SCEPTIVE.

Veilith: Perfect Deniable Encryption

Harun ESUR - Sceptive LLC.

V0.1: 1 August 2025

Abstract

In an era of increasing digital surveillance and data breaches, traditional encryption methods often fail to provide plausible deniability under coercion or forensic analysis.

This white paper introduces a novel deniable vault management system, which enables users to store multiple hidden messages within a single file, each accessible via distinct passwords. By leveraging randomized salt headers, scattered encrypted blocks, and device-specific integrity checks, the system ensures that revealing one message does not compromise others.

Built on battle-tested cryptographic primitives including Argon2id for key derivation and XChaCha20-Poly1305 for authenticated encryption, this approach addresses key challenges in secure data storage. We discuss the underlying methods, algorithms, benefits, potential drawbacks, and avenues for further research.

Introduction

Digital privacy faces unprecedented threats from state actors, cybercriminals, and even physical coercion scenarios where individuals may be forced to disclose encryption keys. Standard encryption schemes, such as AES, protect data confidentiality but offer no mechanism for deniability—once a key is revealed, the entire contents are exposed. Deniable encryption emerges as a countermeasure, allowing users to maintain "decoy" data layers that can be plausibly presented as the true contents while concealing sensitive information.

Veilith advances this concept by creating fixed-size files with multiple independent encryption layers. Each layer corresponds to a (password, message) pair, embedded in a structure that mimics random data. Additionally, the system incorporates device integrity verification to detect unauthorized modifications or transfers across devices. This white paper elucidates the methods employed, the cryptographic algorithms underpinning them, the problems solved, benefits, and future research directions.

Problem statement

Conventional data encryption suffers from several limitations:

1. Lack of Plausible Deniability: Under duress, users may be compelled to reveal passwords, exposing all data. There is no way to hide the existence of additional sensitive information without arousing suspicion.
2. Device Portability Risks: Encrypted files can be copied and modified on unauthorized devices, potentially leading to tampering or key extraction without detection.
3. Forensic Vulnerabilities: File structures often reveal metadata (e.g., number of encrypted entries) through size variations or patterns, enabling side-channel attacks like timing analysis during decryption.
4. Scalability for Multiple Secrets: Storing multiple independent secrets in one file typically requires complex partitioning, increasing the risk of detection if not randomized properly.
5. Performance and Security Trade-offs: Mobile environments demand lightweight yet secure key derivation and encryption, resistant to brute-force and rainbow table attacks.

These issues are particularly acute in scenarios involving journalists, activists, or professionals handling sensitive data in hostile environments.

Proposed Solution

Veilith introduces a multi-layer deniable encryption scheme where a single file can contain up to several independent messages (limited by configuration parameters like 64 salts and blocks), each tied to a unique password. The file appears as a monolithic blob of random data, with real encrypted content scattered among decoys.

Key features include;

- Fixed-Structure Randomization: Always generates exactly 64 salts and 64 blocks (each 8192 bytes), padding with random data to obscure the number of real entries.
- Device Binding: An HMAC computed over the file contents using a device-specific key detects if the file was created or modified elsewhere.
- Exhaustive Decryption Search: To mitigate timing attacks, decryption always iterates over all salts and blocks, regardless of success.

- **Update Mechanism:** Allows modification of specific entries while preserving deniability, with optional override for device changes.

The system supports creation, decryption, updating, and recalculating integrity for imported files, making it suitable for mobile apps with limited resources.

Technical Details

File Structure

The encrypted file follows a rigid format to enhance deniability:

- **Device Integrity HMAC (32 bytes)**: Computed over the remaining file content using a device-derived key. Ensures the file hasn't been altered on another device.
- **Salt Header (64 × 16 = 1024 bytes)**: Contains exactly 64 salts. Real salts (one per entry) are shuffled among random ones.
- **Encrypted Blocks (64 × 8192 = 524,288 bytes)**: Fixed-size blocks where real encrypted messages are placed at random indices; others filled with random data.

Total minimum file size: 32 + 1024 + 524,288 = 525,344 bytes. Messages are limited to ~8164 bytes (block size minus nonce and tag).

Encryption Process

1. **Key Derivation**: For each entry, generate a 16-byte salt. Derive a 32-byte encryption key using Argon2id from the password and salt.
2. **Message Encryption**: Encrypt the message with XChaCha20-Poly1305, appending a 24-byte nonce and 16-byte authentication tag. Pad to block size with zeros.
3. **Placement**: Shuffle salts and place encrypted blocks at random block indices.
4. **Padding**: Fill unused salts and blocks with cryptographically secure random data.
5. **Integrity Seal**: Compute device HMAC over salts + blocks.

Decryption Process

1. **Integrity Check**: Verify device HMAC; if failed, return invalid unless ignored.

2. **Salt Extraction**: Read all 64 salts.
3. **Exhaustive Search**: For each salt, and derive key and attempt decryption on every block. Return the first successful message -or indicate failure- after all attempts).

This brute-force approach prevents timing-based inference on which salt/block is real.

Update Process

1. **Verify Integrity**: Check device HMAC; allow override if changing devices.
2. **Locate Entry**: Use provided salt and block indices (from prior decryption).
3. **Re-encrypt**: Generate new salt, derive new key, encrypt new message.
4. **Update File**: Replace old salt and block; recompute device HMAC.

Device Integrity

Relies on device specific data to provide a persistent device key as it can be various hardware serial data or OS provided unique identifier. HMAC-SHA256 is used for verification, binding the file to the originating device.

Algorithms Used

The system leverages the cryptographic set of standards for secure, efficient primitives:

- **Key Derivation**: Argon2id (interactive parameters: OpsLimitInteractive, MemLimitInteractive). Memory-hard to resist GPU/ASIC brute-force.
- **Authenticated Encryption**: XChaCha20-Poly1305. Stream cipher with 192-bit nonce for collision resistance; provides confidentiality, integrity, and authentication.
- **HMAC**: HMAC-SHA256 for device integrity and (implicitly) in Poly1305.
- **Random Number Generation**: Cryptographically secure RNGs for salts, nonces, and block indices.
- **Secure Zeroing**: OS internals to clear sensitive memory (passwords, keys).

These choices ensure post-quantum resistance in key derivation (Argon2) and high-speed encryption suitable for mobile devices.

Benefits and Advantages

- **Plausible Deniability**: Users can reveal a "decoy" password/message while denying knowledge of others. The fixed structure and randomization make it impossible to prove additional layers exist without all passwords.
- **Coercion Resistance**: Ideal for high-risk users; exhaustive search masks which entry is being accessed.
- **Tamper Detection**: Device HMAC prevents undetected modifications or cross-device tampering.
- **Efficiency**: Fixed sizes enable constant-time operations; Argon2id balances security and performance.
- **Flexibility**: Supports multiple entries (up to ~64, limited by salts/blocks), updates, and integrity recalculation for imports.

Compared to alternatives like TrueCrypt's hidden volumes, this system offers finer-grained layers and device binding without relying on filesystem tricks.

Potential Drawbacks

- **Fixed Overhead**: Always produces ~525KB files, inefficient for tiny messages.
- **Performance Impact**: Exhaustive decryption (64 salts × 64 blocks = 4096 attempts) may be slow on low-end devices, though each attempt is fast (~milliseconds total).
- **Limited Layers**: Capped at 64 entries; scaling requires larger configurations, increasing file size.
- **Device Dependency**: Integrity checks hinder legitimate multi-device use unless overridden, risking security.
- **No Forward Secrecy**: Updates reuse the same file structure; compromised past keys could expose history if not rotated properly.

Future Research Topics

- **Quantum-Resistant Enhancements**: Integrate post-quantum KDFs (e.g., Kyber) or hybrid schemes to future-proof against quantum attacks.
- **Adaptive Layering**: Dynamically adjust salt/block counts based on message count, while preserving deniability through steganographic techniques.
- **Mobile Optimizations**: Benchmark and optimize for battery-constrained devices; explore hardware-accelerated Argon2 variants.
- **Forensic Analysis Resistance**: Study side-channel defenses (e.g., power consumption during decryption) and integrate oblivious RAM for memory access patterns.
- **Multi-Device Synchronization**: Develop secure protocols for transferring files across devices while maintaining integrity, perhaps using threshold cryptography.
- **Integration with Broader Ecosystems**: Extend to cloud storage with deniable proofs of possession or combine with homomorphic encryption for searchable deniable data.

Conclusion

Veilith represents a practical advancement in deniable encryption, solving critical problems in privacy-preserving data storage. By combining randomized structures, robust algorithms, and device-bound integrity, it empowers users to protect sensitive information under adversarial conditions. While not without trade-offs, its benefits in deniability and security make it a valuable tool for modern digital defenses. Future iterations could address scalability and multi-device challenges, paving the way for widespread adoption in privacy-focused applications.