🔐 Guía de Integración - Encriptación de Imágenes ADL API

📋 Nueva Funcionalidad Disponible

Tu integración con ADL Validation API ahora incluye **encriptación de imágenes y videos** - cualquier respuesta que contenga imágenes o videos será encriptada automáticamente. Esta funcionalidad está activa en **todos los endpoints** que reciban y envíen imágenes. Esta guía te ayudará a adaptar tu aplicación para manejar las imágenes encriptadas.

⚡ Cambios Inmediatos en tu Integración

A partir de ahora, **TODOS los endpoints de ADL API** que manejen imágenes y videos están protegidos con encriptación bidireccional:

📥 **Imágenes/Videos de Entrada (que envías):**

- **Deben estar encriptadas** antes del envío
- **Algoritmo AES-256-CBC** requerido
- **Padding PKCS7** estándar

📤 **Imágenes/Videos de Salida (que recibes):**

- **Imágenes y videos encriptados** en lugar de contenido en texto plano
- **Algoritmo AES-256-CBC** para máxima seguridad
- **Padding PKCS7** estándar

🔑 Clave Compartida de Encriptación

Solicitar Clave a ADO

La encriptación utiliza una **clave compartida de 32 bytes (256 bits)** que debe ser solicitada al equipo de ADO. Esta clave es única para tu integración y es requerida tanto para encriptar las imágenes que envías como para desencriptar las que recibes.

Especificaciones de la clave:

- **Tamaño**: 32 bytes (256 bits)
- **Algoritmo**: AES-256-CBC
- **Formato**: String que será procesado con SHA-256 para generar la clave final
- **Confidencialidad**: Esta clave debe mantenerse segura y no debe ser compartida

🧪 Aplicación de Prueba

Hemos creado una **aplicación de escritorio de prueba** que te permite validar la implementación de encriptación/desencriptación antes de integrar en tu sistema:

📦 CipherTestWpf - Aplicación de Prueba

- **Repositorio**: https://github.com/Ado-Tech/CipherTestWpf
- **Propósito**: Validar encriptación/desencriptación de imágenes
- **Incluye**: Interfaz gráfica para probar los métodos de encriptación
- **Uso**: Carga tu clave compartida y prueba con imágenes reales

Esta aplicación implementa exactamente los mismos métodos mostrados en esta guía, permitiéndote verificar que tu implementación funciona correctamente.

🛠️ Adaptación de tu Código

🔐 Encriptación de Imágenes para Envío

Método Principal: EncryptImageData

```
/// <summary>
/// Encrypts image data in base64 format using the specified algorithm
/// </summary>
/// <param name="imageBase64">The image data in base64 format to encry
/// <param name="algorithm">The encryption algorithm to use</param>
/// <returns>Encrypted image data in base64 format</returns>
```

```
public string EncryptImageData(string imageBase64, EImageEncryptionAlg
{
    if (string.IsNullOrEmpty(imageBase64))
        return imageBase64;

    try
    {
        // Convert base64 to byte array
        byte[] imageBytes = Convert.FromBase64String(imageBase64);

        // Encrypt the byte array
        byte[] encryptedBytes = EncryptImageData(imageBytes, algorithm

        // Convert back to base64
        return Convert.ToBase64String(encryptedBytes);
    }
    catch (Exception ex)
    {
        logger?.Log("Error", "EncryptImageData", $"Error encrypting im
        return imageBase64; // Return original data if encryption fail
    }
}
```

Método Interno: EncryptWithAES

```
private byte[] EncryptWithAES(byte[] data, int keySize)
{
    using (Aes aes = Aes.Create())
    {
        aes.KeySize = keySize;
        aes.Mode = CipherMode.CBC;         // Modo CBC
        aes.Padding = PaddingMode.PKCS7;  // Padding PKCS7 estándar

        // Get shared key from database configuration (32-byte key pro
        string sharedSecret = GetParameterValue(ParamsNames.IMAGE_ENCR

        byte[] sharedKey = GenerateFixedKey(sharedSecret, keySize / 8)

        aes.Key = sharedKey;
        aes.GenerateIV(); // IV can be random, it's included with encr

        using (var encryptor = aes.CreateEncryptor())
        using (var msEncrypt = new MemoryStream())
        {
            // Write IV first
```

```
            msEncrypt.Write(aes.IV, 0, aes.IV.Length);

            using (var csEncrypt = new CryptoStream(msEncrypt, encrypt
            {
                csEncrypt.Write(data, 0, data.Length);
            }
            return msEncrypt.ToArray();
        }
    }
}
```

Método de Generación de Clave: GenerateFixedKey

```
/// <summary>
/// Generates a fixed key from a shared secret string
/// </summary>
/// <param name="sharedSecret">The shared secret string</param>
/// <param name="keyLength">Required key length in bytes</param>
/// <returns>Fixed key bytes</returns>
private byte[] GenerateFixedKey(string sharedSecret, int keyLength)
{
    using (var sha256 = SHA256.Create())
    {
        byte[] hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(shared

        // If we need more bytes than SHA256 provides (32), repeat the
        if (keyLength <= hash.Length)
        {
            byte[] key = new byte[keyLength];
            Array.Copy(hash, key, keyLength);
            return key;
        }
        else
        {
            // For longer keys, concatenate multiple hashes
            byte[] extendedKey = new byte[keyLength];
            int bytesWritten = 0;
            int iteration = 0;

            while (bytesWritten < keyLength)
            {
                string iteratedSecret = sharedSecret + iteration.ToStr
                byte[] iteratedHash = sha256.ComputeHash(Encoding.UTF8

                int bytesToCopy = Math.Min(iteratedHash.Length, keyLen
```

```
                Array.Copy(iteratedHash, 0, extendedKey, bytesWritten,

                bytesWritten += bytesToCopy;
                iteration++;
            }

            return extendedKey;
        }
    }
}
```

🔓 Implementación de Desencriptación

Método Principal: DecryptImageData

```
/// <summary>
/// Decrypts image data in base64 format using the specified algorithm
/// </summary>
/// <param name="encryptedImageBase64">The encrypted image data in bas
/// <param name="algorithm">The encryption algorithm that was used</pa
/// <returns>Decrypted image data in base64 format, or null if decrypt
public string DecryptImageData(string encryptedImageBase64, EImageEncr
{
    if (string.IsNullOrEmpty(encryptedImageBase64))
        return null; // Return null instead of original data

    try
    {
        // Convert base64 to byte array
        byte[] encryptedBytes = Convert.FromBase64String(encryptedImag

        // Decrypt the byte array
        byte[] decryptedBytes = DecryptImageData(encryptedBytes, algor
        if (decryptedBytes==null)
        {
            return null;
        }
        // Convert back to base64
        return Convert.ToBase64String(decryptedBytes);
    }
    catch (Exception ex)
    {
        logger?.Log("Error", "DecryptImageData", $"Error decrypting im
        return null; // Return null if decryption fails
```

```
    }
}
```

Método Interno: DecryptWithAES

```
private byte[] DecryptWithAES(byte[] encryptedData, int keySize)
{
    using (Aes aes = Aes.Create())
    {
        aes.KeySize = keySize;
        aes.Mode = CipherMode.CBC;         // Modo CBC
        aes.Padding = PaddingMode.PKCS7;  // Padding PKCS7 estándar

        int ivSize = aes.IV.Length;

        if (encryptedData.Length < ivSize)
            throw new ArgumentException("Encrypted data is too short")

        // Extract IV from the beginning
        byte[] iv = new byte[ivSize];
        Array.Copy(encryptedData, 0, iv, 0, ivSize);

        // Get the same shared key from database configuration (32-byt
        string sharedSecret = GetParameterValue(ParamsNames.IMAGE_ENCR

        byte[] sharedKey = GenerateFixedKey(sharedSecret, keySize / 8)

        aes.IV = iv;
        aes.Key = sharedKey;

        using (var decryptor = aes.CreateDecryptor())
        using (var msDecrypt = new MemoryStream(encryptedData, ivSize,
        using (var csDecrypt = new CryptoStream(msDecrypt, decryptor,
        using (var result = new MemoryStream())
        {
            csDecrypt.CopyTo(result);
            return result.ToArray();
        }
    }
}
```