Reference Implementation & Software Overview (SecretML v4)

4.1 The Functional Prototype: SecretML v4

The current reference implementation, SecretML v4, serves as a stable, cross-platform proof of concept (optimized for Windows 10+). It features a dedicated graphical user interface (GUI) designed to facilitate secure key generation without compromising the underlying Zero-Knowledge principles.

Core Operational Principles:

- **Zero-Persistence State:** The application operates in a strictly volatile environment. Upon initialization, the internal storage (`AppState`) is empty. No sensitive data is ever cached to a local database or external server.
- **Memory Isolation:** All critical data — questions, answers, and secret payloads — exists only within RAM for the duration of the active session.
- **Active Sanitization:** Once the encryption/decryption cycle is complete, the `reset_user_data()` routine is triggered, purging all cryptographic contexts and user inputs from memory.

4.2 The Key Derivation Pipeline

SecretML v4 utilizes a deterministic pipeline to forge a 256-bit Master Key. The process is designed to be "Memory-Hard," making it resistant to brute-force acceleration.

The Derivation Formula:
The system generates the final key by layering the entropy of user responses with a unique physical "salt" (the hash of a secret file).

$K_{combined} = H(Ans_0 + Salt_{file}) + \sum_{i=1}^{n} H(Ans_i)$
$K_{final} = H(K_{combined})$ Where H represents the SHA-256 hashing function.

Technical Workflow:

1. **Input Normalization:** Stripping semantic noise from user responses.
2. **Entropy Fusion:** Combining cognitive responses with the secret file's unique hash.
3. **Memory-Hard Processing:** Passing the seed through Argon2 to ensure computational cost for attackers.
4. **Final Key Forging:** Generating the 256-bit key for ChaCha20-Poly1305.

4.3 AI-Enhanced Security: "Generate AI Questions"

A unique feature of SecretMemoryLocker is its integration with Large Language Models (LLMs) to solve the "Creative Block" of security questions.

- **Pre-generated Database:** Version 4 includes 1,000+ AI-curated questions, categorized by difficulty, theme, and cognitive model.
- **The AI Edge (Upcoming):** The "Generate AI Questions" feature is currently in development. It will provide a local or secure-online assistant to help users craft highly personal, high-entropy questions that are easy for them to remember but impossible for others to guess.

4.4 The "Phantom-Step" Engine: Cascading Decryption

The reference implementation successfully realizes the Phantom-Step logic. Instead of a single "unlock" event, the system uses a step-wise decryption cascade.

- **Layered Isolation:** Each correct answer acts as the decryption key for the next prompt.
- **Physical Unavailability:** Data for Step 3 does not exist in a readable state until Step 2 is successfully unlocked.
- **Chain Integrity:** Any disruption in the sequence renders the final payload permanently inaccessible, providing a robust defense against forensic memory analysis.

4.5 Cryptographic Primitives & Stack

We rely exclusively on industry-standard, audited libraries for the reference implementation:

COMPONENT	PRIMITIVE / LIBRARY	PURPOSE
KDF	Argon2id	Memory-hard deterministic key generation.
Hashing	SHA-256	Integrity verification and entropy stacking.
Encryption	ChaCha20-Poly1305	Authenticated encryption (AEAD) for payloads.

COMPONENT	PRIMITIVE / LIBRARY	PURPOSE
Interface	Tkinter / ttk	Lightweight, native GUI for Windows environment.

4.6 Prototype Constraints & Hardening Roadmap

As a Reference Implementation, SecretML v4 is intended for security research and concept validation.

Future Development Stages:

- **Memory Hardening:** Moving beyond Python's memory management to **Rust** to prevent side-channel attacks and memory probing.
- **MirageLoop Implementation:** Fully automating the generation of "decoy" questions to misdirect automated brute-force tools.
- **Audit Readiness:** Finalizing the documentation for a full third-party cryptographic audit.

4.7 Hardening Roadmap: Evolution of Entropy Anchors

In the current reference implementation (SecretML v4), the "External Salt" is derived from the SHA-256 hash of a user-selected file. However, the protocol is designed for modular expansion to include advanced physical and biometric anchors.

Future Entropy Sources for the Pipeline:

- **Biometric Vectors:** Integration of FaceID (geometry mapping), TouchID (fingerprint minutiae), and Iris scanning as a non-reproducible 256-bit salt.

- **Acoustic Fingerprinting:** Utilizing the unique frequency characteristics of the user's voice to salt the final derivation stage.
- **Hardware/Digital Identity:** Support for Electronic Signatures (e-ID) and hardware tokens (YubiKey/U2F) to act as the primary anchor for the "Phantom-Step" sequence.

> **Development Status:**
>
> The architecture for these integrations is already present in the `AppState` logic. The next development phase involves implementing secure drivers to bridge local biometric sensors with the SecretML hashing pipeline while maintaining our **Zero-Knowledge stance**.
>
> *Note: Raw biometric data is never stored; only its one-way cryptographic representation is used during the session.*