Security Architecture Overview

Zero-Knowledge File Transfer System

Developed by: Goutam Adityan Madan Kumar

January 14, 2026

1 Executive Summary

This document outlines the cryptographic architecture of the Secure File Transfer System. The system is designed as a **Zero-Knowledge Architecture**, ensuring that the server stores encrypted data but never possesses the decryption keys. Confidentiality and integrity are enforced using industry-standard primitives (AES-GCM) and robust key derivation functions (PBKDF2).

2 Cryptographic Primitives

The system relies on two primary cryptographic algorithms to ensure data security at rest.

2.1 Encryption: AES-256-GCM

We utilize the **Advanced Encryption Standard (AES)** with a 256-bit key length operating in **Galois/Counter Mode (GCM)**.

- **Why AES?** It is the NIST standard for symmetric encryption.
- **Why GCM Mode?** Unlike ECB (insecure) or CBC (malleable), GCM provides **Authenticated Encryption (AE)**. It ensures:
  1. **Confidentiality:** The data is unreadable without the key.
  2. **Integrity:** Any tampering with the encrypted file (bit-flipping) invalidates the authentication tag, causing decryption to fail safely.

2.2 Key Derivation: PBKDF2-HMAC-SHA256

User passwords are never used directly as encryption keys due to low entropy. Instead, we derive a cryptographically strong 32-byte key using **PBKDF2** (Password-Based Key Derivation Function 2).

$$K = PBKDF2(HMAC\text{-}SHA256, Password, Salt, Iterations, dkLen) \tag{1}$$

- **Iterations:** Set to $100,000$ to increase the computational work factor, mitigating brute-force and dictionary attacks.
- **Salt:** A unique 16-byte random value for every file, preventing Rainbow Table attacks.

3 Key Management Strategy

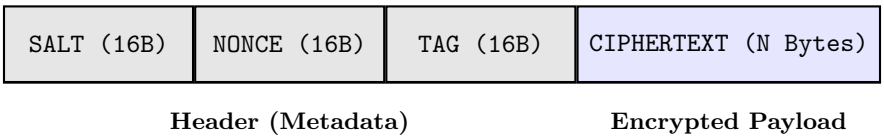
The system implements an **Ephemeral Key Strategy**.

1. **Generation:** The encryption key K is derived in RAM only when the user provides the password.
2. **Usage:** The key is used immediately to encrypt or decrypt the data stream.
3. **Destruction:** Once the operation is complete, the key is discarded from memory. It is **never** written to the database or the file system.

This ensures that even if the server is fully compromised, the attacker obtains only encrypted blobs (ciphertext) and cannot decrypt them without the users' passwords.

4 Data Storage Format

To maintain state for decryption, specific cryptographic artifacts are stored alongside the ciphertext. These are packed into a single binary container file ($filename.enc$).



- **Salt (Bytes 0-15):** Required to recreate the Key from the Password.
- **Nonce (Bytes 16-31):** The initialization vector for AES-GCM. Unique per file.
- **Tag (Bytes 32-47):** The integrity signature. If this check fails, decryption is aborted (Preventing tampering).
- **Ciphertext (Bytes 48+):** The actual encrypted file data.

5 Threat Model Analysis

Threat Vector	Mitigation Strategy	Status
Server Compromise (HDD Theft)	Data is encrypted at rest. Keys are not stored.	**Protected**
Database Leak	No sensitive data (passwords/keys) in DB.	**Protected**
Man-in-the-Middle (MITM)	Use of HTTPS/TLS (Production Requirement).	*Out of Scope*
User Weak Password	PBKDF2 with 100k iterations (Key Stretching).	**Mitigated**
File Tampering	AES-GCM Integrity Tag verification.	**Protected**

6 Conclusion

The system successfully implements a robust security model suitable for high-sensitivity environments. By decoupling the encryption keys from the storage layer, we achieve a high degree of privacy and resilience against data breaches.