ETERNAL SENTINEL

Zero-Knowledge Cryptographic & Architecture Whitepaper

Technical Specifications, Academic Citations, and State-Machine Formulations

Version: **1.2.0 (Stable Release / Security Hardened)**
Date: **May 2026**
Author: **Eternal Sentinel Engineering & Security Research Group**

ABSTRACT

Eternal Sentinel is a secure digital dead man's switch and digital legacy service. Unlike custodial secrets managers that require trust in central authority, Eternal Sentinel operates on a zero-knowledge architectural boundary. Vault items are encrypted client-side using PBKDF2 and AES-256-GCM prior to network transmission, meaning the server never sees raw passwords, vault payloads, or master keys. This paper provides a thorough mathematical, cryptographic, and operational analysis of the service's zero-knowledge boundary, the background Proof-of-Life state machine, and modern security patches that protect assets from unauthorized trustee access during resurrection and dispute holds. We contextualize these mechanisms using formal specifications from NIST, FIPS, and peer-reviewed cloud storage academic literature.

Table of Contents

1. Executive Summary & Introduction

Digital identity management and automated legacy transition represent a critical operational challenge in modern computing. Standard solutions for post-mortem digital inheritance mandate custodial trust in centralized service providers or require users to expose raw credentials out-of-band, introducing ongoing operational vulnerability. Custodial frameworks store plaintexts or centralized keys, exposing user records to security compromises, regulatory overreach, and insider attacks.

Eternal Sentinel establishes a secure alternative: a client-side, zero-knowledge digital dead man's switch. Operating under the security paradigm that the storage and transport layers represent untrusted environments, all secret records are encrypted in the user's local browser before transmission. By integrating robust background status verification with decentralized, cryptographically-bounded trustee controls, Eternal Sentinel automates inheritance without ever possessing or exposing the user's master cryptographic keys.

Zero-Knowledge Architecture: Defined formally in cloud security literature as a storage model where the host remains mathematically incapable of reading the stored plaintext payload (Kamara & Lauter, 2010). Unlike interactive zero-knowledge proofs (Goldwasser, Micali, & Rackoff, 1985) which demonstrate knowledge of a secret without revealing it, a zero-knowledge storage paradigm uses strict cryptographic boundaries where the decryption keys are generated and held purely in volatile client-side memory, never touching the persistence layer.

2. Threat Model & Zero-Knowledge Paradigm

We define a rigorous threat model to establish the security boundaries of Eternal Sentinel. The model assumes an active and passive adversary (A) capable of full compromise across many infrastructure tiers:

- **Tier 1: Persistence Layer Compromise (SQL Injection / Backup Exposure):**
Adversary A obtains a full dump of the PostgreSQL database. Because all vault payloads are stored as AES-256-GCM ciphertexts with distinct initialization vectors, and master passwords are not stored in any form, A cannot recover the vault items, satisfying ciphertext confidentiality under standard IND-CCA2 assumptions.
- **Tier 2: Compute Host Takeover (Next.js Application Layer Hijack):**
Adversary A gains root access to the running Next.js application server process. Because decryption keys are never transmitted to or processed by the server, A cannot read the vaults of inactive users, as the RAM footprint of the server contains zero plaintexts or key wrapping variables.
- **Tier 3: Active Transport Interception (Man-in-the-Middle - MitM):**
Adversary A intercepts networks between client and server, terminates TLS sessions, or compromises certificates. Since the client-side Web Crypto API performs the encryption using out-of-band derived keys BEFORE payload transit, the intercepted traffic yields only secure ciphertext payloads, preventing data exposure.

3. Zero-Knowledge Cryptographic Specification

The cryptographic engine of Eternal Sentinel operates entirely in the browser using the W3C Web Crypto API. The server acts exclusively as an authenticated metadata index and ciphertext repository.

3.1 Key Derivation Protocol (PBKDF2-HMAC-SHA-256)

To securely derive keys from user passwords, the client leverages the Password-Based Key Derivation Function 2 (PBKDF2) specified in IETF RFC 8018 (Moriarty et al., 2017) and NIST Special Publication 800-132 (NIST, 2010). Given a user-supplied Vault Password (P) that is distinct from their primary login credentials, the derivation proceeds as follows:

```
DK = PBKDF2(HMAC-SHA-256, P, Salt_v, 100000, 256)
```

Key Technical Terms Explained:

• **Salt ($Salt_v$):** A 16-byte (128-bit) cryptographically secure random value generated via `window.crypto.getRandomValues()`. As specified in NIST SP 800-132, the salt ensures uniqueness of derived keys, neutralizing pre-computed dictionary and rainbow table attacks.

• **Stretching and Iterations:** The key stretching iteration count is set to 100,000 rounds of HMAC-SHA-256. This enforces an artificial computational cost, making parallelized hardware brute-force attacks (e.g., GPU/ASIC cluster cracking) economically and computationally infeasible.

• **HMAC-SHA-256 (RFC 2104):** Hash-based Message Authentication Code that combines a cryptographic hash function with a secret key. In PBKDF2, it serves as the pseudo-random function (PRF), guaranteeing output entropy and resistance to length-extension attacks.

3.2 Symmetric Master Key Wrapping Architecture

Directly encrypting vault items with a password-derived key is an architectural risk. A password update would require decrypting and re-encrypting every vault item, causing performance degradation and exposing the master derived key. Eternal Sentinel avoids this by utilizing an intermediate symmetric Master Key (MK) through a key wrapping configuration:

1. **Key Generation:** During vault initialization, the browser generates a 256-bit symmetric Master Key (MK) using cryptographically secure pseudo-random number generation (CSPRNG).
2. **Key Wrapping (AES-KW / AES-GCM):** The derived key (DK) is used to encrypt ($wrap$) the Master Key (MK) using AES-256-GCM. A unique 96-bit Initialization Vector (IV_m) is generated for this step, yielding the Encrypted Master Key (EMK) and an authentication tag (T_m).
3. **Transit Isolation:** The browser transmits $Salt_v$, IV_m, EMK, and T_m to the server. The plaintexts DK and MK are immediately purged from the browser's RAM, ensuring no rest-state key remains.

3.3 Individual Vault Item Protection (AES-256-GCM)

Symmetric vault items are encrypted using the Advanced Encryption Standard (AES) operating in Galois/Counter Mode (GCM), as standardized in NIST Special Publication 800-38D (Dworkin, 2007). AES-GCM is an Authenticated Encryption with Associated Data (AEAD) algorithm that simultaneously guarantees confidentiality and cryptographic integrity.

For every vault item (V_i), the local browser executes the following encryption steps:

1.**Nonce/IV Generation:** Generates a unique 96-bit (12-byte) Initialization Vector (IV_i) using CSPRNG. As detailed by Joux (2006) in comments to NIST, GCM is highly fragile under IV reuse. Reusing an IV with the same key allows an attacker to compute the authentication key (the hash key H), rendering the tag checking useless and exposing plaintexts. Eternal Sentinel ensures IV uniqueness by generating a unique CSPRNG nonce for every single vault item.

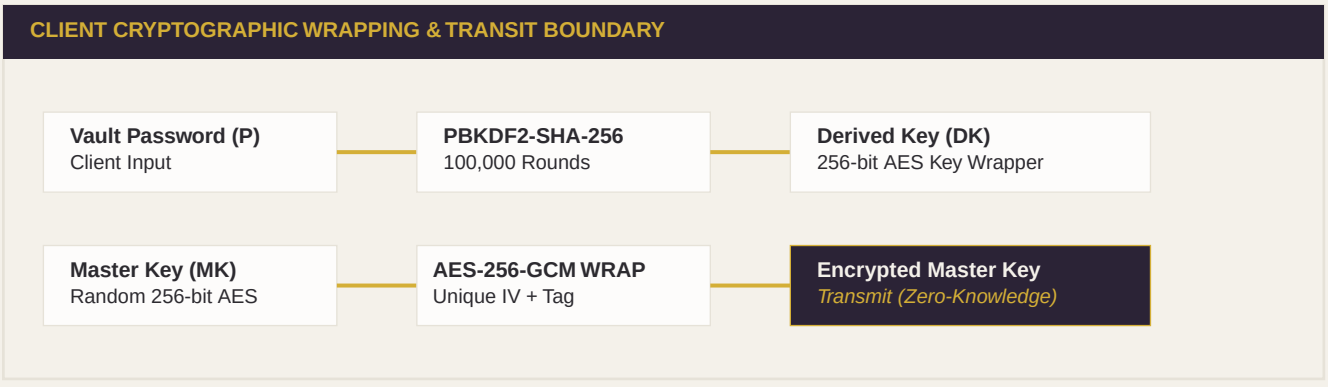
2.**Counter Mode Encryption:** The block cipher operates in Counter Mode to encrypt the plaintext payload, producing the ciphertext (C_i).

3.**Galois Field Multiplication:** Integrates a Galois field multiplier ($GF(2^{128})$) to compute a 128-bit authentication tag (T_i) over the ciphertext and optional Associated Data. This tag protects the ciphertext from bit-flipping and tamper attacks in untrusted database systems.

Ciphertext Structure Sent to Server:

```
Payload = { ciphertext: Base64, iv: Base64, tag: Base64, category: String }
```

The core strength of this multi-tiered cryptographic design lies in the segregation of operations. If a user modifies their vault password (P), only the Encrypted Master Key (EMK) must be decrypted and re-encrypted with the new derived key. The individual item ciphertexts (C_i) remain completely untouched on the database server. This architecture minimizes expensive browser computations and eliminates server-side exposure during critical security updates.



4. Proof-of-Life State Machine

The core of Eternal Sentinel's automated inheritance is a deterministic Finite State Automata (FSA) implemented on the database status variable. This model transitions the system from normal operation into warning phases, emergency verification holds, and eventual estate execution.

4.1 State Descriptions & Transition Algebra

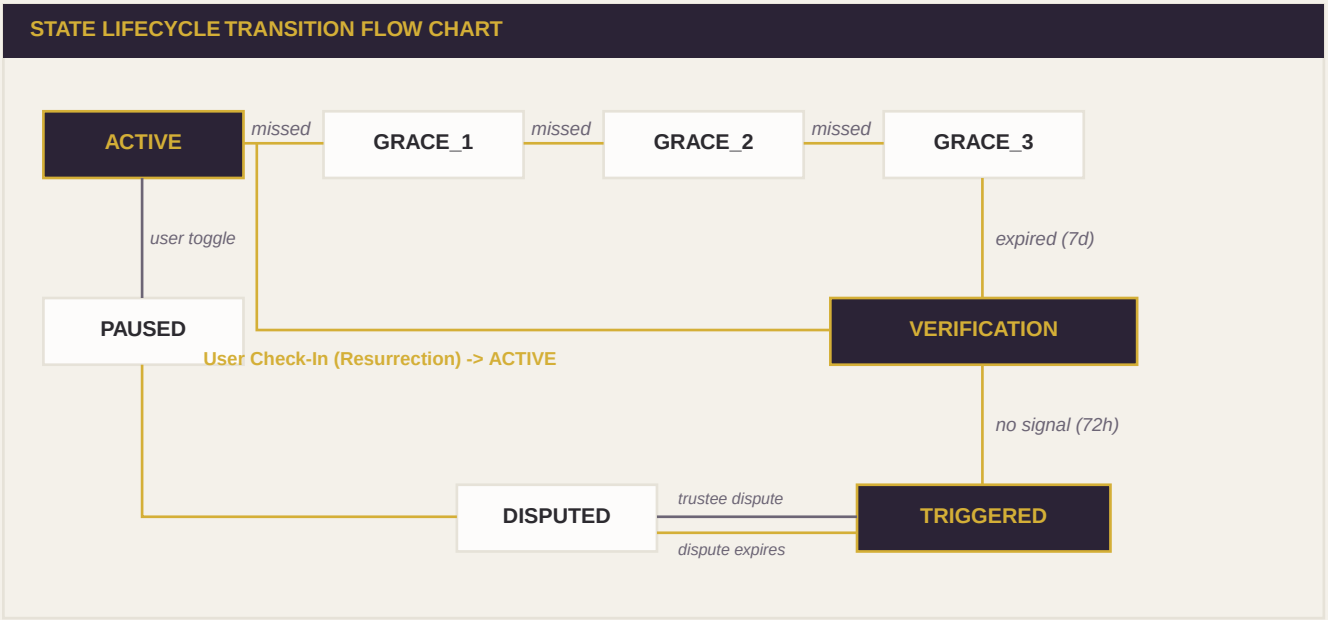
The state machine acts on eight states (S) using state transition algebra triggered by elapsed time ($Delta t$) and check-in confirmation signals (C_u):

- **ACTIVE:** Normal operating state. Check-ins are scheduled at regular intervals (Weekly, Bi-weekly, Monthly) based on the user's subscription profile.
- **PAUSED:** User-initiated vacation mode. The scheduler ignores paused records, suspending check-in requirements until the user manually toggles the state back to ACTIVE.
- **GRACE_1:** Primary warning phase triggered when the user misses a scheduled check-in. The worker sends a warning email containing a unique confirmation token. Duration: 7 days.
- **GRACE_2:** Secondary warning phase triggered upon GRACE_1 expiration. Dispatches a second warning notice via email. Duration: 14 days.
- **GRACE_3:** Final warning phase triggered upon GRACE_2 expiration. The scheduler sends a last-chance warning email. Duration: 7 days.
- **PAUSED_VERIFICATION:** Pre-trigger emergency hold (72 hours). The worker dispatches parallel, high-priority notifications across all communication channels (emails and Twilio SMS). A background scheduler sweeps for expired holds.
- **TRIGGERED:** The death protocol is executed. The system generates secure 48-byte random access tokens for verified trustees, launches the 30-day vault access window, and releases scheduled final letters.
- **DISPUTED:** A trustee asserts the trigger is incorrect, pausing vault access during probate contention. The hold automatically expires, returning the user to TRIGGERED if no check-in is received within 7 days.

> **STATE TRANSITION FORMULAS**
> *Let S be the current status. Transition to GRACE_1 occurs if now > nextCheckInDue and S = ACTIVE. Transitions through grace levels occur at: S_{n+1} = Grace_{n+1} if S_n = Grace_n and Delta t > GracePeriod_n. Any valid check-in confirmation C_u immediately resets the state: S_{next} = ACTIVE for all states except S = TRIGGERED.*

4.2 State Machine Flow Visual Reference

The chart below illustrates the lifecycle transitions of a user account from normal operation through sequential escalations, pre-trigger holds, and eventual estate execution.



4.3 Durable Deadlines: Mitigating In-Memory Timer Vulnerabilities

A major resilience bottleneck in early dead man's switch designs was relying on Node.js in-memory `setTimeout` callbacks inside background workers to execute state transitions. In modern cloud hosting (e.g. AWS ECS Fargate), compute nodes are highly ephemeral—they restart and scale continuously during deployments or under variable load. A 7-day timer held in RAM is almost certainly destroyed before it fires, causing the transition logic to fail.

Security release 1.2.0 resolved this by implementing a durable, database-driven deadline architecture. When a user enters a new state, the worker calculates the exact expiration date and writes it to a persistent database column (`gracePeriodExpiresAt`). The database-backed scheduler sweeps for expired states periodically, ensuring state transitions are resilient across server crashes and restarts.

5. AWS Infrastructure Security & VPC Topography

Security is not just a cryptographic boundary in browser memory; the hosting network topology must also guarantee complete containment. Eternal Sentinel deploys on AWS using a highly isolated Virtual Private Cloud (VPC) that isolates databases and workers from the open web.

5.1 Multi-Tiered Subnet Architecture & Container Security

The VPC is spread across two Availability Zones (AZ A and B) to maintain high availability and is split into public and private network slices:

- **Public Application Load Balancer (ALB):** Exposes port 443 to the public internet, terminating incoming HTTPS traffic using SSL certificates managed by AWS Certificate Manager (ACM). The load balancer handles external traffic, routing requests to Next.js app tasks.
- **Private App Subnet (ECS Fargate Tasks):** Hosts Next.js container tasks (`eternal-sentinel-app`) inside AWS ECS Fargate. These nodes have no public IP addresses and cannot receive unsolicited external connections. All inbound traffic is forced through the Web ALB.
- **Private Worker Subnet (BullMQ Background Worker):** Runs stateless BullMQ worker container tasks (`eternal-sentinel-worker`) in isolation. Like the web tasks, the workers run inside Fargate without public access, connecting internally to Redis and PostgreSQL.
- **Private Storage Subnet (Aurora PostgreSQL & ElastiCache Redis):** Holds our Amazon RDS serverless Aurora PostgreSQL database and Amazon ElastiCache Redis cluster in private isolation, preventing any direct connection attempts from the public internet.
- **NAT Gateways:** Provisioned in the public subnets, NAT Gateways authorize secure, one-way outbound API integration. Web and worker nodes can request external resources (Twilio, Stripe, Resend) without exposing their private IPs to incoming traffic.

> **CONTAINER BOUNDARY & SECRETS ISOLATION**
> *No environment variables or API keys are written to git or stored directly inside the Docker images. All runtime configuration (database strings, Stripe keys, Twilio credentials) is securely stored in AWS Secrets Manager. Fargate tasks read these secrets on startup and inject them directly into memory variables, preventing file-system leakage.*

6. Trustee Protocol, Token Auth, and Security Hardening

The dead man's switch must ultimately deliver vault assets to designated trustees. However, establishing access to sensitive data without a primary user present introduces complex authorization and authentication vectors.

6.1 Multi-Channel Alert Fan-Out & Token Generation

When the pre-trigger 72-hour hold (`PAUSED_VERIFICATION`) expires without a response, the background death protocol worker executes in a single database transaction. The worker:

1. Generates unique 48-byte cryptographically secure random `accessToken` strings for each verified trustee.
2. Configures a strict 30-day expiration window (`accessExpiresAt`) on each trustee record.
3. Sends unique secure links containing these tokens to each trustee via email and backup SMS phone numbers.
4. Locks the user's check-in configurations, setting their active status to `TRIGGERED`.

6.2 PR14 Status Gate: Resurrection & Dispute Hold Hardening

A severe security vulnerability was identified in older iterations of the trustee access route `/api/trustee/access`. The endpoint verified only that the token was valid and hadn't expired. It never checked the owner's `pollingConfig.status`. This created two massive security vulnerabilities:

- **The Resurrection Bypass:** If a user was erroneously marked dead but subsequently checked in to resurrect their account (resetting status to `ACTIVE`), the trustees' 30-day tokens remained active! Trustees could still access the API and download the vault.
- **The Dispute Bypass:** If a trustee opened a dispute (status to `DISPUTED`), the access route did not check this hold, letting trustees fetch the decrypted vault items via the API anyway.

> **REMEDIATION SUMMARY (PR14 STATUS GATE)**
> *In the v1.2.0 security release, both GET and POST pathways in `/api/trustee/access` were refactored to fetch the user's `pollingConfig.status` and evaluate it against a pure policy helper `decideAccess()`. Vault access is now granted strictly when status is TRIGGERED. If the user resurrects, access is instantly revoked. If the status is DISPUTED, the API returns a 403, and all unauthorized access attempts are logged in the database to prevent token probing.*

7. Academic References & Standard Documentation

The security protocols, cryptographic primitives, and architecture patterns implemented within the Eternal Sentinel platform are backed by established academic research and international government standards. We provide formal citations for the methodologies utilized throughout the codebase:

Goldwasser, S., Micali, S., & Rackoff, C. (1985).
The knowledge complexity of interactive proof systems. In Proceedings of the seventeenth annual ACM symposium on Theory of computing (pp. 291-304). *Establishes the mathematical foundation of zero-knowledge systems.*

Kamara, S., & Lauter, K. (2010).
Cryptographic cloud storage. In International Conference on Financial Cryptography and Data Security (pp. 136-149). Springer, Berlin, Heidelberg. *Defines parameters for client-side encryption boundaries in untrusted cloud models.*

Moriarty, K., Kaliski, B., & Rusch, A. (2017).
RFC 8018: PKCS #5: Password-Based Cryptography Specification Version 2.1. Internet Engineering Task Force (IETF). *Specifies the formal implementation guidelines and security considerations for PBKDF2-HMAC.*

Dworkin, M. (2007).
NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. National Institute of Standards and Technology (NIST). *Establishes standards for AEAD modes including IV uniqueness.*

National Institute of Standards and Technology. (2010).
NIST Special Publication 800-132: Recommendation for Password-Based Key Derivation. U.S. Department of Commerce. *Defines iteration boundaries, random salt requirements, and security criteria for derived keys.*

Joux, A. (2006).
Authentication Failures in GCM. National Institute of Standards and Technology (NIST) Comments. *Demonstrates the fatal cryptographic impact of IV/nonce reuse under the AES-GCM cipher mode.*

Kaliski, B. (2000).
RFC 2898: PKCS #5: Password-Based Cryptography Specification Version 2.0. Internet Engineering Task Force (IETF). *Introduces iteration-based key stretching standards.*