GKP Format Specification (Gankpo Cryptographic Container)

1. Description Technique

Le format GKP (Gankpo Cryptographic Container) est un conteneur d'archives cryptographiques natif, conçu pour le scellement, la traçabilité multi-signature et le partage sécurisé de documents numériques avec une garantie d'intégrité sans faille. GKP encapsule à la fois la charge utile (le document original), les métadonnées de l'archive, les signatures cryptographiques de la gouvernance et une enveloppe de partage chiffrée (si configurée).

L'intégralité du conteneur est structurée, sérialisée et compressée via **CBOR** (Concise Binary Object Representation), assurant une taille minimale et une analyse rapide sur des environnements contraints ou asynchrones. Le registre de preuves est scellé via **Blake3**, offrant avec une sécurité cryptographique de niveau militaire (Zeroize, ed25519, AES-256-GCM).

2. Architecture du Conteneur

Un fichier GKP classique est subdivisé en trois grandes sections cryptographiques et informationnelles, le tout sérialisé dans un objet pivot :

```
graph TD
    GKP[Fichier GKP Root Object]
    GKP --> H[Header]
    GKP --> C[Content Envelope]
    GKP --> M[Multisig Ledger]
    H --> Manifest[Metadonnées, UID, Horodatage]
    H --> HASH[hash_payload, hash_full]
    C --> CY[Ciphertext AES-GCM]
    C --> NONCE[Nonce XChaCha20]
    C --> SALT[Password Salt]
    M --> P[Signature Policy]
    M --> R[Records / Actes cryptographiques]
```

3. Composants et Structure des Données

La sérialisation interne stricte suit le schéma de données Rust :

3.1. GkpHeader

Le `Header` contient les identifiants racines de l'archive.

- **`version`** : Iteration du protocole ("4.0").
- **`gkp_ref`** : Identifiant UUID ou format Gankpo unique.
- **`algorithms`** : Suite de chiffrement déclarée (ex: "Ed25519+Blake3").
- **`hash_payload`** : Le hash Blake3 direct de la donnée brute.

- **`hash_full`** : L'empreinte holistique finale de l'ensemble de la structure, incluant le registre multisig et l'en-tête (utiliée pour détecter l'altération globale).

3.2. Content Envelope (Charge Utile)

Le contenu brut n'est **jamais** exposé directement dans les modes sécurisés. Il est encapsulé via :

- `crypto_mode` : `clear`, `symmetric_password` (copie privée), `asymmetric_identity` (partage d'enclave).
- `salt`, `nonce` : Vecteurs d'initialisation pour le chiffrement symétrique.
- `ciphertext` : Fichier compressé originel chiffré.

3.3. Multisig Ledger (Gouvernance)

Gère l'historique inaltérable de l'enclave.

- **`policy`** : Définit les règles d'approbation (`mode` : Ordered, Threshold, Allowlist) et les `expected_roles`.
- **`records`** : Liste chronologique stricte des signatures apposées (Actes). Chaque record chaîne cryptographiquement le hash du précédent.

4. Flux de Validation Cryptographique

1. **Désérialisation** : Le fichier `.gkp` est parsé du format CBOR vers un graphe en mémoire.
2. **Vérification de l'Enveloppe** : Le module cryptographique s'assure que le `hash_payload` encapsulé correspond à l'extraction de la donnée.
3. **Vérification de la Chaîne Multisig** :
   - Le *Genesis Record* (Création) est évalué.
   - Chaque signature subséquente est vérifiée sur la courbe elliptique `Ed25519`.
   - La `SignaturePolicy` est sondée pour valider le statut (`partial`, `complete`, `invalid`).
4. **Hash Final** : Le `hash_full` est recalculé et confronté.

5. Exemples de Flux et Propriétés

Les signatures sous GKP ne valident pas "que" le fichier. Elles valident le fichier + sa politique de sécurité + la continuité du workflow :

```
// Extrait de l'interface en Rust du Conteneur GKP
pub struct GkpContainer {
    pub header: GkpHeader,
    pub content: EnvelopedContent,
    pub multisig: MultisigLedger,
    pub trust_anchors: Option<Vec<TrustAnchor>>,
}
```

Grâce à la conception de GKP, même décorrélé de son autorité (fonctionnement hors-ligne via un `Reader`), toute modification mineure brise la chaîne d'empreinte d'Ed25519 et passe l'état en lecture seule définitiveisée.