Design of a Hybrid AES–RSA Encryption System for Secure Transfer of Large Files

Prepared by: **Mohammad Hafizur Rahman Sakib**

Course: Network and Computer Security (CSE 437)

Spring 2025

September 29, 2025

Problem Summary

We must design an encryption system that:

- Encrypts large files efficiently (AES for bulk data).
- Exchanges AES session keys securely (RSA for key transport).
- Ensures confidentiality, integrity, and authenticity.
- Handles very large files (chunking, streaming).
- Defends against man-in-the-middle, replay, and tampering.

High-level Design Overview

Core idea: Use an **AEAD** symmetric cipher (AES-GCM or AES-GCM-SIV) with a randomly generated 256-bit session key to encrypt file data. Use RSA with OAEP padding (e.g., RSA-OAEP with SHA-256) to encrypt the session key for transport. Optionally sign the AES key (or the message digest) with the sender's RSA private key to provide non-repudiation and authenticity.

Recommended primitives and parameters

- **Symmetric:** AES-256 in GCM mode (AES-256-GCM) for authenticated encryption (confidentiality + integrity).

- **Asymmetric key exchange:** RSA-3072 or RSA-4096 for long-term security; use RSA-OAEP (SHA-256) for encryption of the session key.
- **Signatures (authenticity, optional):** RSA-PSS with SHA-256 or ECDSA with curve P-384 (or Ed25519) depending on environment.
- **Hashing/HMAC:** SHA-256; HMAC-SHA256 for any keyed integrity chains when needed.
- **KDF:** HKDF-SHA256 when deriving multiple keys from a single seed (IV salts, per-chunk keys).

System Architecture

Step-by-step Flow

1. **Key generation (sender):** Create a cryptographically secure random 256-bit session key K and a unique random nonce/IV per file (or per chunk) IV_i.
2. **File chunking:** Split the file into chunks $C_1, \ldots, C_n$ (e.g., 1–10 MiB each) to support streaming and retransmission.
3. **Encrypt chunks:** For each chunk C_i, compute $E_i = \text{AES\_256\_GCM\_Encrypt}(K, IV_i, C_i)$ producing ciphertext and an authentication tag T_i.
4. **Create integrity over file:** Compute a file-level HMAC or a Merkle tree root using SHA-256: $H = \text{HMAC\_SHA256}(K_{auth}, \text{file\_metadata})$ or use the concatenation of chunk tags into a signed manifest.
5. **Encrypt session key:** Encrypt K using receiver's RSA public key with OAEP: $S = \text{RSA\_OAEP\_Encrypt}(RSA_{pub}, K)$.
6. **Package:** Send {RSA-blob S, per-chunk tuples (IV_i, E_i, T_i), manifest (file size, sequence numbers), optional signature}.
7. **Receiver:** Decrypt S with its RSA private key to recover K, then verify and decrypt chunks using AES-GCM and verify tags. Finally validate the file-level integrity (HMAC or manifest).

Chunking, Transfer and Integrity

Chunking strategy

- Choose chunk size to balance latency and memory: typical values 1 MB–10 MB.

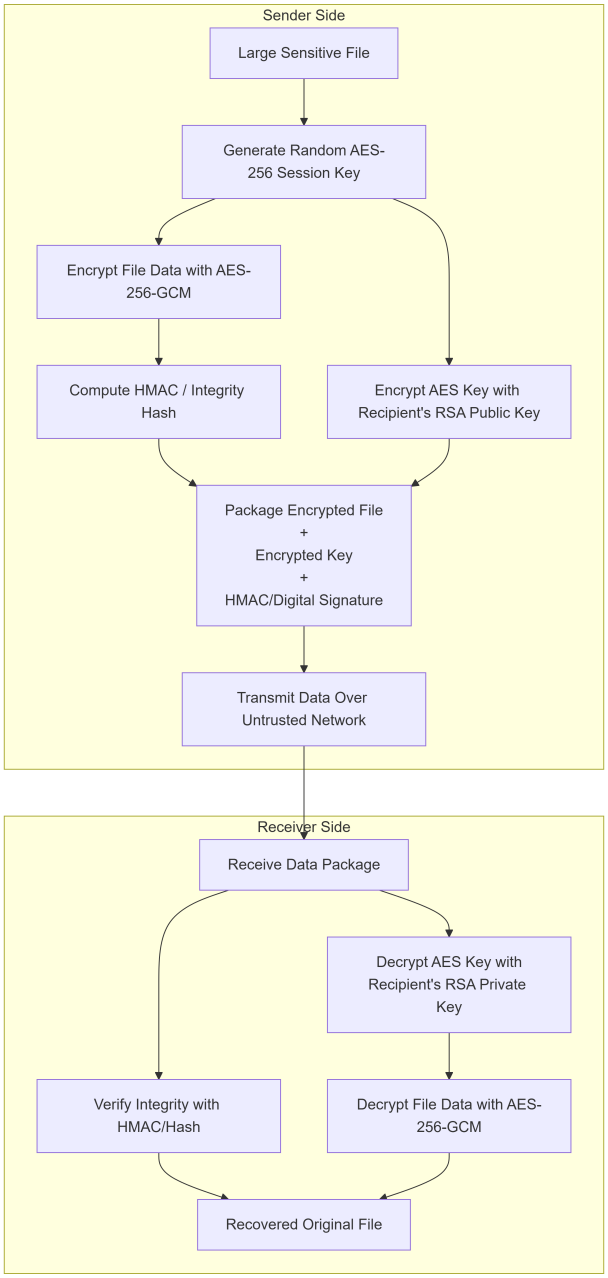

Figure 1: System architecture: hybrid AES (bulk) + RSA (key transport).

- Each chunk uses a unique IV (recommendation: 96-bit nonce for AES-GCM).
- Chunks allow retransmission of failed pieces without resending whole file.
- Use sequence numbers and a manifest file containing chunk count and hashes.

Integrity verification

- Use AES-GCM's authentication tag per chunk. If any chunk's tag fails, discard the chunk and request retransmission.
- For whole-file assurance, compute a signed manifest or a Merkle root over chunk hashes. Optionally sign the manifest with the sender's private key (RSA-PSS).
- Alternative: use HMAC-SHA256 over the entire file or metadata using a derived authentication key.

Flowchart (detailed implementation)

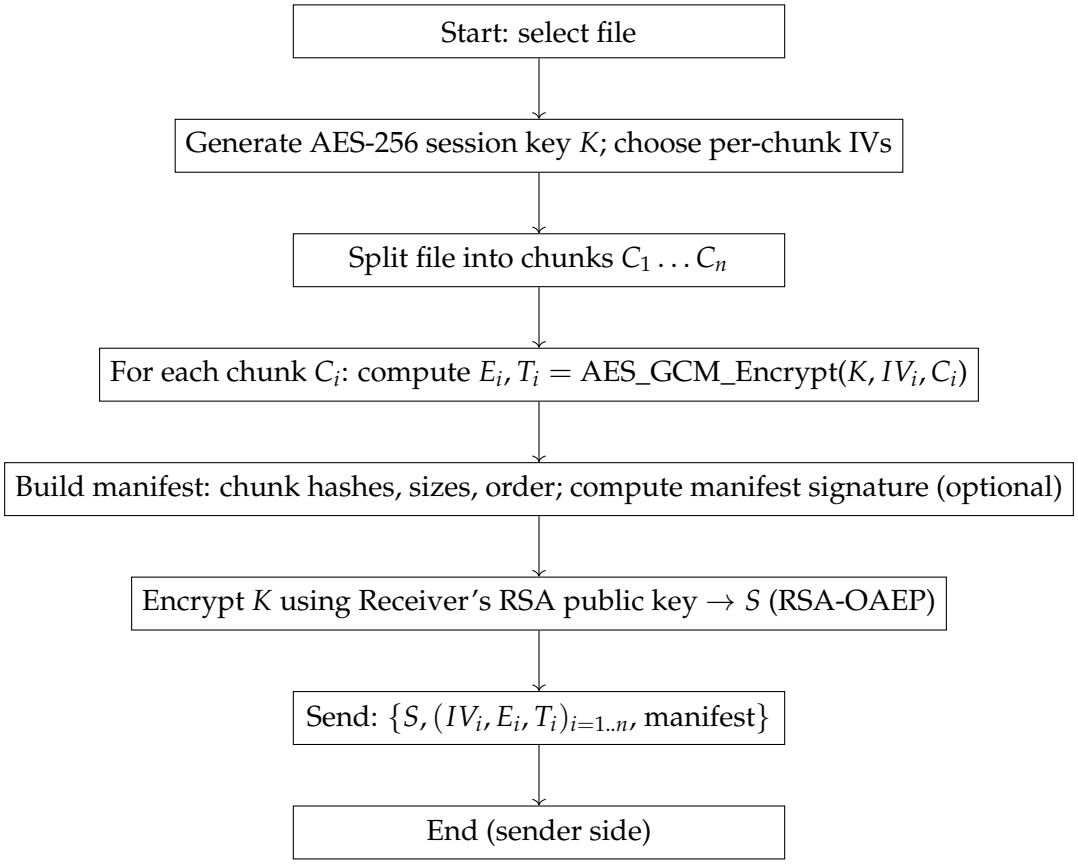

Figure 2: Sender detailed flow for encryption and packaging.

Security Analysis

We explicitly address the threats requested.

Confidentiality

- Bulk data confidentiality is provided by AES-256-GCM using a fresh random key K and unique IVs. AES-256 is currently considered secure for long-term confidentiality.
- Session key confidentiality during transport is ensured by RSA-OAEP encryption using the receiver's public key.

Integrity and Authenticity

- Integrity per chunk is provided by AES-GCM authentication tags (T_i). Any modification during transport will cause tag verification failure.
- Authenticity can be provided by signing the manifest (or the session-key blob) with the sender's private key (RSA-PSS). That prevents an attacker from forging sender identity.
- Optionally, use a public-key infrastructure (PKI) or certificate pinning for verifying public keys and preventing key substitution.

Man-in-the-Middle (MitM) protection

- If the receiver's RSA public key is known and verified (via certificate/PKI or public fingerprint out-of-band), MitM cannot substitute the key — MitM cannot decrypt S.
- To defeat MitM on authenticity, sign the manifest or the hash of the session key using the sender's private key. The receiver verifies the signature.

Replay protection

- Include timestamps and nonces in the manifest; reject manifests older than an acceptable window.
- Use sequence numbers per chunk and session IDs so old messages cannot be replayed as part of a new session.

Brute-force and cryptanalysis

- AES-256 resists practical brute force. RSA-3072 or greater gives sufficient security margin for decades.
- Use proper randomness (CSPRNG) for keys and IVs; avoid ECB mode and never reuse nonces with AES-GCM for the same key.

Performance, Trade-offs and Compatibility

Why hybrid AES–RSA is better than AES-only or RSA-only

- **AES-only (symmetric only):** extremely fast for bulk data; however, requires both parties to share a pre-existing symmetric key securely (out-of-band). Key distribution for many parties becomes unmanageable.
- **RSA-only (asymmetric only):** can encrypt small messages and keys but is orders of magnitude slower and impractical for multi-megabyte files (and limited by modulus size). RSA encryption of large data also inflates size and is inefficient.
- **Hybrid (AES+RSA):** combines the speed of AES for large data and the convenience of RSA for secure key exchange. This is the standard best practice (used in TLS, S/MIME, etc.).

Key size trade-offs (security vs performance)

- **AES key length:** AES-128 faster and adequate for many uses; AES-256 provides additional security margin with a small performance cost; choose AES-256 for high-value data.
- **RSA key length:** RSA-2048 is commonly used but has less margin for long-term confidentiality; RSA-3072/4096 are stronger but slower. RSA operations are performed on small data (the session key), so performance hit is limited to key transport events.
- For environments with constrained CPU and many connections, consider using EC-based KEMs (e.g., X25519) for key exchange — faster and much smaller keys.

Scaling for very large files

- Use chunking and parallel transfers to utilize multi-threaded upload/download.
- Use streaming AES-GCM (or AES-GCM-SIV) to avoid loading entire files into memory.
- For very large datasets, consider resumable uploads with per-chunk acknowledgements and retry logic.

Answers to the Assignment Questions

1. **How does your system ensure both confidentiality and authenticity?**
   Confidentiality: AES-256-GCM encrypts bulk data. RSA-OAEP encrypts the session

key.

Authenticity: AES-GCM provides integrity for each chunk; signing the manifest (RSA-PSS) or the session-key blob provides sender authenticity/non-repudiation.

2. **What trade-offs between security level (key size) and performance?**
   Larger RSA keys increase CPU and latency for key operations; prefer RSA-3072 for long-term security but accept small performance hit since RSA is used only for the session key. AES-256 slightly slower than AES-128 but increases security margin with negligible bulk throughput impact on modern hardware (including AES-NI).
3. **How does your design handle large files efficiently?**
   Chunking, streaming encryption/decryption, per-chunk IVs and tags, resumable transfers, parallel upload of chunks, and verifying per-chunk tags so only failed chunks are retransmitted.
4. **How is your system secure against replay or tampering attacks?**
   Tampering is detected by AES-GCM tags; replay is prevented using session IDs, sequence numbers, manifest timestamps, and by signing the manifest (so a replayed manifest would be rejected if it's outside allowed time-window or has mismatched session identifiers).
5. **Does your design align with modern cryptographic standards?**
   Yes — recommended primitives are AES-GCM, RSA-OAEP, RSA-PSS, SHA-256, HKDF; these follow modern best-practices. For even stronger modernity and performance, consider using ephemeral ECDH (X25519) for key exchange instead of RSA.

Comparative Analysis Table

Approach	Confidentiality	Integrity	Scalability
AES-only	High (with secure key)	High (AEAD)	Poor (k
RSA-only	Medium	Low (unless using hybrid integrity schemes)	Good
Hybrid AES+RSA	High	High	

Table 1: Practical comparison summary.