**🔐 Mango Cryptographic Workbench Documentation** **Section 09: Library Versioning**

**Overview**

Mango's CryptoLib versioning system is designed to ensure long-term stability, transparent upgrade paths, and robust compatibility between encrypted data and the libraries that process it. At its core, the versioning strategy is tied directly to the number of transforms known to the encryption library, ensuring that versioning is intuitive, automatic, and self-contained within the encrypted output.

**Versioning Model**

* **Transform IDs**: Each cryptographic transform is assigned a fixed, stable Transform ID starting from 1.
* **Transform Registry**: The library maintains a registry of known transforms. The number of transforms in this registry determines the version of the library.
* **Core Version Anchor**: Version 1 (v1) of CryptoLib is defined as supporting the first 40 transforms (IDs 1 through 40). These transforms are frozen and will never be altered or reordered.
* **Version Formula**:
  + Let CoreVersion = 40
  + Let maxTransformID be the highest transform ID used in an encrypted sequence
  + The **minimum required version** to decrypt a sequence is computed as:
  + requiredVersion = maxTransformID - CoreVersion + 1
  + This means:
    - Transform ID 40 => v1
    - Transform ID 41 => v2
    - Transform ID 42 => v3
    - ...and so on

**Embedded Version Header**

Every encrypted output contains a 2-byte version header:

* byte[0]: **Major Version** (based on transform count)
* byte[1]: **Minor Version** (reserved for internal tracking; ignored during decryption)

These values are written at the start of the encrypted blob, enabling tools and libraries to identify compatibility expectations immediately.

**Backwards Compatibility Guarantee**

A key promise of Mango's CryptoLib is **guaranteed backwards compatibility**:

**Any version of CryptoLib can decrypt any encrypted packet, as long as the transforms used fall within its known transform set.**

This means even an older library (e.g., v1) can decrypt a packet generated by a newer library (e.g., v5) if the sequence used only includes transforms with IDs 1–40.

This model avoids unnecessary compatibility failures and makes it possible to safely share encrypted data across devices and systems with different versions of the library, provided the selected transform sequence respects the recipient’s supported transform set.

**Validation Logic**

CryptoLib includes a CheckVersion() method which inspects a transform sequence and ensures all transform IDs are within bounds. If a packet requires transforms beyond the current library's known set, decryption will fail gracefully with an appropriate error:

if (CheckVersion(trConfig, out byte required) == false)

throw new InvalidOperationException($"Encrypted packet requires CryptoLib version {required} or higher. Decryption aborted.");

This ensures integrity while still allowing maximum compatibility.

**Benefits of the Versioning Model**

* **Self-contained**: No external version files or manifests are needed.
* **Transparent**: The version is embedded directly into the encrypted blob.
* **Stable**: The first 40 transforms are permanently frozen, ensuring consistency.
* **Predictable**: Versioning scales naturally as new transforms are added.
* **Safe**: Decryption always verifies transform compatibility before proceeding.
* **Developer-friendly**: Versioning is automatic and maintenance-free.

**Conclusion**

Mango’s transform-based versioning system provides a simple, elegant solution to cryptographic version control. By anchoring versions to transform IDs and freezing the first 40 transforms, CryptoLib offers long-term stability while still allowing future growth. Developers can confidently evolve their cryptographic capabilities without breaking compatibility for existing users.

CryptoLib will always respect this foundational promise: **if you stay within the transform range of your target audience, your encryption will always be decryptable.**