🌌 NOCTURNE: Documentação Técnica e Arquitetural (v1.0)

Data: 17 de Dezembro de 2025
Autores: Rafael Oliveira, Jamerson Bednarski

1. Visão Geral do Sistema e Princípios Fundamentais

NOCTURNE é um **Sistema de Estado Distribuído com Esquecimento Inteligente** (Intelligent Forgetting Distributed State System). Seu objetivo principal é gerenciar o estado de dados em um ambiente distribuído, garantindo que o ciclo de vida dos dados (desde a ingestão até a remoção) seja auditável criptograficamente e regido por políticas de retenção estritas. O NOCTURNE adota uma arquitetura que prioriza a conformidade legal (GDPR, CCPA, etc.) através de garantias formais de deleção.

1.1 Quatro Princípios Fundamentais

1. **Bounded Memory (Memória Delimitada):** Todo dado possui um tempo de vida finito e obrigatório, garantido por políticas de esquecimento ativas. O sistema deve resistir à retenção acidental ou infinita.
2. **Cryptographic Audit (Auditoria Criptográfica):** Toda ação de deleção (esquecimento) deve gerar uma prova irrefutável e verificável por terceiros (proof of forgetfulness), sem revelar o dado original.
3. **Modular Extensibility (Extensibilidade Modular):** O núcleo do motor de estado e esquecimento é mínimo. Funcionalidades avançadas (e.g., Blockchain Anchoring, Digital Asset Management) são integradas como extensões opcionais.
4. **Partition Tolerance (Tolerância a Partições):** A sincronização entre nós é baseada em estruturas de dados replicadas livres de conflito (CRDTs), garantindo convergência eventual do estado, mesmo após falhas de rede.

2. Componentes Fundamentais

2.1 MemoryCell (Unidade Atômica de Estado)

A MemoryCell é o contêiner fundamental de dados. Sua identidade é criptograficamente ligada ao seu conteúdo e ao seu momento de criação.

Campo	Tipo	Função	Invariante de Segurança
id	Hash (SHA256)	Identificador único da célula.	Deve ser calculado como `HASH(data

data	Data	Carga útil (payload) opaca do dado.	Nunca deve ser revelada em ForgetProof.
timestamp	Instant	Momento de ingestão (monotônico).	timestamp é monotônico crescente, prevenindo a violação causal (I2).
tags	Set<String>	Metadados usados por políticas de esquecimento.	Não deve conter o prefixo "compliance::".
metadata	Map	Metadados extensíveis para uso da aplicação.	Opcional.

2.2 ForgetPolicy (Políticas de Esquecimento)

Definem o predicado de "esquecimento". O sistema aplica uma lista de políticas a cada célula no ciclo de esquecimento. Se **qualquer** política for avaliada como True, a célula é marcada para remoção.

Estruturas de Política (Exemplos):

- TimeBasedPolicy(max_age: Duration): Remove células mais antigas que max_age.
- AccessBasedPolicy(max_idle: Duration): Remove células que não foram lidas/acessadas por um período.
- CompliancePolicy(regulation: Regulation): Remove dados com base em requisitos legais (e.g., "dados de cliente europeu não podem exceder 7 anos").

Garantia: A avaliação da política deve ser **determinística** (I5).

2.3 TraumaEngine (Motor de Estado Principal)

O orquestrador central do estado, gerenciamento de políticas e geração de provas.

Componente	**Tipo**	**Função**	**Invariante Crítico**
buffer	Sequence<MemoryCell>	Estado ativo dos dados.	Tamanho máximo limitado por config.max_capacity.

policies	List<ForgetPolicy>	Políticas aplicáveis para o ciclo de esquecimento.	
proofs	List<ForgetProof>	Histórico imutável de todas as deleções formais.	Deve conter o hash de toda célula removida (I1).
tombstones	Set<CellID>	Registro de IDs permanentemente deletados.	buffer ∩ tombstones = ∅ (I3).

3. Algoritmos Nucleares

3.1 Algoritmo 1: Ingestão (Ingest)

Propriedades: Idempotente, Determinístico, O(1) Amortizado.

```
Input: data, tags, metadata
Output: cell_id (hash)

1. timestamp = monotonic_clock()
2. cell = MemoryCell(
      id = SHA256(data || timestamp),
      data = data,
      timestamp = timestamp,
      tags = tags ∪ infer_tags(data), # infer_tags é opcional
      metadata = metadata
   )
3. IF config.deduplication_enabled:
      IF ∃ cell' ∈ buffer: cell'.id == cell.id:
         RETURN cell'.id  # Deduplicação baseada em ID
4. buffer.append(cell)
5. RETURN cell.id
```

3.2 Algoritmo 2: Ciclo de Esquecimento (Forget Cycle)

Este algoritmo executa a purga de estado baseada nas políticas.

Complexidade: O(n × m) onde n = |buffer|, m = |policies|.

```
Input: engine (TraumaEngine)
```

Output: proofs_generated (List<ForgetProof>)

1. to_remove = []
2. FOR cell IN engine.buffer:
   IF ANY policy IN engine.policies.evaluate(cell) == True:
   to_remove.append(cell)
3. FOR cell IN to_remove:
   proof = Algoritmo 3: Geração de ForgetProof(cell, "policy_trigger")

   # ATOMIC OPERATION START (Crucial para I3)
   engine.buffer.remove(cell)
   engine.tombstones.add(cell.id)
   engine.proofs.append(proof)
   # ATOMIC OPERATION END

4. RETURN proofs_generated

4. Sistema de Auditoria Criptográfica

A garantia de **Cryptographic Audit** é fornecida pela estrutura ForgetProof e pelo uso de Árvores de Merkle.

4.1 ForgetProof (Prova de Esquecimento)

É o artefato gerado no momento da deleção, garantindo **Non-repudiation** e **Verifiability**.

Campo	**Função**	**Propriedade**
cell_id	ID da célula esquecida.	Público. Permite auditoria.
cell_hash	Hash do **conteúdo** da célula (sem timestamp).	Requerido para verificação por terceiros.
forgotten_at	Timestamp exato da deleção.	Utilizado em verificação LWW em sincronização.
reason	Motivo da deleção (e.g., "TimeBasedPolicy expiração").	Semântico para relatórios.
merkle_path	Caminho necessário para provar inclusão da deleção	Opcional (se merkle_enabled).

	na Raiz Merkle.	
witness	Hash de transação ou recibo de ancoragem em Blockchain.	Opcional (se blockchain_enabled).

4.2 Merkle Tree (Para Batch Proofs)

Deleções são batched (agrupadas) em uma Árvore de Merkle para reduzir custos de auditoria e de ancoragem.

MerkleTree:
- leaves: [Hash das células deletadas]
- root: Hash (Raiz da árvore)
- depth: Int

4.3 Algoritmo de Verificação por Terceiros

Este algoritmo permite que um auditor externo verifique a deleção sem acesso ao *buffer* ativo ou aos dados originais.

Algorithm verify_deletion(auditor, proof, alleged_data, merkle_root):
1. # Garante Confidentiality: alleged_data é fornecido pelo auditor
   IF proof.cell_hash != SHA256(alleged_data):
      RETURN False (O dado fornecido não corresponde à prova)

2. # Verifica a Inclusão no Estado Criptográfico (Integrity)
   IF proof.merkle_path:
      computed_root = compute_root_from_path(proof.cell_hash, proof.merkle_path)
      IF computed_root != merkle_root:
         RETURN False (A prova Merkle falhou)

3. # Verifica a Ancoragem On-Chain (Non-repudiation)
   IF proof.witness:
      IF not blockchain_verify(proof.witness):
         RETURN False (A transação de ancoragem não é válida)

4. RETURN True

5. Sincronização Distribuída (CRDT-based)

Para garantir **Partition Tolerance** e **Eventual Consistency**, o NOCTURNE utiliza um modelo de Registro de Última Gravação Vencedora (LWW-Register) modificado para lidar com deleções formais.

5.1 Estruturas de Sincronização

- vector_clock: Controla a ordem causal dos eventos em cada nó.
- tombstones: Conjunto de IDs de células deletadas. Essencial para evitar a **Ressurreição de Dados Esquecidos** (I3).
- active_cells: Mapa CellID -> MemoryCell.

5.2 Algoritmo 4: Sincronização entre Nós

Input: local_state, remote_message
Output: new_local_state, reply_message

```
1. # Processar MUDANÇAS no buffer
   FOR change IN remote_message.changes:
       CASE change OF:
          Added(new_cell):
             IF new_cell.id ∉ local_state.tombstones:
                # LWW: Conflito resolvido pelo timestamp mais recente
                IF new_cell.id ∉ local_state.active_cells OR
                   new_cell.timestamp > local_state.active_cells[new_cell.id].timestamp:
                   local_state.active_cells[new_cell.id] = new_cell

          Removed(cell_id, remote_proof):
             # O esquecimento formal é FINAL e sempre vence
             IF verify_proof(remote_proof):
                local_state.tombstones.add(cell_id)
                local_state.active_cells.remove(cell_id)
                local_state.proofs.append(remote_proof)

2. # Atualizar relógios e preparar resposta com mudanças locais
   local_state.vector_clock.merge(remote_message.vector_clock)
   reply_changes = compute_changes_since(remote_message.vector_clock)

   RETURN (local_state, SyncMessage(local_id, local_state.vector_clock, reply_changes))
```

Propriedade Garantida: Convergência Distribuída (I4). O processo de "esquecimento" é um evento final que sempre vence o processo de "adição" (sempre que a prova de esquecimento é validada).

6. Invariantes e Garantias Formais

A segurança e a auditabilidade do NOCTURNE dependem da manutenção rigorosa de seus invariantes.

6.1 Invariantes do Sistema e Consequências da Violação

Invariante	Definição	Ameaça Mitigada	Consequência Crítica da Violação
I1: Prova de Existência	Toda prova gerada deve corresponder a um único dado histórico.	Fraude de Auditoria / Negação de Deleção.	**Invalidez da Auditoria:** O histórico de auditoria se torna não confiável e contestável em um tribunal.
I2: Monotonicidade Temporal	Todos os timestamps de célula são válidos e crescentes.	Violação Causal / Viagem no Tempo.	**Falha na Política de Esquecimento:** Deleções prematuras ou tardias, comprometendo o princípio de **Bounded Memory**.
I3: Não-Ressurreição	buffer e tombstones não podem ter interseção.	Ressurreição de Dados Esquecidos (Zombie Data).	**Violação da Conformidade (Legal Hold).** Dado irrecuperável por lei se torna acessível novamente.
I4: Convergência Distribuída	Nós sempre convergem para o mesmo estado final.	Inconsistência Permanente de Partições.	**Estado Fragmentado:** Leitura inconsistente e execução não uniforme de políticas em todo o sistema.

I5: Determinizmo de Política	A avaliação de ForgetPolicy é determinística.	Esquecimento Não Auditável/Arbitrário.	**Impossibilidade de Verificação:** O auditor não consegue reproduzir a decisão de esquecimento, violando a **Auditability**.

6.2 Propriedades de Segurança

1. **Confidentiality:** As provas de deleção (públicas) não revelam o dado original. (Requisito: Uso de SHA-256 no conteúdo do dado e não no dado em si).
2. **Integrity:** As provas são seladas contra adulteração. (Requisito: Uso de Merkle Tree e/ou ancoragem Blockchain).
3. **Non-repudiation:** A deleção, uma vez provada, não pode ser negada pelo nó ou administrador.
4. **Auditability:** O histórico completo (incluindo todas as provas) está disponível para inspeção por terceiros.

7. Extensibilidade e Integração (Opcional)

7.1 Integração Blockchain (Ancoragem)

A ancoragem em uma L1 (Layer 1) ou L2 (Layer 2) é crucial para fornecer um *timestamp* e *imprimatur* de terceiros que garante a prova de deleção contra um atacante que controla todos os nós do NOCTURNE.

- **Processo:** O motor agrega um lote (batch) de ForgetProofs, constrói uma Raiz Merkle a partir de seus hashes, e publica essa raiz na Blockchain em uma única transação (TxHash).
- **Eficiência:** Prioriza o modelo BatchAnchor para otimização de custo (gas), enviando 1 Tx para N deleções.

7.2 Digital Asset Management (DAM Extension)

A extensão DAM lida com dados maiores (Imagens, Vídeos, 3D Models) que possuem um ciclo de vida e requisitos de armazenamento específicos.

- **Extensão da Célula:** DigitalAsset herda de MemoryCell e adiciona campos como storage_tier (Hot/Cold/Archived) e retention_policy (com legal_hold).
- **Políticas Específicas:** Inclui StorageQuotaPolicy para forçar o esquecimento quando um limite de armazenamento (por *tier*) é atingido.

8. Interface Canônica e Guia de Implementação

A interface do NOCTURNE deve ser definida usando uma Linguagem de Definição de Interface (IDL) como gRPC/Protobuf para garantir interoperabilidade e portabilidade entre linguagens.

8.1 Interface de Serviço (Pseudo-IDL)

```
service NocturneCore {
  // Ingestão de novos dados, retorna o ID criptográfico da célula.
  rpc Ingest(IngestRequest) returns (IngestResponse);

  // Executa o ciclo de esquecimento em todas as células ativas.
  rpc ForgetCycle(ForgetRequest) returns (ForgetResponse);

  // Consulta provas de esquecimento geradas dentro de um período ou por ID.
  rpc GetProofs(ProofsRequest) returns (ProofsResponse);

  // Troca de mensagens de sincronização CRDT entre nós.
  rpc Sync(SyncRequest) returns (SyncResponse);

  // Verifica a validade de uma prova de deleção, externamente ou internamente.
  rpc Verify(VerifyRequest) returns (VerifyResponse);
}
```

8.2 Complexidades Alvo

A implementação deve visar as seguintes complexidades algorítmicas para garantir performance em escala:

Operação	**Complexidade Alvo**	**Comentários**
Ingestão	$$O(1)$$ amortizado	Requer índices de hash eficientes no *buffer* para deduplicação rápida.
Ciclo de Esquecimento	$$O(n)$$ com *early stop*	A complexidade O(n × m) deve ser mitigada com otimizações em políticas raramente acionadas.
Sincronização	$$O(\log n)$$	Requer otimizações de

		rede, como filtros de Bloom, para determinar rapidamente quais dados enviar.
Verificação de Prova	$$O(1)$$ com precomputação	A verificação de inclusão Merkle é $$O(\log n)$$, mas é considerada quase $$O(1)$$ na prática devido à profundidade logarítmica.

Conclusão:

A documentação do NOCTURNE define um sistema de estado de última geração, onde o **esquecimento** é um processo ativo e tão importante quanto a **retenção**. A fidelidade à manutenção dos Invariantes I1-I5 é o fator de sucesso para a certificação e conformidade do sistema.