**Section 09: Library Versioning**

**Overview**

Mango’s CryptoLib versioning system is built for **stability**, **compatibility**, and **transparency**. It anchors versioning to the number of transforms known to the library, ensuring that the version of the cryptographic engine directly reflects its capabilities. Every encrypted packet carries its version metadata, enabling forward and backward compatibility checks during decryption.

**Versioning Model**

* **Transform IDs**  
  Each transform is assigned a stable and unique numeric ID, starting at 1.
* **Transform Registry**  
  The library internally maintains a registry of known transforms. The number of registered transforms directly maps to the version.
* **Core Version Anchor**
  + Version 1 (v1) is defined as supporting transform IDs **1 through 40**.
  + These 40 transforms are **frozen** and will never be removed, reordered, or altered.
* **Version Calculation**  
  Let:
  + CoreVersion = 40
  + maxTransformID = the highest ID used in a given sequence  
    Then:

requiredVersion = maxTransformID - CoreVersion + 1

**Examples:**

| **Transform ID** | **Minimum Required Version** |
| --- | --- |
| 40 | v1 |
| 41 | v2 |
| 42 | v3 |
| ... | ... |

**Embedded Version Header**

Each encrypted output contains a **2-byte header** at the start of the payload:

* byte[0]: **Major version** (based on the number of known transforms)
* byte[1]: **Minor version** (reserved for future use, currently ignored during decryption)

This header ensures that decryption tools can immediately verify compatibility.

**Backward Compatibility Promise**

💡 **Mango guarantees graceful backward compatibility.**

Any version of CryptoLib can decrypt an encrypted payload, **as long as all transform IDs in the sequence fall within its supported range**.

**✅ Example:**

* A v1 library (with transforms 1–40) **can** decrypt sequences composed entirely of IDs 1–40.
* A sequence containing transform ID 42 **requires v3**, even if most transforms are from v1.

This model **prevents unnecessary failures** while enabling secure collaboration across versioned environments.

**Version Validation Example**

CryptoLib includes a built-in check to validate transform compatibility before decryption.

if (!CheckVersion(profile.Sequence, out byte requiredVersion))

{

Console.WriteLine($"❌ Transform set requires CryptoLib version {requiredVersion} or higher.");

return;

}

**Sample Profile (with unsupported transform):**

var profile = new InputProfile("TestProfile", new (byte, byte)[]

{

(42, 1), // ← Transform ID 42 requires v3

(8, 1),

(10, 1)

}, globalRounds: 5);

Running CheckVersion() on this profile will detect that the current engine (if built with only 40 transforms) cannot decrypt it — and will gracefully report the required version.

**Benefits of the Versioning System**

| **✅ Feature** | **💬 Description** |
| --- | --- |
| **Self-contained** | No external manifests or version files required |
| **Transparent** | Version is embedded within the encrypted output |
| **Stable** | Transforms 1–40 are frozen, ensuring reproducibility across releases |
| **Predictable** | Versioning is a natural function of the highest transform ID used |
| **Safe** | Decryption fails fast if transform support is insufficient |
| **Automatic** | Developers don’t need to manage versions manually — it’s built into the system |

**Conclusion**

Mango's transform-based versioning system is designed for **future growth without fear of breakage**. By linking version numbers directly to transform capabilities, the system ensures deterministic, inspectable, and forward-compatible encryption.

As long as your transform selections respect your recipients’ known version range, **decryption is guaranteed**.

🛡️ *Mango will evolve — but never at the cost of breaking your data.*