Designing a One-Way Symmetric Re-Encryption Architecture for Distributed DRM

I wanted to design a cryptographic architecture where I can **encrypt large AI model files only once**, but where **each authorized recipient receives a unique decryption capability**—and critically, **the CEK (content encryption key) used by the producer must never be revealed, nor equal to the final key that the recipient uses**. At the same time, I require **symmetric-grade performance** for bulk data (ChaCha20, AES-GCM), and I need secure delegation between nodes, proxies, TEEs, or distributed KMS components.

This requirement sits somewhere between **DRM, proxy re-encryption, envelope encryption, trusted execution environments, broadcast encryption**, and **key-homomorphic constructions**. To reason this through properly, I walked through every cryptographic family that could plausibly satisfy these constraints.

---

1. What I Actually Need

I formalized my requirements like this:

1. **Bulk encryption** must happen exactly once, using symmetric AEAD.
2. Every recipient must have a **unique decryption route**, auditable and revocable.
3. The producer's **CEK must never be exposed**, and cannot equal the final per-recipient key.
4. The asymmetric or PRE layer must work only on **small artifacts** (capsules, wrapped CEKs), not the large model file.
5. Decryption should be **local or remote**, but if local, the cryptography must recognize the fundamental limitation that **local plaintext extraction implies possession of decryption capability**.

That last point turns out to be absolutely critical.

---

2. The Fundamental Cryptographic Constraint

A deep and unavoidable fact emerged during analysis:

> **If a recipient decrypts locally (on their own machine, outside a trusted hardware boundary), then cryptographically they must possess a decryption key equivalent to CEK.**

There is no cryptographic trick that allows a user to decrypt data locally while preventing them from learning the plaintext or an equivalent of the key. This is the core reason commercial DRM systems rely on trusted hardware players, TEEs, HSMs, secure elements, and watermarking.

Therefore, any scheme that keeps CEK completely secret from recipients must use **server-side, enclave-based, or proxy-assisted decryption**.

With that reality established, I surveyed all cryptographic approaches that could still meet the desired semantics.

---

3. Exploring All Possible Cryptographic Families

I explored the entire landscape: KEM/DEM envelope encryption, HPKE, proxy re-encryption, threshold PRE (Umbral), multi-hop PRE, attribute-based encryption, broadcast encryption, TEEs with attestation, MPC-assisted decryption, and even key-homomorphic symmetric constructions.

Below is everything I analyzed and how each fits (or fails) the requirement **"CEK must never be exposed and must not equal the final per-recipient decryption key."**

3.1 Standard Envelope Encryption (KEM + DEM)

The typical hybrid approach: - Encrypt data with CEK. - For each recipient, use HPKE/KEM to wrap the CEK.

This gives per-recipient unique encapsulations, but: - **Recipient locally obtains CEK or a CEK-equivalent**, violating my requirement.

Still useful, but not what I ultimately want.

3.2 Proxy Re-Encryption (PRE)

With PRE (e.g., Umbral, AFGH, BBS98, BLS12-381-based): - Producer encrypts CEK once into a capsule. - Proxies re-encrypt capsule for recipients. - Recipient gets CEK (or CEK-derivative) and decrypts the blob.

Advantages: - Encrypt once. - Delegation. - Threshold security.

But again: - **Recipient obtains CEK-equivalent**, so CEK isn't truly kept secret.

PRE is great, but does not meet the strongest requirement unless the final decryption occurs in a trusted environment.

3.3 Key-Translation / Key-Homomorphic Symmetric Schemes

I explored whether I could construct a scheme where:

```
Producer encrypts with CEK.
Recipient decrypts with DK_r.
DK_r ≠ CEK.
```

And DK_r decrypts the same ciphertext.

This requires a **key-homomorphic PRF or key-homomorphic stream cipher layer**. In theory, I could use structured PRFs or ChaCha20 keystream manipulations under a safe construction layer.

This is elegant, but must be designed with extreme care to avoid correlations, keystream leakage, or chosen-key attacks.

3.4 Broadcast Encryption / Subset-Cover Schemes

These are used in large-scale DRM (cable TV, Blu-ray). They allow: - One ciphertext for all recipients. - Per-user secret keys.

But: - Local decryption always gives the effective CEK. - Revocation complexity is high.

3.5 Attribute-Based Encryption (ABE)

ABE allows policy-based encryption of CEK. But: - Still decrypts locally → CEK-equivalent exposed. - Heavy, slow, large ciphertexts.

3.6 Threshold MPC / Server-Assisted Decryption

Here, recipients do not decrypt locally. Instead: - Multiple servers combine shares to decrypt. - CEK never leaves servers.

This **does** preserve CEK secrecy.

3.7 Trusted Execution Environments (TEEs)

This is the first solution that **truly preserves CEK secrecy** while allowing recipients to access plaintext.

In this pattern: - CEK stays inside enclave/KMS/HSM. - Recipient authenticates. - Enclave decrypts ciphertext and streams plaintext. - Optionally, the enclave re-encrypts with per-recipient ephemeral keys.

This satisfies every requirement: - Encrypt once. - CEK never exposed. - Per-recipient keys. - High-performance symmetric decryption.

This is the **standard DRM-grade answer**.

---

4. Deriving the Final Two Possible Architectures

After surveying all options, I realized that only two practical architectures fully match my needs.

4.1 Architecture A — TEE / Server-Assisted Decryption (Recommended)

This is the only construction that satisfies: - **Encrypt once**. - **Unique per-recipient decryption path**. - **CEK never revealed**. - **Symmetric performance**. - **Multi-hop or distributed delegation** (via PRE).

Flow: 1. Producer encrypts content with CEK once. 2. CEK stored only inside TEE/KMS/HSM. 3. Recipients authenticate and receive either: - A plaintext stream, - Or a per-recipient re-encrypted stream key from inside the enclave. 4. PRE can be layered to delegate access without exposing CEK to proxies.

This is exactly how modern DRM for video, game assets, and premium ML models is built.

4.2 Architecture B — PRE + Local Decryption (If CEK exposure is acceptable)

If I relax the "CEK never exposed" condition to: - CEK may be exposed to the recipient, but delegation must be dynamic,

then PRE is perfect: - Producer encrypts CEK once. - PRE proxies re-encrypt capsules for each user. - Recipients get CEK-equivalent and decrypt locally.

This is ideal when redistribution prevention is not cryptographically enforced.

---

5. Practical DRM Architecture I Would Build

Ultimately, the architecture that matches everything I want is a **hybrid**:

1. **Encrypt once with CEK** (ChaCha20-Poly1305).
2. **Store CEK inside a TEE/KMS/HSM**.

3. **Use PRE (Umbral, BLS12-381)** to delegate access without the TEE directly managing all recipients.
4. **Perform final decryption or per-recipient re-encryption inside the TEE**, not on the recipient's device.
5. Use **remote attestation** so the recipient device proves it is running trusted decryption software.
6. Add **watermarking** or **audit logs** on PRE transformations.

This gives: - CEK secrecy - Unique keys/tokens per user - Multi-hop trust - Symmetric speed for data - Real DRM security - Revocation & auditing

---

6. Final Summary

From all cryptographic possibilities, I arrived at two solid answers:

A. If CEK must never be exposed (strict requirement):

- Use **TEE/server-assisted decryption**, with PRE for delegation.
- Encrypt once; re-encrypt or stream inside trusted hardware.

B. If recipients can gain CEK-equivalent (weaker requirement):

- Use **PRE (Umbral)** to delegate CEK access.
- Encrypt once; recipients decapsulate and decrypt.

Considering all constraints—performance, DRM integrity, multi-hop delegation, and CEK secrecy—the **TEE + PRE hybrid architecture** is the most powerful and practical solution.

It gives me exactly what I set out to build: **encrypt once, decrypt with unique per-recipient keys, without ever exposing the producer's CEK, all at symmetric performance levels**.