parâmetros indefinidos, spam de proofs, DoS.

Mitigação:

- **Parâmetros Cristalinos**: Bond, slashing, janelas definidos claramente
- **Rate Limiting**: 1 proof/minuto por nó
- **Bonding**: Nós devem depositar bond para participar
- **Slashing**: Proofs inválidos resultam em slashing (10% do bond)
- **Gas Economics**: Relayers pagam gas, batching otimiza custos
- **Stress Testing**: Testes com 1000+ nós simultâneos

9.5 Governança e Ataques Sybil

Risco: Voto comprável, colusão, ataques Sybil.

Mitigação:

- **Caps por Operador**: Máximo 10% do poder de voto total
- **Timelocks**: 7 dias entre aprovação e execução
- **Decay Temporal**: 1% por mês no poder de voto
- **Quorum Mínimo**: 20% do poder de voto total para aprovar
- **Processo de Disputa**: On-chain com penalidades
- **Slashing**: Voto malicioso resulta em slashing

9.6 Observability e Resposta a Incidentes

Risco: Falta de monitoramento, alertas, playbooks.

Mitigação:

- **Métricas Prometheus**: Requests, latency, cache hit rate
- **SLI/SLO**: 99.9% availability, p95 < 500ms
- **Alertas PagerDuty**: Para incidentes críticos
- **Dashboards Grafana**: Visualização de métricas
- **Playbooks**: 5 cenários de incidente documentados
- **Pentest**: Plano de 4 fases com critérios de aprovação

10 Testes e Validação Detalhados

10.1 Testes Unitários

- **Framework**: Vitest
- **Cobertura**: Serviços TypeScript, hooks, utilities
- **Mocks**: Viem/Wagmi providers, contratos

10.2 Testes de Integração

- **Ambiente**: Scroll Sepolia testnet
- **Fluxos**: Wallet connection, leitura de contratos, votação

10.3 Testes E2E

- **Framework**: Playwright
- **Cenários**: Fluxo completo de usuário, wallet, visualização de dados

10.4 Testes On-Chain Avançados

- **Foundry/Hardhat**: Testes de contratos Solidity
- **Forking**: Fork da mainnet para testes realistas
- **Fixtures**: Simulação de hardware (EK, TPM Quotes, PoU proofs)
- **Reorgs**: Simulação de reorganizações de blockchain

11 Deploy e Operação

11.1 Ambientes

- **dev**: Desenvolvimento local, hot-reload
- **testnet**: Scroll Sepolia, validação antes de produção
- **mainnet**: Scroll L2, produção

11.2 CI/CD

- **Build**: Vite para frontend, tsc para TypeScript
- **Tests**: Execução automática de todos os testes
- **Deploy Canary**: Deploy gradual com rollback automático

11.3 Observability

- **Prometheus**: Coleta de métricas
- **Grafana**: Dashboards e visualização
- **Sentry**: Error tracking
- **Logs**: Estruturados (JSON) para análise

12 Métricas de Sucesso

- **Dados Hardcoded**: Redução para 0%
- **Integração On-Chain**: 100% dos fluxos
- **Dashboard**: Mostra PoU, Merkle roots, status de nós
- **API Proxy**: Cache hit rate $> 80\%$
- **Auditoria**: Zero vulnerabilidades high-severity outstanding
- **Uptime**: 99.9% para Tier 1, 99.5% para Tier 2, 99.0% para Tier 3

13 Próximos Passos Imediatos

1. **Fase 0.5**: Criar Threat Model & SRS (prioridade alta)
2. **Fase 0.6**: Documentar Compliance e Regulação
3. **Fase 3.5**: Implementar zeroization proof model
4. **Fase 3.6**: Configurar indexer/subgraph (The Graph ou custom)
5. **Fase 3.7**: Definir e implementar PoU economic model
6. **Fase 4.5**: Implementar SNE Relayer (CRÍTICO - recebe payloads do hardware)
7. **Fase 4.6**: Implementar governance hardening
8. **Fase 7.5**: Setup testes on-chain com Foundry
9. **Fase 8.5**: Implementar observability completa
10. **Auditoria**: Planejar auditoria smart contracts e pentest hardware
11. **E2E**: Executar testes E2E na testnet com hardware simulado

Apêndices

A. Estrutura de Arquivos Sugerida

```
SNE-Monorepo/
          packages/
                sne-vault/              # Dashboard
                      src/
                            app/
                                  pages/
                                  components/
                            lib/
                            services/
                            hooks/
                      package.json
                      vite.config.ts
                sne-passport/            # Passport
                sne-contracts/           # ABIs e types
                sne-sdk/                 # SDK compartilhado
          apps/
                sne-relayer/             # Relayer (CR TICO)
                      src/
                            batcher.ts
                            verifier.ts
                            gas-manager.ts
                            server.ts
                      Dockerfile
                sne-box-firmware/        # Firmware (futuro)
          package.json
```

B. Exemplo de .env.production

```
# Network
VITE_NETWORK=mainnet

# RPC
VITE_SCROLL_RPC=https://rpc.scroll.io

# WalletConnect
VITE_WALLETCONNECT_PROJECT_ID=your_project_id

# API
VITE_SNE_API_URL=https://pass.snelabs.space/api

# Indexer
VITE_USE_INDEXER=true
VITE_INDEXER_API_URL=https://indexer.snelabs.space

# Contratos (do beta repo)
VITE_SNE_LICENSE_REGISTRY=0x...
VITE_SNE_POU_REGISTRY=0x...
VITE_SNE_GOVERNANCE=0x...
VITE_SNE_ATTESTATION_REGISTRY=0x...
```

C. Parâmetros do PoU Economic Model

- `WINDOW_DURATION` = 3600s (1 hora)
- `CONTESTATION_WINDOW` = 86400s (24 horas)
- `SUBMISSION_DEADLINE` = 300s (5 minutos)

- `MAX_BATCH_SIZE` = 100 proofs
- `MIN_BATCH_SIZE` = 10 proofs
- `BATCH_TIMEOUT` = 300s (5 minutos)
- `MIN_BOND` = 1 ETH (exemplo)
- `SLASHING_PERCENTAGE` = 10%
- `MAX_PROOFS_PER_NODE_PER_MINUTE` = 1

D. SLI/SLO Definitions

- **API Availability**: SLI = `(requests_200 + requests_429) / total_requests`, SLO = 99.9% (30 dias)
- **API Latency**: SLI = p95 latency, SLO = p95 < 500ms (1 hora)
- **On-Chain Read Success**: SLI = `successful_reads / total_reads`, SLO = 99.5% (1 hora)
- **Cache Hit Rate**: SLI = `cache_hits / (cache_hits + cache_misses)`, SLO = > 80% (1 hora)
- **Node Uptime**: Tier 1 = 99.9%, Tier 2 = 99.5%, Tier 3 = 99.0%

E. Glossário

PoU Proof of Uptime - Prova criptográfica de existência física do hardware
EK Endorsement Key - Chave única do Secure Element, assinada por CA_HW
TPM Trusted Platform Module - Módulo de segurança de hardware
TEE Trusted Execution Environment - Ambiente de execução confiável
SNIP SNE Improvement Proposal - Proposta de melhoria do protocolo via governança
NTE Motor de Inferência Determinístico - Processa tensor de estado V_t
PCR Platform Configuration Register - Registro do TPM para measured boot
HKDF HMAC-based Key Derivation Function - Função de derivação de chaves
AEAD Authenticated Encryption with Associated Data - Criptografia autenticada
Zeroization Autodestruição física via Tamper-Detection Line
Relayer Serviço que agrega proofs PoU e submete Merkle roots on-chain

Documento gerado a partir do Plano de Implementação Completa — SNE Vault Protocol.
SNE Labs — Infraestrutura Soberana para Processamento de Sinais de Mercado