INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
“ESCOM”

SUBJECT

SELECTED TOPICS IN CRYPTOGRAPHY

Project Advances.

7CV1

Teamwork name: Selected Students in Crypto

García García Aram Jesua
Hernández Díaz Roberto Ángel

Teacher: Sandra Díaz Santiago

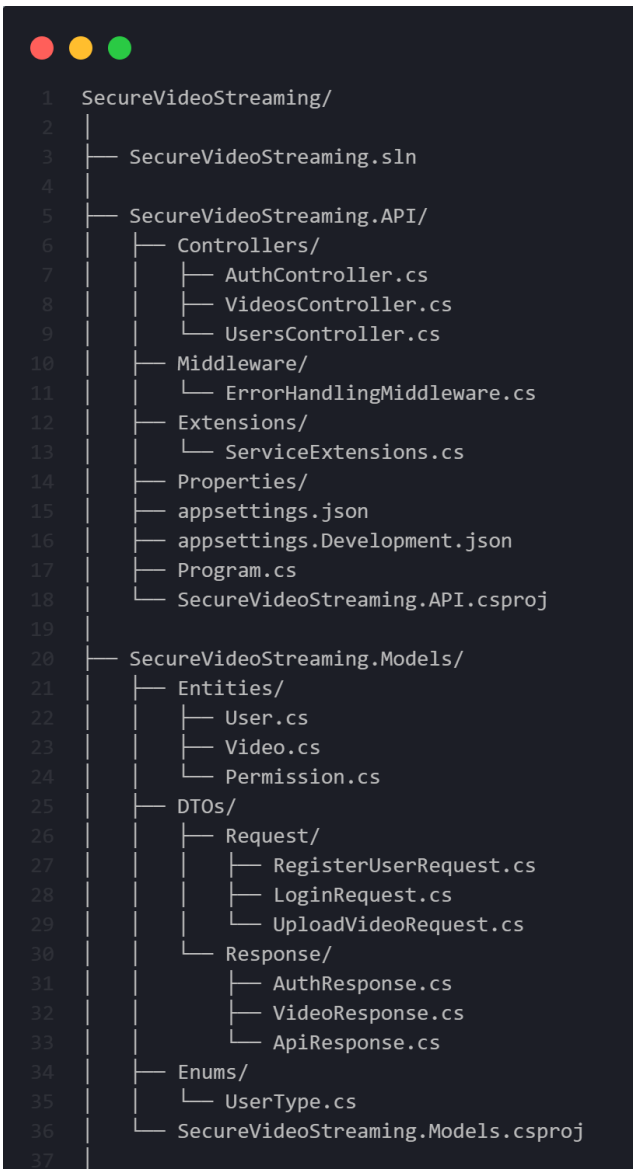
Endline date: November 24th, 2025

1. Week 1 Deliveries

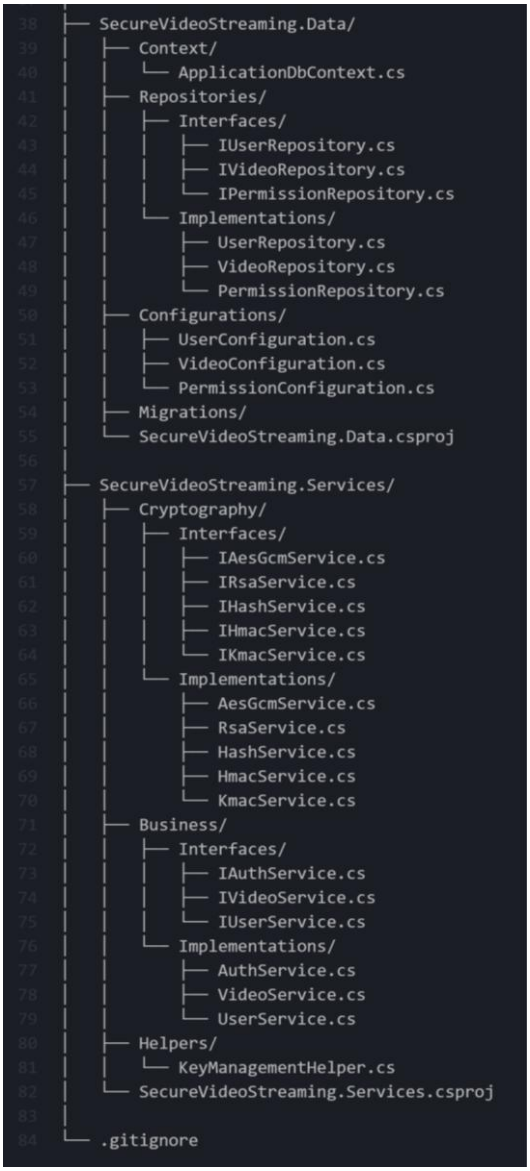
In this section I will explain every module built to elaborate environment's configuration project and basic cryptographic module that corresponds to my individual contribution.

a. Environment's configuration

Firstly, we used VS Code IDE to do code development, created an ASP.NET Core Web API project and generated the following directory structure:



```
SecureVideoStreaming/
│
├── SecureVideoStreaming.sln
│
├── SecureVideoStreaming.API/
│   ├── Controllers/
│   │   ├── AuthController.cs
│   │   ├── VideosController.cs
│   │   └── UsersController.cs
│   ├── Middleware/
│   │   └── ErrorHandlingMiddleware.cs
│   ├── Extensions/
│   │   └── ServiceExtensions.cs
│   ├── Properties/
│   ├── appsettings.json
│   ├── appsettings.Development.json
│   ├── Program.cs
│   └── SecureVideoStreaming.API.csproj
│
├── SecureVideoStreaming.Models/
│   ├── Entities/
│   │   ├── User.cs
│   │   ├── Video.cs
│   │   └── Permission.cs
│   ├── DTOs/
│   │   ├── Request/
│   │   │   ├── RegisterUserRequest.cs
│   │   │   ├── LoginRequest.cs
│   │   │   └── UploadVideoRequest.cs
│   │   └── Response/
│   │       ├── AuthResponse.cs
│   │       ├── VideoResponse.cs
│   │       └── ApiResponse.cs
│   ├── Enums/
│   │   └── UserType.cs
│   └── SecureVideoStreaming.Models.csproj
```



```
├── SecureVideoStreaming.Data/
│   ├── Context/
│   │   └── ApplicationDbContext.cs
│   ├── Repositories/
│   │   ├── Interfaces/
│   │   │   ├── IUserRepository.cs
│   │   │   ├── IVideoRepository.cs
│   │   │   └── IPermissionRepository.cs
│   │   └── Implementations/
│   │       ├── UserRepository.cs
│   │       ├── VideoRepository.cs
│   │       └── PermissionRepository.cs
│   ├── Configurations/
│   │   ├── UserConfiguration.cs
│   │   ├── VideoConfiguration.cs
│   │   └── PermissionConfiguration.cs
│   ├── Migrations/
│   └── SecureVideoStreaming.Data.csproj
│
├── SecureVideoStreaming.Services/
│   ├── Cryptography/
│   │   ├── Interfaces/
│   │   │   ├── IAesGcmService.cs
│   │   │   ├── IRsaService.cs
│   │   │   ├── IHashService.cs
│   │   │   ├── IHmacService.cs
│   │   │   └── IKmacService.cs
│   │   └── Implementations/
│   │       ├── AesGcmService.cs
│   │       ├── RsaService.cs
│   │       ├── HashService.cs
│   │       ├── HmacService.cs
│   │       └── KmacService.cs
│   ├── Business/
│   │   ├── Interfaces/
│   │   │   ├── IAuthService.cs
│   │   │   ├── IVideoService.cs
│   │   │   └── IUserService.cs
│   │   └── Implementations/
│   │       ├── AuthService.cs
│   │       ├── VideoService.cs
│   │       └── UserService.cs
│   ├── Helpers/
│   │   └── KeyManagementHelper.cs
│   └── SecureVideoStreaming.Services.csproj
│
└── .gitignore
```

As SecureVideoStreaming.API has been seen, can be understood that this is running project module, so here we need to indicate all needed libraries, such as EntityFrameworkCore, Microsoft.ApsNetCore.Authentication,

Swashbuckle.AspNetCore, and this last allows us to do Unit Tests for each module without a frontend module with a tools package named Swagger.

And in SecureVideoStreaming.Data folder and SecureVideoStreaming.Services, I indicated needed libraries, i.e., EntityFrameworkCore and EntityFrameworkCore.Relational for first, and BouncyCastle.Cryptography for second folder.

Finally, in the .gitignore file I did the following instructions:

```
## Ignore Visual Studio temporary files, build results, and
## files generated by popular Visual Studio add-ons.

# User-specific files
*.suo
*.user
*.userosscache
*.sln.docstates

# Build results
[Dd]ebug/
[Dd]ebugPublic/
[Rr]elease/
[Rr]eleases/
x64/
x86/
build/
bld/
[Bb]in/
[Oo]bj/

# Visual Studio cache/options directory
.vs/

# .NET Core
project.lock.json
project.fragment.lock.json
artifacts/

# Database
*.db
*.db-shm
*.db-wal

# Environment variables
.env
appsettings.*.json
!appsettings.Development.json

# Storage
Storage/
Videos/

# Keys (IMPORTANTE: nunca versionar claves privadas)
Keys/
*.key
*.pem
*.pfx
```

Then, to build entities and DTO's modules, over Models directory and establish all the data models that will be implemented on this project, I created UserType, to define roles on the system, user Owner (users that will upload their homemade videos) and user Consumer (users that would see these videos).

```
namespace SecureVideoStreaming.Models.Enums{
    public enum UserType
    {
        Owner = 1,
        Consumer = 2
    }
}
```

Also, I created User model, to specify all attributes that will have all system users and will manage database project, with key attributes such as, "idUsuario", "TipoUsuario" that matches with UserType enum, "PasswordHash", "Salt", "ClavePublicaRSA" are the cryptographic data needed to get access to see videos or to upload videos (it depends by the user type), and links to other entities or tables on other data models; Video model, to indicate all video attributes, with key attributes such as, "IdVideo", "IdAdministrador" matching with video's owner identifier, "NombreArchivoOriginal" that it's the name of the file's video, "TamañoVideo" (video's size), "RutaAlmacenamiento" which is the file path where will be saved the video, and links to other entities or tables on other data models; and Permission model, to indicate all permission attributes, such as "idPermiso", "idVideo" that indicates to which video is granted to be watched, "FechaOtorgamiento" which means date when permission got user and "OtorgadoPor" to define whom given it and links to other entities or tables on other data models.

```
using SecureVideoStreaming.Models.Enums;
using System;
using System.Collections.Generic;

namespace SecureVideoStreaming.Models.Entities
{
    public class User
    {
        public int IdUsuario { get; set; }
        public string NombreUsuario { get; set; } = string.Empty;
        public string Email { get; set; } = string.Empty;
        public string TipoUsuario { get; set; } = string.Empty; // 'Administrador' o 'Usuario'

        // Criptografía
        public byte[] PasswordHash { get; set; } = Array.Empty<byte>();
        public byte[] Salt { get; set; } = Array.Empty<byte>();
        public string ClavePublicaRSA { get; set; } = string.Empty;

        // Metadata
        public DateTime FechaRegistro { get; set; } = DateTime.UtcNow;
        public DateTime? UltimoAcceso { get; set; }
        public bool Activo { get; set; } = true;

        // Relaciones
        public ICollection<UserKeys> ClavesUsuarios { get; set; } = new List<UserKeys>();
        public ICollection<Video> VideosAdministrados { get; set; } = new List<Video>();
        public ICollection<Permission> Permisos { get; set; } = new List<Permission>();
        public ICollection<AccessLog> RegistrosAccesos { get; set; } = new List<AccessLog>();
    }
}
```

```
using System;
using System.Collections.Generic;

namespace SecureVideoStreaming.Models.Entities
{
    public class Video
    {
        public int IdVideo { get; set; }
        public int IdAdministrador { get; set; }
        public string TituloVideo { get; set; } = string.Empty;
        public string? Descripcion { get; set; }
        public string NombreArchivoOriginal { get; set; } = string.Empty;
        public string NombreArchivoCifrado { get; set; } = string.Empty;
        public long TamañoArchivo { get; set; }
        public int? Duracion { get; set; }
        public string? FormatoVideo { get; set; }
        public string RutaAlmacenamiento { get; set; } = string.Empty;
        public string EstadoProcesamiento { get; set; } = "Procesando"; // 'Procesando', 'Disponible', 'Error', 'Eliminado'

        // Metadata
        public DateTime FechaSubida { get; set; } = DateTime.UtcNow;
        public DateTime? FechaModificacion { get; set; }

        // Relaciones
        public User Administrador { get; set; } = null!;
        public CryptoData? DatosCriptograficos { get; set; }
        public ICollection<Permission> Permisos { get; set; } = new List<Permission>();
        public ICollection<AccessLog> RegistrosAccesos { get; set; } = new List<AccessLog>();
    }
}
```

```
using System;

namespace SecureVideoStreaming.Models.Entities
{
    public class Permission
    {
        public int IdPermiso { get; set; }
        public int IdVideo { get; set; }
        public int IdUsuario { get; set; }
        public string TipoPermiso { get; set; } = "Lectura"; // 'Pendiente', 'Aprobado', 'Revocado'

        // Metadata
        public DateTime FechaOtorgamiento { get; set; } = DateTime.UtcNow;
        public DateTime? FechaExpiracion { get; set; }
        public DateTime? FechaRevocacion { get; set; }
        public int NumeroAccesos { get; set; } = 0;
        public int? MaxAccesos { get; set; } // Límite de accesos (null = ilimitado)
        public DateTime? UltimoAcceso { get; set; }

        // Otorgamiento y Revocación
        public int OtorgadoPor { get; set; }
        public int? RevocadoPor { get; set; }

        // Relaciones
        public Video Video { get; set; } = null!;
        public User Usuario { get; set; } = null!;
        public User UsuarioOtorgante { get; set; } = null!;
        public User? UsuarioRevocador { get; set; }
    }
}
```

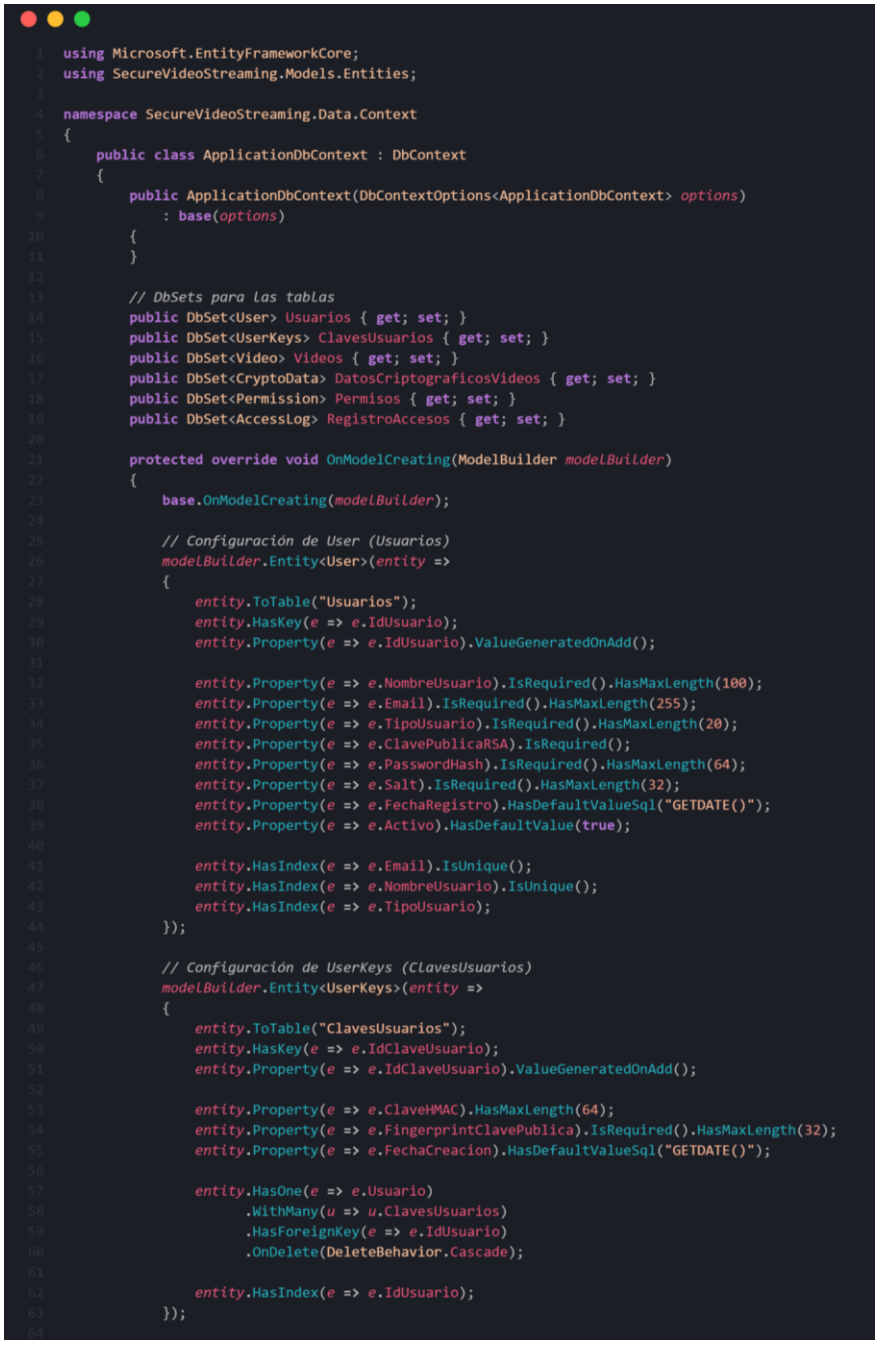
Also, I created ApiResponse, RegisterUserRequest, LoginRequest, to define what is the exact data that will receive or give the system from or to user.

```
namespace SecureVideoStreaming.Models.DTOs.Response
{
    public class ApiResponse<T>
    {
        public bool Success { get; set; }
        public string Message { get; set; } = string.Empty;
        public T? Data { get; set; }
        public List<string> Errors { get; set; } = new List<string>();

        public static ApiResponse<T> SuccessResponse(T data, string message = "Success")
        {
            return new ApiResponse<T>
            {
                Success = true,
                Message = message,
                Data = data
            };
        }

        public static ApiResponse<T> ErrorResponse(string message, List<string>? errors = null)
        {
            return new ApiResponse<T>
            {
                Success = false,
                Message = message,
                Errors = errors ?? new List<string>()
            };
        }
    }
}
```

```
using System.ComponentModel.DataAnnotations;

namespace SecureVideoStreaming.Models.DTOs.Request
{
    public class RegisterUserRequest
    {
        [Required(ErrorMessage = "El nombre de usuario es requerido")]
        [StringLength(100, MinimumLength = 3, ErrorMessage = "El nombre debe tener entre 3 y 100 caracteres")]
        public string NombreUsuario { get; set; } = string.Empty;

        [Required(ErrorMessage = "El email es requerido")]
        [EmailAddress(ErrorMessage = "Email no válido")]
        [StringLength(255)]
        public string Email { get; set; } = string.Empty;

        [Required(ErrorMessage = "La contraseña es requerida")]
        [StringLength(100, MinimumLength = 6, ErrorMessage = "La contraseña debe tener al menos 6 caracteres")]
        public string Password { get; set; } = string.Empty;

        [Required(ErrorMessage = "El tipo de usuario es requerido")]
        [RegularExpression("^(Administrador|Usuario)$", ErrorMessage = "Tipo de usuario debe ser 'Administrador' o 'Usuario'")]
        public string TipoUsuario { get; set; } = "Usuario"; // "Administrador" o "Usuario"
    }
}
```

```
using System.ComponentModel.DataAnnotations;

namespace SecureVideoStreaming.Models.DTOs.Request
{
    public class LoginRequest
    {
        [Required(ErrorMessage = "El email es requerido")]
        [EmailAddress(ErrorMessage = "Email no válido")]
        public string Email { get; set; } = string.Empty;

        [Required(ErrorMessage = "La contraseña es requerida")]
        public string Password { get; set; } = string.Empty;
    }
}
```

Then, in a way that we establish an intermediate between the services that we will create and the database project, I created ApplicationDbContext, searching a fast way to configure how the system environment will be communicated with our database.



```
using Microsoft.EntityFrameworkCore;
using SecureVideoStreaming.Models.Entities;

namespace SecureVideoStreaming.Data.Context
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options)
        {
        }

        // DbSets para las tablas
        public DbSet<User> Usuarios { get; set; }
        public DbSet<UserKeys> ClavesUsuarios { get; set; }
        public DbSet<Video> Videos { get; set; }
        public DbSet<CryptoData> DatosCriptograficosVideos { get; set; }
        public DbSet<Permission> Permisos { get; set; }
        public DbSet<AccessLog> RegistroAccesos { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // Configuración de User (Usuarios)
            modelBuilder.Entity<User>(entity =>
            {
                entity.ToTable("Usuarios");
                entity.HasKey(e => e.IdUsuario);
                entity.Property(e => e.IdUsuario).ValueGeneratedOnAdd();

                entity.Property(e => e.NombreUsuario).IsRequired().HasMaxLength(100);
                entity.Property(e => e.Email).IsRequired().HasMaxLength(255);
                entity.Property(e => e.TipoUsuario).IsRequired().HasMaxLength(20);
                entity.Property(e => e.ClavePublicaRSA).IsRequired();
                entity.Property(e => e.PasswordHash).IsRequired().HasMaxLength(64);
                entity.Property(e => e.Salt).IsRequired().HasMaxLength(32);
                entity.Property(e => e.FechaRegistro).HasDefaultValueSql("GETDATE()");
                entity.Property(e => e.Activo).HasDefaultValue(true);

                entity.HasIndex(e => e.Email).IsUnique();
                entity.HasIndex(e => e.NombreUsuario).IsUnique();
                entity.HasIndex(e => e.TipoUsuario);
            });

            // Configuración de UserKeys (ClavesUsuarios)
            modelBuilder.Entity<UserKeys>(entity =>
            {
                entity.ToTable("ClavesUsuarios");
                entity.HasKey(e => e.IdClaveUsuario);
                entity.Property(e => e.IdClaveUsuario).ValueGeneratedOnAdd();

                entity.Property(e => e.ClaveHMAC).HasMaxLength(64);
                entity.Property(e => e.FingerprintClavePublica).IsRequired().HasMaxLength(32);
                entity.Property(e => e.FechaCreacion).HasDefaultValueSql("GETDATE()");

                entity.HasOne(e => e.Usuario)
                      .WithMany(u => u.ClavesUsuarios)
                      .HasForeignKey(e => e.IdUsuario)
                      .OnDelete(DeleteBehavior.Cascade);

                entity.HasIndex(e => e.IdUsuario);
            });
```

```
            // Configuración de Video (Videos)
            modelBuilder.Entity<Video>(entity =>
            {
                entity.ToTable("Videos");
                entity.HasKey(e => e.IdVideo);
                entity.Property(e => e.IdVideo).ValueGeneratedOnAdd();

                entity.Property(e => e.TituloVideo).IsRequired().HasMaxLength(255);
                entity.Property(e => e.NombreArchivoOriginal).IsRequired().HasMaxLength(255);
                entity.Property(e => e.NombreArchivoCifrado).IsRequired().HasMaxLength(255);
                entity.Property(e => e.RutaAlmacenamiento).IsRequired().HasMaxLength(500);
                entity.Property(e => e.EstadoProcesamiento).IsRequired().HasMaxLength(50).HasDefaultValue("Procesando");
                entity.Property(e => e.FormatoVideo).HasMaxLength(50);
                entity.Property(e => e.FechaSubida).HasDefaultValueSql("GETDATE()");

                entity.HasOne(e => e.Administrador)
                      .WithMany(u => u.VideosAdministrados)
                      .HasForeignKey(e => e.IdAdministrador)
                      .OnDelete(DeleteBehavior.Restrict);

                entity.HasIndex(e => e.NombreArchivoCifrado).IsUnique();
                entity.HasIndex(e => e.IdAdministrador);
                entity.HasIndex(e => e.EstadoProcesamiento);
                entity.HasIndex(e => e.FechaSubida);
            });

            // Configuración de CryptoData (DatosCriptograficosVideos)
            modelBuilder.Entity<CryptoData>(entity =>
            {
                entity.ToTable("DatosCriptograficosVideos");
                entity.HasKey(e => e.IdDatoCripto);
                entity.Property(e => e.IdDatoCripto).ValueGeneratedOnAdd();

                entity.Property(e => e.KEKCifrada).IsRequired();
                entity.Property(e => e.AlgoritmoKEK).IsRequired().HasMaxLength(50).HasDefaultValue("ChaCha20-Poly1305");
                entity.Property(e => e.Nonce).IsRequired().HasMaxLength(12);
                entity.Property(e => e.AuthTag).IsRequired().HasMaxLength(16);
                entity.Property(e => e.HashSHA256Original).IsRequired().HasMaxLength(32);
                entity.Property(e => e.HMACDelVideo).IsRequired().HasMaxLength(64);
                entity.Property(e => e.FechaGeneracionClaves).HasDefaultValueSql("GETDATE()");
                entity.Property(e => e.VersionAlgoritmo).IsRequired().HasMaxLength(20).HasDefaultValue("1.0");

                entity.HasOne(e => e.Video)
                      .WithOne(v => v.DatosCriptograficos)
                      .HasForeignKey<CryptoData>(e => e.IdVideo)
                      .OnDelete(DeleteBehavior.Cascade);

                entity.HasIndex(e => e.IdVideo).IsUnique();
            });
```

```
            // Configuración de Permission (Permisos)
            modelBuilder.Entity<Permission>(entity =>
            {
                entity.ToTable("Permisos");
                entity.HasKey(e => e.IdPermiso);
                entity.Property(e => e.IdPermiso).ValueGeneratedOnAdd();

                entity.Property(e => e.TipoPermiso).IsRequired().HasMaxLength(50).HasDefaultValue("Lectura");
                entity.Property(e => e.FechaOtorgamiento).HasDefaultValueSql("GETDATE()");
                entity.Property(e => e.NumeroAccesos).HasDefaultValue(0);

                entity.HasOne(e => e.Video)
                      .WithMany(v => v.Permisos)
                      .HasForeignKey(e => e.IdVideo)
                      .OnDelete(DeleteBehavior.Cascade);

                entity.HasOne(e => e.Usuario)
                      .WithMany(u => u.Permisos)
                      .HasForeignKey(e => e.IdUsuario)
                      .OnDelete(DeleteBehavior.NoAction);

                entity.HasOne(e => e.UsuarioOtorgante)
                      .WithMany()
                      .HasForeignKey(e => e.OtorgadoPor)
                      .OnDelete(DeleteBehavior.NoAction);

                entity.HasOne(e => e.UsuarioRevocador)
                      .WithMany()
                      .HasForeignKey(e => e.RevocadoPor)
                      .OnDelete(DeleteBehavior.NoAction);

                entity.HasIndex(e => new { e.IdVideo, e.IdUsuario }).IsUnique();
                entity.HasIndex(e => e.IdVideo);
                entity.HasIndex(e => e.IdUsuario);
                entity.HasIndex(e => e.TipoPermiso);
                entity.HasIndex(e => e.FechaExpiracion);
            });

            // Configuración de AccessLog (RegistroAccesos)
            modelBuilder.Entity<AccessLog>(entity =>
            {
                entity.ToTable("RegistroAccesos");
                entity.HasKey(e => e.IdRegistro);
                entity.Property(e => e.IdRegistro).ValueGeneratedOnAdd();

                entity.Property(e => e.TipoAcceso).IsRequired().HasMaxLength(50);
                entity.Property(e => e.DireccionIP).HasMaxLength(50);
                entity.Property(e => e.UserAgent).HasMaxLength(500);
                entity.Property(e => e.FechaHoraAcceso).HasDefaultValueSql("GETDATE()");

                entity.HasOne(e => e.Usuario)
                      .WithMany(u => u.RegistrosAccesos)
                      .HasForeignKey(e => e.IdUsuario)
                      .OnDelete(DeleteBehavior.NoAction);

                entity.HasOne(e => e.Video)
                      .WithMany(v => v.RegistrosAccesos)
                      .HasForeignKey(e => e.IdVideo)
                      .OnDelete(DeleteBehavior.NoAction);

                entity.HasIndex(e => e.IdUsuario);
                entity.HasIndex(e => e.IdVideo);
                entity.HasIndex(e => e.FechaHoraAcceso);
                entity.HasIndex(e => e.TipoAcceso);
            });
        }
    }
}
```

Also, I modified Program file (which was auto generated when we created the project) to define the controllers, database, HTTP request pipeline, authorization, and test deployment configuration, because this file is that will be executed when we run the project.

```
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using SecureVideoStreaming.Data.Context;
using SecureVideoStreaming.API.Extensions;
using SecureVideoStreaming.API.Middleware;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddControllers();
builder.Services.AddRazorPages();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Session support for Razor Pages
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.IdleTimeout = TimeSpan.FromMinutes(30);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

// JWT Authentication
var jwtSettings = builder.Configuration.GetSection("Jwt");
var secretKey = jwtSettings["SecretKey"];
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = jwtSettings["Issuer"],
        ValidAudience = jwtSettings["Audience"],
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(secretKey!))
    };
});
```

```
// Database Configuration - SQL Server
builder.Services.AddDbContext<ApplicationDbContext>(options =>
{
    options.UseSqlServer(
        builder.Configuration.GetConnectionString("DefaultConnection"),
        sqlServerOptionsAction: sqlOptions =>
        {
            sqlOptions.EnableRetryOnFailure(
                maxRetryCount: 5,
                maxRetryDelay: TimeSpan.FromSeconds(30),
                errorNumbersToAdd: null);
        });
});

// Cryptography Services
builder.Services.AddCryptographyServices();
```

```
// Business Services
builder.Services.AddBusinessServices();
builder.Services.AddScoped<SecureVideoStreaming.Services.Business.Interfaces.IVideoService, SecureVideoStreaming.Services.Business.Implementations.VideoService>();

// CORS Configuration
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", policy =>
    {
        policy.AllowAnyOrigin()
              .AllowAnyMethod()
              .AllowAnyHeader();
    });
});

var app = builder.Build();

// Configure the HTTP request pipeline.
// IMPORTANTE: ErrorHandling debe estar PRIMERO para capturar todas las excepciones
app.UseErrorHandling();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseCors("AllowAll");
app.UseSession();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.MapRazorPages();

app.Run();
```

Then, database connection settings were defined in the appsettings.json and appsettings.Development.json.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost\\SQLEXPRESS;Database=Data_base_cripto;Integrated Security=True;TrustServerCertificate=True;MultipleActiveResultSets=true"
  },
  "Jwt": {
    "SecretKey": "TU_CLAVE_SECRETA_MUY_LARGA_Y_SEGURA_MINIMO_32_CARACTERES_AQUI",
    "Issuer": "SecureVideoStreaming",
    "Audience": "SecureVideoStreamingUsers",
    "ExpirationMinutes": 60
  },
  "Storage": {
    "VideosPath": "./Storage/Videos",
    "KeysPath": "./Storage/Keys"
  }
}
```

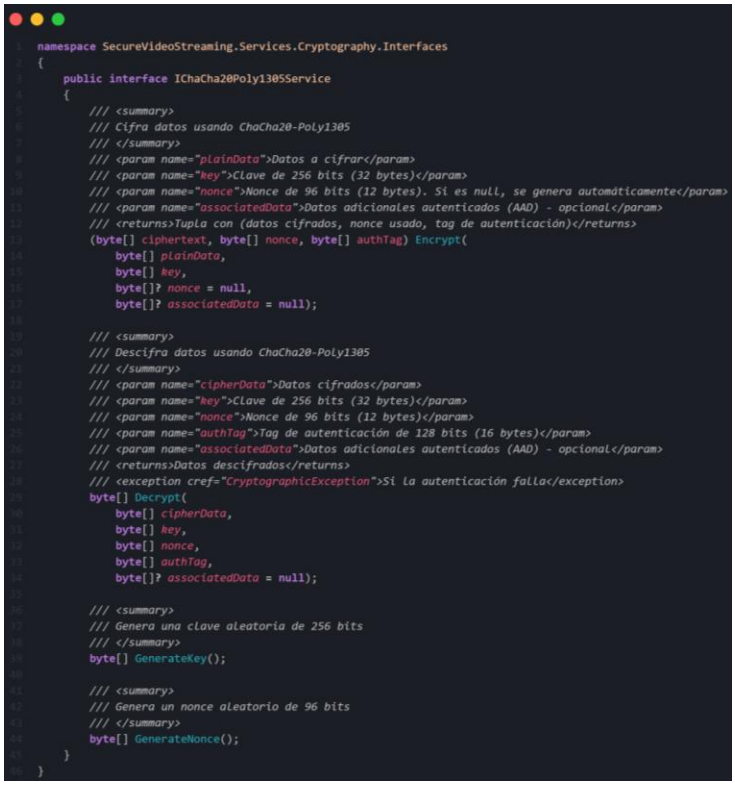
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "Microsoft.AspNetCore": "Information",
      "Microsoft.EntityFrameworkCore.Database.Command": "Information"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost\\SQLEXPRESS;Database=Data_base_cripto;Integrated Security=True;TrustServerCertificate=True;MultipleActiveResultSets=true;Encrypt=False"
  }
}
```

And finally, was created a HealthController, a tool with the mission of helping us to test the system for first time on Swagger tool.

```
using Microsoft.AspNetCore.Mvc;

namespace SecureVideoStreaming.API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class HealthController : ControllerBase
    {
        [HttpGet]
        public IActionResult Get()
        {
            return Ok(new
            {
                status = "healthy",
                timestamp = DateTime.UtcNow,
                version = "1.0.0"
            });
        }
    }
}
```

b. Cryptographic base module

For this part, I was guided by the architecture defined before (a controllers layer and a services layer), so here firstly I create the service layer, considering the cryptographic algorithms, such as, RSA, ChaCha20-Poly1305, SHA256 (Hash) and HMAC, and for each service files that will have each algorithm, I need to create an interface to only define how will be the parameters that will receive the functions that I will define before on the services files.



```
namespace SecureVideoStreaming.Services.Cryptography.Interfaces
{
    public interface IChaCha20Poly1305Service
    {
        /// <summary>
        /// Cifra datos usando ChaCha20-Poly1305
        /// </summary>
        /// <param name="plainData">Datos a cifrar</param>
        /// <param name="key">Clave de 256 bits (32 bytes)</param>
        /// <param name="nonce">Nonce de 96 bits (12 bytes). Si es null, se genera automáticamente</param>
        /// <param name="associatedData">Datos adicionales autenticados (AAD) - opcional</param>
        /// <returns>Tupla con (datos cifrados, nonce usado, tag de autenticación)</returns>
        (byte[] ciphertext, byte[] nonce, byte[] authTag) Encrypt(
            byte[] plainData,
            byte[] key,
            byte[]? nonce = null,
            byte[]? associatedData = null);

        /// <summary>
        /// Descifra datos usando ChaCha20-Poly1305
        /// </summary>
        /// <param name="cipherData">Datos cifrados</param>
        /// <param name="key">Clave de 256 bits (32 bytes)</param>
        /// <param name="nonce">Nonce de 96 bits (12 bytes)</param>
        /// <param name="authTag">Tag de autenticación de 128 bits (16 bytes)</param>
        /// <param name="associatedData">Datos adicionales autenticados (AAD) - opcional</param>
        /// <returns>Datos descifrados</returns>
        /// <exception cref="CryptographicException">Si la autenticación falla</exception>
        byte[] Decrypt(
            byte[] cipherData,
            byte[] key,
            byte[] nonce,
            byte[] authTag,
            byte[]? associatedData = null);

        /// <summary>
        /// Genera una clave aleatoria de 256 bits
        /// </summary>
        byte[] GenerateKey();

        /// <summary>
        /// Genera un nonce aleatorio de 96 bits
        /// </summary>
        byte[] GenerateNonce();
    }
}
```

```csharp
namespace SecureVideoStreaming.Services.Cryptography.Interfaces
{
    public interface IRsaService
    {
        /// <summary>
        /// Genera un par de claves RSA (pública y privada)
        /// </summary>
        /// <param name="keySize">Tamaño de la clave (2048 o 4096 bits recomendado)</param>
        /// <returns>Tupla con (clave pública PEM, clave privada PEM)</returns>
        (string publicKey, string privateKey) GenerateKeyPair(int keySize = 2048);

        /// <summary>
        /// Cifra datos con la clave pública RSA usando OAEP
        /// </summary>
        /// <param name="data">Datos a cifrar (máximo ~214 bytes para RSA-2048)</param>
        /// <param name="publicKeyPem">Clave pública en formato PEM</param>
        /// <returns>Datos cifrados</returns>
        byte[] Encrypt(byte[] data, string publicKeyPem);

        /// <summary>
        /// Descifra datos con la clave privada RSA usando OAEP
        /// </summary>
        /// <param name="encryptedData">Datos cifrados</param>
        /// <param name="privateKeyPem">Clave privada en formato PEM</param>
        /// <returns>Datos descifrados</returns>
        byte[] Decrypt(byte[] encryptedData, string privateKeyPem);

        /// <summary>
        /// Firma datos con la clave privada RSA
        /// </summary>
        byte[] Sign(byte[] data, string privateKeyPem);

        /// <summary>
        /// Verifica la firma de datos con la clave pública RSA
        /// </summary>
        bool VerifySignature(byte[] data, byte[] signature, string publicKeyPem);
    }
}
```

```csharp
namespace SecureVideoStreaming.Services.Cryptography.Interfaces
{
    public interface IHashService
    {
        /// <summary>
        /// Calcula el hash SHA-256 de los datos
        /// </summary>
        byte[] ComputeSha256(byte[] data);

        /// <summary>
        /// Calcula el hash SHA-256 de un stream (útil para archivos grandes)
        /// </summary>
        byte[] ComputeSha256(Stream stream);

        /// <summary>
        /// Calcula el hash SHA-256 y lo retorna en formato hexadecimal
        /// </summary>
        string ComputeSha256Hex(byte[] data);

        /// <summary>
        /// Calcula el hash SHA-256 de un stream y lo retorna en formato hexadecimal
        /// </summary>
        string ComputeSha256Hex(Stream stream);

        /// <summary>
        /// Deriva una clave usando PBKDF2 con SHA-256
        /// </summary>
        /// <param name="password">Contraseña</param>
        /// <param name="salt">Salt (mínimo 16 bytes)</param>
        /// <param name="iterations">Número de iteraciones (mínimo 100,000)</param>
        /// <param name="keyLength">Longitud de la clave derivada en bytes</param>
        byte[] DeriveKey(string password, byte[] salt, int iterations = 100000, int keyLength = 32);

        /// <summary>
        /// Genera un salt aleatorio
        /// </summary>
        byte[] GenerateSalt(int length = 32);
    }
}
```

```
namespace SecureVideoStreaming.Services.Cryptography.Interfaces
{
    public interface IHmacService
    {
        /// <summary>
        /// Calcula HMAC-SHA256
        /// </summary>
        byte[] ComputeHmac(byte[] data, byte[] key);

        /// <summary>
        /// Calcula HMAC-SHA256 de un stream
        /// </summary>
        byte[] ComputeHmac(Stream stream, byte[] key);

        /// <summary>
        /// Calcula HMAC-SHA256 y lo retorna en formato hexadecimal
        /// </summary>
        string ComputeHmacHex(byte[] data, byte[] key);

        /// <summary>
        /// Verifica un HMAC
        /// </summary>
        bool VerifyHmac(byte[] data, byte[] key, byte[] hmacToVerify);

        /// <summary>
        /// Genera una clave HMAC aleatoria
        /// </summary>
        byte[] GenerateKey(int length = 64);
    }
}
```

Then, I created each service file for each algorithm, so here is where all algorithm functions was built, from encrypt, decrypt, generate key and generate nonce process with ChaCha20-Poly1305, generate key pair, encrypt, decrypt, sign, and verify signature process for RSA-OAEP, compute hash, derive key and generate salt for Hash process with SHA-256, compute HMAC, verify HMAC and generate key process for HMAC.

```
using SecureVideoStreaming.Services.Cryptography.Interfaces;
using System.Security.Cryptography;

namespace SecureVideoStreaming.Services.Cryptography.Implementations
{
    public class ChaCha20Poly1305Service : IChaCha20Poly1305Service
    {
        private const int KEY_SIZE = 32; // 256 bits
        private const int NONCE_SIZE = 12; // 96 bits
        private const int TAG_SIZE = 16; // 128 bits

        public (byte[] ciphertext, byte[] nonce, byte[] authTag) Encrypt(
            byte[] plainData,
            byte[] key,
            byte[]? nonce = null,
            byte[]? associatedData = null)
        {
            if (plainData == null || plainData.Length == 0)
                throw new ArgumentException("Los datos no pueden estar vacíos", nameof(plainData));

            if (key == null || key.Length != KEY_SIZE)
                throw new ArgumentException($"La clave debe tener {KEY_SIZE} bytes", nameof(key));

            // Generar nonce si no se proporciona
            if (nonce == null)
            {
                nonce = GenerateNonce();
            }
            else if (nonce.Length != NONCE_SIZE)
            {
                throw new ArgumentException($"El nonce debe tener {NONCE_SIZE} bytes", nameof(nonce));
            }
```

```
            try
            {
                // Crear buffers para ciphertext y tag
                var ciphertext = new byte[plainData.Length];
                var authTag = new byte[TAG_SIZE];

                // Usar ChaCha20Poly1305 nativo de .NET
                using var cipher = new System.Security.Cryptography.ChaCha20Poly1305(key);

                cipher.Encrypt(
                    nonce: nonce,
                    plaintext: plainData,
                    ciphertext: ciphertext,
                    tag: authTag,
                    associatedData: associatedData ?? Array.Empty<byte>());

                return (ciphertext, nonce, authTag);
            }
            catch (Exception ex)
            {
                throw new CryptographicException("Error al cifrar con ChaCha20-Poly1305", ex);
            }
        }

        public byte[] Decrypt(
            byte[] cipherData,
            byte[] key,
            byte[] nonce,
            byte[] authTag,
            byte[]? associatedData = null)
        {
            if (cipherData == null || cipherData.Length == 0)
                throw new ArgumentException("Los datos cifrados no pueden estar vacíos", nameof(cipherData));

            if (key == null || key.Length != KEY_SIZE)
                throw new ArgumentException($"La clave debe tener {KEY_SIZE} bytes", nameof(key));

            if (nonce == null || nonce.Length != NONCE_SIZE)
                throw new ArgumentException($"El nonce debe tener {NONCE_SIZE} bytes", nameof(nonce));

            if (authTag == null || authTag.Length != TAG_SIZE)
                throw new ArgumentException($"El tag de autenticación debe tener {TAG_SIZE} bytes", nameof(authTag));

            try
            {
                // Crear buffer para plaintext
                var plaintext = new byte[cipherData.Length];

                // Usar ChaCha20Poly1305 nativo de .NET
                using var cipher = new System.Security.Cryptography.ChaCha20Poly1305(key);

                cipher.Decrypt(
                    nonce: nonce,
                    ciphertext: cipherData,
                    tag: authTag,
                    plaintext: plaintext,
                    associatedData: associatedData ?? Array.Empty<byte>());

                return plaintext;
            }
            catch (CryptographicException ex)
            {
                throw new CryptographicException("Error de autenticación: los datos han sido modificados o la clave/nonce son incorrectos", ex);
            }
            catch (Exception ex)
            {
                throw new CryptographicException("Error al descifrar con ChaCha20-Poly1305", ex);
            }
        }

        public byte[] GenerateKey()
        {
            return RandomNumberGenerator.GetBytes(KEY_SIZE);
        }

        public byte[] GenerateNonce()
        {
            return RandomNumberGenerator.GetBytes(NONCE_SIZE);
        }
    }
}
```

```
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Encodings;
using Org.BouncyCastle.Crypto.Engines;
using Org.BouncyCastle.Crypto.Generators;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.OpenSsl;
using Org.BouncyCastle.Security;
using SecureVideoStreaming.Services.Cryptography.Interfaces;
using System.Security.Cryptography;
using System.Text;

namespace SecureVideoStreaming.Services.Cryptography.Implementations
{
    public class RsaService : IRsaService
    {
        public (string publicKey, string privateKey) GenerateKeyPair(int keySize = 2048)
        {
            if (keySize != 2048 && keySize != 4096)
                throw new ArgumentException("El tamaño de clave debe ser 2048 o 4096 bits", nameof(keySize));

            try
            {
                var keyPairGenerator = new RsaKeyPairGenerator();
                keyPairGenerator.Init(new KeyGenerationParameters(new SecureRandom(), keySize));
                var keyPair = keyPairGenerator.GenerateKeyPair();

                // Exportar a formato PEM
                string publicKeyPem, privateKeyPem;

                using (var publicWriter = new StringWriter())
                {
                    var pemWriter = new PemWriter(publicWriter);
                    pemWriter.WriteObject(keyPair.Public);
                    publicKeyPem = publicWriter.ToString();
                }

                using (var privateWriter = new StringWriter())
                {
                    var pemWriter = new PemWriter(privateWriter);
                    pemWriter.WriteObject(keyPair.Private);
                    privateKeyPem = privateWriter.ToString();
                }

                return (publicKeyPem, privateKeyPem);
            }
            catch (Exception ex)
            {
                throw new CryptographicException("Error al generar par de claves RSA", ex);
            }
        }
```

```
        public byte[] Encrypt(byte[] data, string publicKeyPem)
        {
            if (data == null || data.Length == 0)
                throw new ArgumentException("Los datos no pueden estar vacíos", nameof(data));

            if (string.IsNullOrWhiteSpace(publicKeyPem))
                throw new ArgumentException("La clave pública no puede estar vacía", nameof(publicKeyPem));

            try
            {
                // Cargar clave pública
                AsymmetricKeyParameter publicKey;
                using (var reader = new StringReader(publicKeyPem))
                {
                    var pemReader = new PemReader(reader);
                    publicKey = (AsymmetricKeyParameter)pemReader.ReadObject();
                }

                // Cifrar con OAEP (SHA-256)
                var engine = new OaepEncoding(new RsaEngine(), new Org.BouncyCastle.Crypto.Digests.Sha256Digest());
                engine.Init(true, publicKey);

                return engine.ProcessBlock(data, 0, data.Length);
            }
            catch (Exception ex)
            {
                throw new CryptographicException("Error al cifrar con RSA", ex);
            }
        }

        public byte[] Decrypt(byte[] encryptedData, string privateKeyPem)
        {
            if (encryptedData == null || encryptedData.Length == 0)
                throw new ArgumentException("Los datos cifrados no pueden estar vacíos", nameof(encryptedData));

            if (string.IsNullOrWhiteSpace(privateKeyPem))
                throw new ArgumentException("La clave privada no puede estar vacía", nameof(privateKeyPem));

            try
            {
                // Cargar clave privada
                AsymmetricKeyParameter privateKey;
                using (var reader = new StringReader(privateKeyPem))
                {
                    var pemReader = new PemReader(reader);
                    var keyPair = pemReader.ReadObject();

                    if (keyPair is AsymmetricCipherKeyPair pair)
                        privateKey = pair.Private;
                    else
                        privateKey = (AsymmetricKeyParameter)keyPair;
                }

                // Descifrar con OAEP (SHA-256)
                var engine = new OaepEncoding(new RsaEngine(), new Org.BouncyCastle.Crypto.Digests.Sha256Digest());
                engine.Init(false, privateKey);

                return engine.ProcessBlock(encryptedData, 0, encryptedData.Length);
            }
            catch (Exception ex)
            {
                throw new CryptographicException("Error al descifrar con RSA", ex);
            }
        }
```

```
        public byte[] Sign(byte[] data, string privateKeyPem)
        {
            if (data == null || data.Length == 0)
                throw new ArgumentException("Los datos no pueden estar vacios", nameof(data));

            if (string.IsNullOrWhiteSpace(privateKeyPem))
                throw new ArgumentException("La clave privada no puede estar vacía", nameof(privateKeyPem));

            try
            {
                // Cargar clave privada
                AsymmetricKeyParameter privateKey;
                using (var reader = new StringReader(privateKeyPem))
                {
                    var pemReader = new PemReader(reader);
                    var keyPair = pemReader.ReadObject();

                    if (keyPair is AsymmetricCipherKeyPair pair)
                        privateKey = pair.Private;
                    else
                        privateKey = (AsymmetricKeyParameter)keyPair;
                }

                // Firmar con SHA-256
                var signer = SignerUtilities.GetSigner("SHA256withRSA");
                signer.Init(true, privateKey);
                signer.BlockUpdate(data, 0, data.Length);

                return signer.GenerateSignature();
            }
            catch (Exception ex)
            {
                throw new CryptographicException("Error al firmar con RSA", ex);
            }
        }

        public bool VerifySignature(byte[] data, byte[] signature, string publicKeyPem)
        {
            if (data == null || data.Length == 0)
                throw new ArgumentException("Los datos no pueden estar vacios", nameof(data));

            if (signature == null || signature.Length == 0)
                throw new ArgumentException("La firma no puede estar vacía", nameof(signature));

            if (string.IsNullOrWhiteSpace(publicKeyPem))
                throw new ArgumentException("La clave pública no puede estar vacía", nameof(publicKeyPem));

            try
            {
                // Cargar clave pública
                AsymmetricKeyParameter publicKey;
                using (var reader = new StringReader(publicKeyPem))
                {
                    var pemReader = new PemReader(reader);
                    publicKey = (AsymmetricKeyParameter)pemReader.ReadObject();
                }

                // Verificar firma con SHA-256
                var verifier = SignerUtilities.GetSigner("SHA256withRSA");
                verifier.Init(false, publicKey);
                verifier.BlockUpdate(data, 0, data.Length);

                return verifier.VerifySignature(signature);
            }
            catch
            {
                return false;
            }
        }
    }
}
```

Then, in the ServiceExtensions file was registered all services in the dependencies docker and was created a cryptographic test controller.

```csharp
using SecureVideoStreaming.Services.Business.Implementations;
using SecureVideoStreaming.Services.Business.Interfaces;
using SecureVideoStreaming.Services.Cryptography.Implementations;
using SecureVideoStreaming.Services.Cryptography.Interfaces;

namespace SecureVideoStreaming.API.Extensions
{
    public static class ServiceExtensions
    {
        public static IServiceCollection AddCryptographyServices(this IServiceCollection services)
        {
            // Registrar servicios criptográficos como Singleton (son stateless)
            services.AddSingleton<IChaCha20Poly1305Service, ChaCha20Poly1305Service>();
            services.AddSingleton<IRsaService, RsaService>();
            services.AddSingleton<IHashService, HashService>();
            services.AddSingleton<IHmacService, HmacService>();
            services.AddSingleton<IKmacService, KmacService>();

            // Servicios de gestión de claves
            services.AddSingleton<IKeyManagementService, KeyManagementService>();
            services.AddSingleton<IKekService, KekService>();

            // Servicio de cifrado de videos
            services.AddSingleton<IVideoEncryptionService, VideoEncryptionService>();

            return services;
        }

        public static IServiceCollection AddBusinessServices(this IServiceCollection services)
        {
            // Registrar servicios de negocio como Scoped (trabajan con DbContext)
            services.AddScoped<IAuthService, AuthService>();
            services.AddScoped<IUserService, UserService>();
            services.AddScoped<IVideoService, VideoService>();

            // Servicios de distribución de claves y permisos
            services.AddScoped<IPermissionService, PermissionService>();
            services.AddScoped<IKeyDistributionService, KeyDistributionService>();
            services.AddScoped<IVideoStreamingService, VideoStreamingService>();

            return services;
        }
    }
}
```

```
using Microsoft.AspNetCore.Mvc;
using SecureVideoStreaming.Services.Cryptography.Interfaces;
using System.Text;

namespace SecureVideoStreaming.API.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class CryptoTestController : ControllerBase
    {
        private readonly IChaCha20Poly1305Service _chaChaService;
        private readonly IRsaService _rsaService;
        private readonly IHashService _hashService;
        private readonly IHmacService _hmacService;

        public CryptoTestController(
            IChaCha20Poly1305Service chaChaService,
            IRsaService rsaService,
            IHashService hashService,
            IHmacService hmacService)
        {
            _chaChaService = chaChaService;
            _rsaService = rsaService;
            _hashService = hashService;
            _hmacService = hmacService;
        }

        [HttpGet("test-chacha20")]
        public IActionResult TestChaCha20()
        {
            var plaintext = "Hola, este es un mensaje de prueba con ChaCha20-Poly1305";
            var data = Encoding.UTF8.GetBytes(plaintext);
            var key = _chaChaService.GenerateKey();

            var (ciphertext, nonce, authTag) = _chaChaService.Encrypt(data, key);
            var decrypted = _chaChaService.Decrypt(ciphertext, key, nonce, authTag);
            var decryptedText = Encoding.UTF8.GetString(decrypted);

            return Ok(new
            {
                algorithm = "ChaCha20-Poly1305",
                original = plaintext,
                decrypted = decryptedText,
                success = plaintext == decryptedText,
                ciphertextLength = ciphertext.Length,
                nonceLength = nonce.Length,
                authTagLength = authTag.Length
            });
        }

        [HttpGet("test-rsa")]
        public IActionResult TestRsa()
        {
            var message = "Mensaje secreto";
            var data = Encoding.UTF8.GetBytes(message);

            var (publicKey, privateKey) = _rsaService.GenerateKeyPair(2048);
            var encrypted = _rsaService.Encrypt(data, publicKey);
            var decrypted = _rsaService.Decrypt(encrypted, privateKey);
            var decryptedMessage = Encoding.UTF8.GetString(decrypted);

            var signature = _rsaService.Sign(data, privateKey);
            var isValid = _rsaService.VerifySignature(data, signature, publicKey);

            return Ok(new
            {
                algorithm = "RSA-2048-OAEP",
                original = message,
                decrypted = decryptedMessage,
                encryptionSuccess = message == decryptedMessage,
                signatureValid = isValid,
                encryptedLength = encrypted.Length,
                signatureLength = signature.Length
            });
        }
```

```
        [HttpGet("test-hash")]
        public IActionResult TestHash()
        {
            var data = Encoding.UTF8.GetBytes("Datos para hashear");
            var hash = _hashService.ComputeSha256(data);
            var hashHex = _hashService.ComputeSha256Hex(data);

            var password = "MiContraseñaSegura123!";
            var salt = _hashService.GenerateSalt();
            var derivedKey = _hashService.DeriveKey(password, salt, 100000, 32);

            return Ok(new
            {
                algorithm = "SHA-256 & PBKDF2",
                hashLength = hash.Length,
hashHex = hashHex,
saltLength = salt.Length,
derivedKeyLength = derivedKey.Length,
derivedKeyHex = Convert.ToHexString(derivedKey).ToLowerInvariant()
});
}
    [HttpGet("test-hmac")]
    public IActionResult TestHmac()
    {
        var data = Encoding.UTF8.GetBytes("Datos para autenticar");
        var key = _hmacService.GenerateKey();

        var hmac = _hmacService.ComputeHmac(data, key);
        var hmacHex = _hmacService.ComputeHmacHex(data, key);
        var isValid = _hmacService.VerifyHmac(data, key, hmac);

        return Ok(new
        {
            algorithm = "HMAC-SHA256",
            hmacLength = hmac.Length,
            hmacHex = hmacHex,
            verificationSuccess = isValid,
            keyLength = key.Length
        });
    }

    [HttpGet("test-all")]
    public IActionResult TestAll()
    {
        try
        {
            // Test ChaCha20
            var plaintext = "Mensaje de prueba completo";
            var data = Encoding.UTF8.GetBytes(plaintext);
            var chachaKey = _chaChaService.GenerateKey();
            var (ciphertext, nonce, authTag) = _chaChaService.Encrypt(data, chachaKey);
            var decrypted = _chaChaService.Decrypt(ciphertext, chachaKey, nonce, authTag);
            var chachaSuccess = Encoding.UTF8.GetString(decrypted) == plaintext;

            // Test RSA
            var (publicKey, privateKey) = _rsaService.GenerateKeyPair(2048);
            var rsaData = Encoding.UTF8.GetBytes("RSA Test");
            var encrypted = _rsaService.Encrypt(rsaData, publicKey);
            var rsaDecrypted = _rsaService.Decrypt(encrypted, privateKey);
            var rsaSuccess = Encoding.UTF8.GetString(rsaDecrypted) == "RSA Test";

            // Test Hash
            var hash = _hashService.ComputeSha256(data);
            var hashSuccess = hash.Length == 32;

            // Test HMAC
            var hmacKey = _hmacService.GenerateKey();
            var hmac = _hmacService.ComputeHmac(data, hmacKey);
            var hmacSuccess = _hmacService.VerifyHmac(data, hmacKey, hmac);

            return Ok(new
            {
                message = "Todas las pruebas criptográficas completadas",
                results = new
                {
                    chaCha20Poly1305 = new { success = chachaSuccess, status = "✓" },
                    rsa = new { success = rsaSuccess, status = "✓" },
                    sha256 = new { success = hashSuccess, status = "✓" },
                    hmac = new { success = hmacSuccess, status = "✓" },
                },
                allTestsPassed = chachaSuccess && rsaSuccess && hashSuccess && hmacSuccess
            });
        }
        catch (Exception ex)
        {
            return StatusCode(500, new
            {
                error = "Error en las pruebas criptográficas",
                message = ex.Message,
                stackTrace = ex.StackTrace
            });
        }
    }
}
}
```

2. Week 2 Deliveries

In this section I will explain every module built to elaborate key management module that corresponds to my individual contribution.

a. Key Management Module

Since in the project analysis we defined a service and controller architecture, to resolve key management, I created its service file and interface, where in the first, I did functions to get private and public keys generated by the server that are parameters of Key Encryption Key process, check if these keys exists on the server or save them, derive an authentication message (HMAC) from the user private key.

```
using SecureVideoStreaming.Services.Cryptography.Interfaces;
using System.Security.Cryptography;
using System.Text;

namespace SecureVideoStreaming.Services.Cryptography.Implementations
{
    public class KeyManagementService : IKeyManagementService
    {
        private readonly string _keysPath;
        private readonly IRsaService _rsaService;
        private readonly IHashService _hashService;
        private const string PrivateKeyFileName = "server_private_key.pem";
        private const string PublicKeyFileName = "server_public_key.pem";
        private const string MasterSecretFileName = ".master_secret";

        public KeyManagementService(IRsaService rsaService, IHashService hashService)
        {
            _rsaService = rsaService;
            _hashService = hashService;

            // Obtener ruta de almacenamiento desde configuración o usar default
            _keysPath = Path.Combine(Directory.GetCurrentDirectory(), "Storage", "Keys");

            // Crear directorio si no existe
            if (!Directory.Exists(_keysPath))
            {
                Directory.CreateDirectory(_keysPath);
            }

            // Inicializar claves del servidor si no existen
            if (!ServerKeysExist())
            {
                GenerateServerKeyPair();
            }
        }

        public string GetServerPrivateKey()
        {
            var privateKeyPath = Path.Combine(_keysPath, PrivateKeyFileName);

            if (!File.Exists(privateKeyPath))
            {
                throw new FileNotFoundException("La clave privada del servidor no existe");
            }

            return File.ReadAllText(privateKeyPath);
        }
```

```
        public string GetServerPublicKey()
        {
            var publicKeyPath = Path.Combine(_keysPath, PublicKeyFileName);

            if (!File.Exists(publicKeyPath))
            {
                throw new FileNotFoundException("La clave pública del servidor no existe");
            }

            return File.ReadAllText(publicKeyPath);
        }

        public void GenerateServerKeyPair(int keySize = 4096)
        {
            var (publicKey, privateKey) = _rsaService.GenerateKeyPair(keySize);
            SaveServerKeys(publicKey, privateKey);

            // Generar y guardar master secret para derivación de claves HMAC
            GenerateMasterSecret();
        }

        public bool ServerKeysExist()
        {
            var privateKeyPath = Path.Combine(_keysPath, PrivateKeyFileName);
            var publicKeyPath = Path.Combine(_keysPath, PublicKeyFileName);

            return File.Exists(privateKeyPath) && File.Exists(publicKeyPath);
        }

        public void SaveServerKeys(string publicKey, string privateKey)
        {
            var privateKeyPath = Path.Combine(_keysPath, PrivateKeyFileName);
            var publicKeyPath = Path.Combine(_keysPath, PublicKeyFileName);

            // Guardar clave privada con permisos restrictivos
            File.WriteAllText(privateKeyPath, privateKey);

            // Guardar clave pública
            File.WriteAllText(publicKeyPath, publicKey);

            // En Windows, establecer permisos restrictivos (solo lectura para el propietario)
            try
            {
                var fileInfo = new FileInfo(privateKeyPath);
                fileInfo.IsReadOnly = false; // Asegurar que podemos modificar permisos
            }
            catch
            {
                // Ignorar errores de permisos en entornos donde no se pueden modificar
            }
        }
```

```
        public byte[] DeriveHmacKeyForUser(int userId, string userEmail)
        {
            // Obtener master secret
            var masterSecret = GetOrCreateMasterSecret();

            // Crear datos de entrada únicos para cada usuario
            var userData = Encoding.UTF8.GetBytes($"HMAC_KEY_USER_{userId}_{userEmail}");

            // Derivar clave HMAC usando PBKDF2 con el master secret como password
            // Usamos el userId y email como salt adicional para unicidad
            var salt = _hashService.ComputeSha256(userData);

            var hmacKey = _hashService.DeriveKey(
                Convert.ToBase64String(masterSecret),
                salt,
                iterations: 210000, // OWASP recomienda 210,000+ para PBKDF2-SHA256
                keyLength: 64 // 512 bits para HMAC-SHA256
            );

            return hmacKey;
        }

        private void GenerateMasterSecret()
        {
            var masterSecretPath = Path.Combine(_keysPath, MasterSecretFileName);

            // Generar 256 bits (32 bytes) de entropía criptográfica
            var masterSecret = RandomNumberGenerator.GetBytes(32);

            // Guardar en formato Base64
            File.WriteAllText(masterSecretPath, Convert.ToBase64String(masterSecret));
        }

        private byte[] GetOrCreateMasterSecret()
        {
            var masterSecretPath = Path.Combine(_keysPath, MasterSecretFileName);

            if (!File.Exists(masterSecretPath))
            {
                GenerateMasterSecret();
            }

            var masterSecretB64 = File.ReadAllText(masterSecretPath);
            return Convert.FromBase64String(masterSecretB64);
        }
    }
}
```

```
namespace SecureVideoStreaming.Services.Cryptography.Interfaces
{
    /// <summary>
    /// Servicio para gestión segura de claves privadas del servidor
    /// </summary>
    public interface IKeyManagementService
    {
        /// <summary>
        /// Obtiene la clave privada RSA del servidor
        /// </summary>
        string GetServerPrivateKey();

        /// <summary>
        /// Obtiene la clave pública RSA del servidor
        /// </summary>
        string GetServerPublicKey();

        /// <summary>
        /// Genera y almacena un nuevo par de claves RSA para el servidor
        /// </summary>
        /// <param name="keySize">Tamaño de la clave (2048 o 4096)</param>
        void GenerateServerKeyPair(int keySize = 4096);

        /// <summary>
        /// Verifica si existen las claves del servidor
        /// </summary>
        bool ServerKeysExist();

        /// <summary>
        /// ALmacena de forma segura las claves del servidor en disco
        /// </summary>
        void SaveServerKeys(string publicKey, string privateKey);

        /// <summary>
        /// Deriva una clave HMAC única para un usuario específico
        /// </summary>
        /// <param name="userId">ID del usuario</param>
        /// <param name="userEmail">Email del usuario para mayor entropía</param>
        byte[] DeriveHmacKeyForUser(int userId, string userEmail);
    }
}
```

Also, I created Key encryption key process to create a kek secure management, in the way to generate a double security layer to protect the existing keys on this project, so in this file we can see encryption and decryption kek process.

```
using SecureVideoStreaming.Services.Cryptography.Interfaces;
using System.Security.Cryptography;

namespace SecureVideoStreaming.Services.Cryptography.Implementations
{
    public class KekService : IKekService
    {
        private readonly IRsaService _rsaService;

        public KekService(IRsaService rsaService)
        {
            _rsaService = rsaService;
        }

        public byte[] GenerateKek()
        {
            // Generar KEK de 256 bits (32 bytes) para ChaCha20-Poly1305
            return RandomNumberGenerator.GetBytes(32);
        }

        public byte[] EncryptKek(byte[] kek, string serverPublicKey)
        {
            if (kek == null || kek.Length != 32)
            {
                throw new ArgumentException("La KEK debe tener exactamente 32 bytes (256 bits)", nameof(kek));
            }

            if (string.IsNullOrWhiteSpace(serverPublicKey))
            {
                throw new ArgumentException("La clave pública del servidor no puede estar vacía", nameof(serverPublicKey));
            }

            return _rsaService.Encrypt(kek, serverPublicKey);
        }
```

```
        public byte[] DecryptKek(byte[] encryptedKek, string serverPrivateKey)
        {
            if (encryptedKek == null || encryptedKek.Length == 0)
            {
                throw new ArgumentException("La KEK cifrada no puede estar vacía", nameof(encryptedKek));
            }

            if (string.IsNullOrWhiteSpace(serverPrivateKey))
            {
                throw new ArgumentException("La clave privada del servidor no puede estar vacía", nameof(serverPrivateKey));
            }

            var kek = _rsaService.Decrypt(encryptedKek, serverPrivateKey);

            if (kek.Length != 32)
            {
                throw new CryptographicException("La KEK descifrada no tiene el tamaño esperado (32 bytes)");
            }

            return kek;
        }

        public (byte[] kek, byte[] encryptedKek) GenerateAndEncryptKek(string serverPublicKey)
        {
            var kek = GenerateKek();
            var encryptedKek = EncryptKek(kek, serverPublicKey);

            return (kek, encryptedKek);
        }
    }
}
```

Finally, I updated authentication service and user service files and same on their controllers, to refer the fixes over key management, adding attributes or parameters to the existing functions, and I did the same on ApplicationDbContext, and ServiceExtensions.

3. Week 3 Deliveries

In this section I will explain every module built to elaborate videos encrypted module and owner videos management module that corresponds to my individual contribution.

a. Videos Encrypted Module

With VideoEncryptionService and its interface, I implemented the encryption process following the next steps:

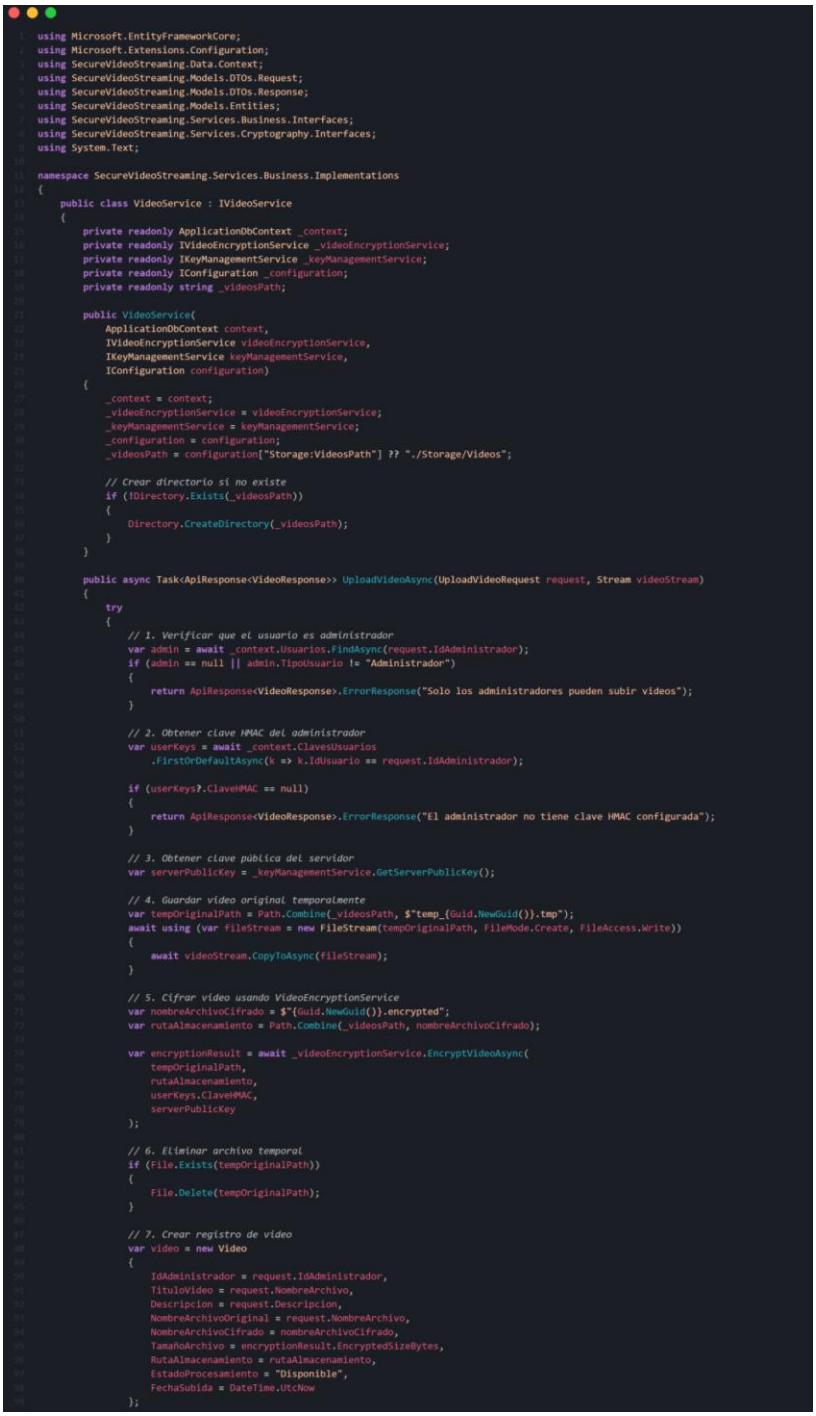
1. KEK (Key Encryption Key) generation: Creates a random 256-bit key using RandomNumberGenerator
2. KEK encryption with RSA-4096 OAEP: Protects the KEK by encrypting it with the server's public key
3. Original Video Hash: Calculates SHA-256 of the video before encrypting it for integrity verification
4. Video Encryption with ChaCha20-Poly1305: Encrypts the content using AEAD (Authenticated Encryption with Associated Data)
5. HMAC generation: Creates an HMAC-SHA256 of the encrypted video using the HMAC key derived from the administrator
6. Metadata storage: Saves encrypted KEK, nonce, authentication tag, hashes in the VideoCryptographicData table

```
using SecureVideoStreaming.Services.Cryptography.Interfaces;
using System.Security.Cryptography;

namespace SecureVideoStreaming.Services.Cryptography.Implementations
{
    public class VideoEncryptionService : IVideoEncryptionService
    {
        private readonly IChaCha20Poly1305Service _chaChaService;
        private readonly IHashService _hashService;
        private readonly IHmacService _hmacService;
        private readonly IKekService _kekService;
        private const int BufferSize = 8192; // 8 KB buffer para procesamiento por bloques

        public VideoEncryptionService(
            IChaCha20Poly1305Service chaChaService,
            IHashService hashService,
            IHmacService hmacService,
            IKekService kekService)
        {
            _chaChaService = chaChaService;
            _hashService = hashService;
            _hmacService = hmacService;
            _kekService = kekService;
        }

        public async Task<VideoEncryptionResult> EncryptVideoAsync(
            string inputFilePath,
            string outputFilePath,
            byte[] hmacKey,
            string serverPublicKey)
        {
            if (!File.Exists(inputFilePath))
            {
                throw new FileNotFoundException("El archivo de video no existe", inputFilePath);
            }

            if (hmacKey == null || hmacKey.Length == 0)
            {
                throw new ArgumentException("La clave HMAC no puede estar vacía", nameof(hmacKey));
            }

            // 1. Calcular hash SHA-256 del video original
            byte[] originalHash;
            using (var fileStream = File.OpenRead(inputFilePath))
            {
                originalHash = _hashService.ComputeSha256(fileStream);
            }

            var originalSize = new FileInfo(inputFilePath).Length;

            // 2. Generar KEK aleatoria de 256 bits
            var kek = _kekService.GenerateKek();

            // 3. Cifrar KEK con RSA del servidor
            var encryptedKek = _kekService.EncryptKek(kek, serverPublicKey);

            // 4. Cifrar el video con ChaCha20-Poly1305
            byte[] nonce;
            byte[] authTag;
            long encryptedSize;

            using (var inputStream = File.OpenRead(inputFilePath))
            using (var outputStream = File.Create(outputFilePath))
            {
                // Leer todo el archivo en memoria para cifrado
                // Nota: Para archivos muy grandes, se debería implementar cifrado por bloques
                var videoData = new byte[inputStream.Length];
                await inputStream.ReadAsync(videoData, 0, videoData.Length);

                // Cifrar con ChaCha20-Poly1305
                var (ciphertext, nonceGenerated, authTagGenerated) = _chaChaService.Encrypt(
                    videoData,
                    kek,
                    nonce: null, // Generar nonce automáticamente
                    associatedData: null
                );

                nonce = nonceGenerated;
                authTag = authTagGenerated;

                // Escribir datos cifrados
                await outputStream.WriteAsync(ciphertext, 0, ciphertext.Length);
                encryptedSize = ciphertext.Length;
            }

            // 5. Calcular HMAC del video cifrado con clave del administrador
            byte[] hmacOfEncryptedVideo;
            using (var encryptedStream = File.OpenRead(outputFilePath))
            {
                hmacOfEncryptedVideo = _hmacService.ComputeHmac(encryptedStream, hmacKey);
            }

            return new VideoEncryptionResult
            {
                EncryptedKek = encryptedKek,
                Nonce = nonce,
                AuthTag = authTag,
                OriginalHash = originalHash,
                HmacOfEncryptedVideo = hmacOfEncryptedVideo,
                OriginalSizeBytes = originalSize,
                EncryptedSizeBytes = encryptedSize
            };
        }
```

```
        public async Task DecryptVideoAsync(
            string encryptedFilePath,
            string outputFilePath,
            byte[] kek,
            byte[] nonce,
            byte[] authTag)
        {
            if (!File.Exists(encryptedFilePath))
            {
                throw new FileNotFoundException("El archivo cifrado no existe", encryptedFilePath);
            }

            if (kek == null || kek.Length != 32)
            {
                throw new ArgumentException("La KEK debe tener 32 bytes", nameof(kek));
            }

            if (nonce == null || nonce.Length != 12)
            {
                throw new ArgumentException("El nonce debe tener 12 bytes", nameof(nonce));
            }

            if (authTag == null || authTag.Length != 16)
            {
                throw new ArgumentException("El authTag debe tener 16 bytes", nameof(authTag));
            }

            using (var inputStream = File.OpenRead(encryptedFilePath))
            using (var outputStream = File.Create(outputFilePath))
            {
                // Leer datos cifrados
                var encryptedData = new byte[inputStream.Length];
                await inputStream.ReadAsync(encryptedData, 0, encryptedData.Length);

                // Descifrar con ChaCha20-Poly1305
                var plaintext = _chaChaService.Decrypt(
                    encryptedData,
                    kek,
                    nonce,
                    authTag,
                    associatedData: null
                );

                // Escribir datos descifrados
                await outputStream.WriteAsync(plaintext, 0, plaintext.Length);
            }
        }

        public bool VerifyVideoIntegrity(
            string encryptedFilePath,
            byte[] expectedHmac,
            byte[] hmacKey)
        {
            if (!File.Exists(encryptedFilePath))
            {
                throw new FileNotFoundException("El archivo cifrado no existe", encryptedFilePath);
            }

            if (expectedHmac == null || expectedHmac.Length == 0)
            {
                throw new ArgumentException("El HMAC esperado no puede estar vacío", nameof(expectedHmac));
            }

            if (hmacKey == null || hmacKey.Length == 0)
            {
                throw new ArgumentException("La clave HMAC no puede estar vacía", nameof(hmacKey));
            }

            using (var fileStream = File.OpenRead(encryptedFilePath))
            {
                var computedHmac = _hmacService.ComputeHmac(fileStream, hmacKey);
                return CryptographicOperations.FixedTimeEquals(computedHmac, expectedHmac);
            }
        }

        public byte[] CalculateFileHash(string filePath)
        {
            if (!File.Exists(filePath))
            {
                throw new FileNotFoundException("El archivo no existe", filePath);
            }

            using (var fileStream = File.OpenRead(filePath))
            {
                return _hashService.ComputeSha256(fileStream);
            }
        }
    }
}
```

b. Owner Videos Management Module

With VideoService, I managed the owner videos and with VideosController, I implemented the endpoints to communicate owner and consumer users with our server, UploadVideoFromRequest to define the parameters to receive from Swagger, VideoResponse to define the data video on the server answer, VideoIntegrityResponse to check the data video validation response, and UpdateVideoMetadataRequest to manage updates on data videos from owner user.



```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using SecureVideoStreaming.Data.Context;
using SecureVideoStreaming.Models.DTOs.Request;
using SecureVideoStreaming.Models.DTOs.Response;
using SecureVideoStreaming.Models.Entities;
using SecureVideoStreaming.Services.Business.Interfaces;
using SecureVideoStreaming.Services.Cryptography.Interfaces;
using System.Text;

namespace SecureVideoStreaming.Services.Business.Implementations
{
    public class VideoService : IVideoService
    {
        private readonly ApplicationDbContext _context;
        private readonly IVideoEncryptionService _videoEncryptionService;
        private readonly IKeyManagementService _keyManagementService;
        private readonly IConfiguration _configuration;
        private readonly string _videosPath;

        public VideoService(
            ApplicationDbContext context,
            IVideoEncryptionService videoEncryptionService,
            IKeyManagementService keyManagementService,
            IConfiguration configuration)
        {
            _context = context;
            _videoEncryptionService = videoEncryptionService;
            _keyManagementService = keyManagementService;
            _configuration = configuration;
            _videosPath = configuration["Storage:VideosPath"] ?? "./Storage/Videos";

            // Crear directorio si no existe
            if (!Directory.Exists(_videosPath))
            {
                Directory.CreateDirectory(_videosPath);
            }
        }

        public async Task<ApiResponse<VideoResponse>> UploadVideoAsync(UploadVideoRequest request, Stream videoStream)
        {
            try
            {
                // 1. Verificar que el usuario es administrador
                var admin = await _context.Usuarios.FindAsync(request.IdAdministrador);
                if (admin == null || admin.TipoUsuario != "Administrador")
                {
                    return ApiResponse<VideoResponse>.ErrorResponse("Solo los administradores pueden subir videos");
                }

                // 2. Obtener clave HMAC del administrador
                var userKeys = await _context.ClavesUsuarios
                    .FirstOrDefaultAsync(k => k.IdUsuario == request.IdAdministrador);

                if (userKeys?.ClaveHMAC == null)
                {
                    return ApiResponse<VideoResponse>.ErrorResponse("El administrador no tiene clave HMAC configurada");
                }

                // 3. Obtener clave pública del servidor
                var serverPublicKey = _keyManagementService.GetServerPublicKey();

                // 4. Guardar video original temporalmente
                var tempOriginalPath = Path.Combine(_videosPath, $"temp_{Guid.NewGuid()}.tmp");
                await using (var fileStream = new FileStream(tempOriginalPath, FileMode.Create, FileAccess.Write))
                {
                    await videoStream.CopyToAsync(fileStream);
                }

                // 5. Cifrar video usando VideoEncryptionService
                var nombreArchivoCifrado = $"{Guid.NewGuid()}.encrypted";
                var rutaAlmacenamiento = Path.Combine(_videosPath, nombreArchivoCifrado);

                var encryptionResult = await _videoEncryptionService.EncryptVideoAsync(
                    tempOriginalPath,
                    rutaAlmacenamiento,
                    userKeys.ClaveHMAC,
                    serverPublicKey
                );

                // 6. Eliminar archivo temporal
                if (File.Exists(tempOriginalPath))
                {
                    File.Delete(tempOriginalPath);
                }

                // 7. Crear registro de video
                var video = new Video
                {
                    IdAdministrador = request.IdAdministrador,
                    TituloVideo = request.NombreArchivo,
                    Descripcion = request.Descripcion,
                    NombreArchivoOriginal = request.NombreArchivo,
                    NombreArchivoCifrado = nombreArchivoCifrado,
                    TamañoArchivo = encryptionResult.EncryptedSizeBytes,
                    RutaAlmacenamiento = rutaAlmacenamiento,
                    EstadoProcesamiento = "Disponible",
                    FechaSubida = DateTime.UtcNow
                };
```

```
                _context.Videos.Add(video);
                await _context.SaveChangesAsync();

                // 8. Crear registro de datos criptográficos
                var cryptoData = new CryptoData
                {
                    IdVideo = video.IdVideo,
                    KEKCifrada = encryptionResult.EncryptedKek,
                    AlgoritmoKEK = "RSA-OAEP",
                    Nonce = encryptionResult.Nonce,
                    AuthTag = encryptionResult.AuthTag,
                    HashSHA256Original = encryptionResult.OriginalHash,
                    HMACDelVideo = encryptionResult.HmacOfEncryptedVideo,
                    FechaGeneracionClaves = DateTime.UtcNow,
                    VersionAlgoritmo = "1.0"
                };

                _context.DatosCriptograficosVideos.Add(cryptoData);
                await _context.SaveChangesAsync();

                var response = new VideoResponse
                {
                    IdVideo = video.IdVideo,
                    TituloVideo = video.TituloVideo,
                    Descripcion = video.Descripcion,
                    NombreArchivoOriginal = video.NombreArchivoOriginal,
                    TamañoArchivo = video.TamañoArchivo,
                    EstadoProcesamiento = video.EstadoProcesamiento,
                    FechaSubida = video.FechaSubida,
                    IdAdministrador = request.IdAdministrador,
                    NombreAdministrador = admin.NombreUsuario,
                    Message = "Video subido y cifrado exitosamente"
                };

                return ApiResponse<VideoResponse>.SuccessResponse(response, "Video subido exitosamente");
            }
            catch (Exception ex)
            {
                return ApiResponse<VideoResponse>.ErrorResponse($"Error al subir video: {ex.Message}");
            }
        }

        public async Task<ApiResponse<List<VideoListResponse>>> GetAllVideosAsync()
        {
            try
            {
                var videos = await _context.Videos
                    .Include(v => v.Administrador)
                    .Where(v => v.EstadoProcesamiento == "Disponible")
                    .OrderByDescending(v => v.FechaSubida)
                    .ToListAsync();

                var videoList = videos.Select(v => new VideoListResponse
                {
                    IdVideo = v.IdVideo,
                    TituloVideo = v.TituloVideo,
                    Descripcion = v.Descripcion,
                    TamañoArchivo = v.TamañoArchivo,
                    Duracion = v.Duracion,
                    FormatoVideo = v.FormatoVideo,
                    EstadoProcesamiento = v.EstadoProcesamiento,
                    FechaSubida = v.FechaSubida,
                    NombreAdministrador = v.Administrador.NombreUsuario
                }).ToList();

                return ApiResponse<List<VideoListResponse>>.SuccessResponse(videoList);
            }
            catch (Exception ex)
            {
                return ApiResponse<List<VideoListResponse>>.ErrorResponse($"Error al obtener videos: {ex.Message}");
            }
        }

        public async Task<ApiResponse<List<VideoListResponse>>> GetVideosByAdminAsync(int adminId)
        {
            try
            {
                var videos = await _context.Videos
                    .Include(v => v.Administrador)
                    .Where(v => v.IdAdministrador == adminId && v.EstadoProcesamiento != "Eliminado")
                    .OrderByDescending(v => v.FechaSubida)
                    .ToListAsync();

                var videoList = videos.Select(v => new VideoListResponse
                {
                    IdVideo = v.IdVideo,
                    TituloVideo = v.TituloVideo,
                    Descripcion = v.Descripcion,
                    TamañoArchivo = v.TamañoArchivo,
                    Duracion = v.Duracion,
                    FormatoVideo = v.FormatoVideo,
                    EstadoProcesamiento = v.EstadoProcesamiento,
                    FechaSubida = v.FechaSubida,
                    NombreAdministrador = v.Administrador.NombreUsuario
                }).ToList();

                return ApiResponse<List<VideoListResponse>>.SuccessResponse(videoList);
            }
            catch (Exception ex)
            {
                return ApiResponse<List<VideoListResponse>>.ErrorResponse($"Error al obtener videos: {ex.Message}");
            }
        }
```

```
        public async Task<ApiResponse<VideoResponse>> GetVideoByIdAsync(int videoId)
        {
            try
            {
                var video = await _context.Videos
                    .Include(v => v.Administrador)
                    .FirstOrDefaultAsync(v => v.IdVideo == videoId);

                if (video == null)
                {
                    return ApiResponse<VideoResponse>.ErrorResponse("Video no encontrado");
                }

                var response = new VideoResponse
                {
                    IdVideo = video.IdVideo,
                    TituloVideo = video.TituloVideo,
                    Descripcion = video.Descripcion,
                    NombreArchivoOriginal = video.NombreArchivoOriginal,
                    TamañoArchivo = video.TamañoArchivo,
                    Duracion = video.Duracion,
                    FormatoVideo = video.FormatoVideo,
                    EstadoProcesamiento = video.EstadoProcesamiento,
                    FechaSubida = video.FechaSubida,
                    IdAdministrador = video.IdAdministrador,
                    NombreAdministrador = video.Administrador.NombreUsuario
                };

                return ApiResponse<VideoResponse>.SuccessResponse(response);
            }
            catch (Exception ex)
            {
                return ApiResponse<VideoResponse>.ErrorResponse($"Error al obtener video: {ex.Message}");
            }
        }

        public async Task<ApiResponse<bool>> DeleteVideoAsync(int videoId, int adminId)
        {
            try
            {
                var video = await _context.Videos.FindAsync(videoId);

                if (video == null)
                {
                    return ApiResponse<bool>.ErrorResponse("Video no encontrado");
                }

                // Verificar que el administrador es el dueño del video
                if (video.IdAdministrador != adminId)
                {
                    return ApiResponse<bool>.ErrorResponse("Solo el administrador dueño puede eliminar el video");
                }

                // Soft delete
                video.EstadoProcesamiento = "Eliminado";
                video.FechaModificacion = DateTime.UtcNow;
                await _context.SaveChangesAsync();

                // Opcional: eliminar archivo físico
                // if (File.Exists(video.RutaAlmacenamiento))
                // {
                //     File.Delete(video.RutaAlmacenamiento);
                // }

                return ApiResponse<bool>.SuccessResponse(true, "Video eliminado exitosamente");
            }
            catch (Exception ex)
            {
                return ApiResponse<bool>.ErrorResponse($"Error al eliminar video: {ex.Message}");
            }
        }

        public async Task<ApiResponse<VideoIntegrityResponse>> VerifyVideoIntegrityAsync(int videoId, int adminId)
        {
            try
            {
                var video = await _context.Videos.FindAsync(videoId);
                if (video == null)
                {
                    return ApiResponse<VideoIntegrityResponse>.ErrorResponse("Video no encontrado");
                }

                // Verificar que el administrador es el dueño del video
                if (video.IdAdministrador != adminId)
                {
                    return ApiResponse<VideoIntegrityResponse>.ErrorResponse("Solo el administrador dueño puede verificar la integridad");
                }

                // Obtener datos criptográficos
                var cryptoData = await _context.DatosCriptograficosVideos
                    .FirstOrDefaultAsync(c => c.IdVideo == videoId);

                if (cryptoData == null)
                {
                    return ApiResponse<VideoIntegrityResponse>.ErrorResponse("No se encontraron datos criptográficos para este video");
                }
```

```
                // Obtener clave HMAC del administrador
                var userKeys = await _context.ClavesUsuarios
                    .FirstOrDefaultAsync(k => k.IdUsuario == adminId);

                if (userKeys?.ClaveHMAC == null)
                {
                    return ApiResponse<VideoIntegrityResponse>.ErrorResponse("No se encontró la clave HMAC del administrador");
                }

                // Verificar integridad del video
                var isValid = _videoEncryptionService.VerifyVideoIntegrity(
                    video.RutaAlmacenamiento,
                    cryptoData.HMACDelVideo,
                    userKeys.ClaveHMAC
                );

                var response = new VideoIntegrityResponse
                {
                    IdVideo = videoId,
                    TituloVideo = video.TituloVideo,
                    IsValid = isValid,
                    HashSHA256Original = Convert.ToBase64String(cryptoData.HashSHA256Original),
                    FechaVerificacion = DateTime.UtcNow,
                    Message = isValid
                        ? "La integridad del video es válida"
                        : "ALERTA: La integridad del video ha sido comprometida"
                };

                return ApiResponse<VideoIntegrityResponse>.SuccessResponse(
                    response,
                    isValid ? "Verificación exitosa" : "Verificación fallida"
                );
            }
            catch (Exception ex)
            {
                return ApiResponse<VideoIntegrityResponse>.ErrorResponse($"Error al verificar integridad: {ex.Message}");
            }
        }

        public async Task<ApiResponse<VideoResponse>> UpdateVideoMetadataAsync(int videoId, UpdateVideoMetadataRequest request, int adminId)
        {
            try
            {
                var video = await _context.Videos
                    .Include(v => v.Administrador)
                    .FirstOrDefaultAsync(v => v.IdVideo == videoId);

                if (video == null)
                {
                    return ApiResponse<VideoResponse>.ErrorResponse("Video no encontrado");
                }

                // Verificar que el administrador es el dueño del video
                if (video.IdAdministrador != adminId)
                {
                    return ApiResponse<VideoResponse>.ErrorResponse("Solo el administrador dueño puede actualizar el video");
                }

                // Actualizar metadata
                if (!string.IsNullOrWhiteSpace(request.TituloVideo))
                {
                    video.TituloVideo = request.TituloVideo;
                }

                if (!string.IsNullOrWhiteSpace(request.Descripcion))
                {
                    video.Descripcion = request.Descripcion;
                }

                video.FechaModificacion = DateTime.UtcNow;
                await _context.SaveChangesAsync();

                var response = new VideoResponse
                {
                    IdVideo = video.IdVideo,
                    TituloVideo = video.TituloVideo,
                    Descripcion = video.Descripcion,
                    NombreArchivoOriginal = video.NombreArchivoOriginal,
                    TamañoArchivo = video.TamañoArchivo,
                    EstadoProcesamiento = video.EstadoProcesamiento,
                    FechaSubida = video.FechaSubida,
                    IdAdministrador = video.IdAdministrador,
                    NombreAdministrador = video.Administrador.NombreUsuario,
                    Message = "Metadata actualizada exitosamente"
                };

                return ApiResponse<VideoResponse>.SuccessResponse(response, "Metadata actualizada exitosamente");
            }
            catch (Exception ex)
            {
                return ApiResponse<VideoResponse>.ErrorResponse($"Error al actualizar metadata: {ex.Message}");
            }
        }
    }
}
```

4. Week 4 Deliveries

In this section I will explain every module built to elaborate middleware pipeline module and key distribution module that corresponds to my individual contribution.

a. Middleware Pipeline Module

1. Interception: Captures all requests using InvokeAsync()
2. Global Try-Catch: Wraps await _next(context) in a try-catch block
3. Exception Handling: If an error occurs, call HandleExceptionAsync()
4. Contextual Logging: Logs errors with request information:
    a. URL path
    b. HTTP method
    c. Authenticated user
    d. IP address
    e. Timestamp
5. Mapping to HTTP Status: Translates exception types to appropriate HTTP codes
6. JSON Response: Returns consistent ErrorResponse

```
using Microsoft.EntityFrameworkCore;
using System.Net;
using System.Security.Authentication;
using System.Text.Json;

namespace SecureVideoStreaming.API.Middleware
{
    /// <summary>
    /// Middleware global para manejo centralizado de excepciones
    /// </summary>
    public class ErrorHandlingMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<ErrorHandlingMiddleware> _logger;
        private readonly IHostEnvironment _environment;

        public ErrorHandlingMiddleware(
            RequestDelegate next,
            ILogger<ErrorHandlingMiddleware> logger,
            IHostEnvironment environment)
        {
            _next = next;
            _logger = logger;
            _environment = environment;
        }

        public async Task InvokeAsync(HttpContext context)
        {
            try
            {
                await _next(context);
            }
            catch (Exception ex)
            {
                await HandleExceptionAsync(context, ex);
            }
        }

        private async Task HandleExceptionAsync(HttpContext context, Exception exception)
        {
            // Log detallado de la excepción
            _logger.LogError(exception,
                "Excepción no controlada en {Path}. Usuario: {User}, IP: {IP}",
                context.Request.Path,
                context.User?.Identity?.Name ?? "Anónimo",
                context.Connection.RemoteIpAddress?.ToString() ?? "Unknown");

            // Determinar código de estado HTTP y mensaje
            var (statusCode, message, errorType) = MapExceptionToResponse(exception);

            // Configurar respuesta
            context.Response.ContentType = "application/json";
            context.Response.StatusCode = (int)statusCode;

            // Construir respuesta de error
            var errorResponse = new ErrorResponse
            {
                Success = false,
                Message = message,
                ErrorType = errorType,
                StatusCode = (int)statusCode,
                Timestamp = DateTime.UtcNow,
                Path = context.Request.Path,
                Method = context.Request.Method
            };

            // Incluir detalles solo en Development
            if (_environment.IsDevelopment())
            {
                errorResponse.Details = exception.Message;
                errorResponse.StackTrace = exception.StackTrace;
                errorResponse.InnerException = exception.InnerException?.Message;
            }

            // Serializar y enviar respuesta
            var jsonOptions = new JsonSerializerOptions
            {
                PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
                WriteIndented = _environment.IsDevelopment()
            };

            var json = JsonSerializer.Serialize(errorResponse, jsonOptions);
            await context.Response.WriteAsync(json);
        }
```

```
        private (HttpStatusCode statusCode, string message, string errorType) MapExceptionToResponse(Exception exception)
        {
            return exception switch
            {
                // 400 - Bad Request (más específico primero)
                ArgumentNullException => (
                    HttpStatusCode.BadRequest,
                    "Se requiere un parámetro obligatorio",
                    "MissingParameter"
                ),

                ArgumentException => (
                    HttpStatusCode.BadRequest,
                    "Los parámetros proporcionados no son válidos",
                    "ValidationError"
                ),

                InvalidOperationException => (
                    HttpStatusCode.BadRequest,
                    exception.Message,
                    "InvalidOperation"
                ),

                // 401 - Unauthorized
                UnauthorizedAccessException => (
                    HttpStatusCode.Unauthorized,
                    "No tiene autorización para realizar esta operación",
                    "Unauthorized"
                ),

                // 403 - Forbidden
                InvalidCredentialException => (
                    HttpStatusCode.Forbidden,
                    "Credenciales inválidas",
                    "Forbidden"
                ),

                // 404 - Not Found
                KeyNotFoundException => (
                    HttpStatusCode.NotFound,
                    "El recurso solicitado no fue encontrado",
                    "NotFound"
                ),

                FileNotFoundException => (
                    HttpStatusCode.NotFound,
                    "El archivo solicitado no existe",
                    "FileNotFound"
                ),

                // 409 - Conflict
                DbUpdateConcurrencyException => (
                    HttpStatusCode.Conflict,
                    "Conflicto de concurrencia. El recurso fue modificado por otro usuario",
                    "ConcurrencyConflict"
                ),

                // 422 - Unprocessable Entity
                DbUpdateException dbEx when dbEx.InnerException?.Message.Contains("duplicate key") == true => (
                    HttpStatusCode.UnprocessableEntity,
                    "El registro ya existe en la base de datos",
                    "DuplicateKey"
                ),

                DbUpdateException dbEx when dbEx.InnerException?.Message.Contains("FOREIGN KEY constraint") == true => (
                    HttpStatusCode.UnprocessableEntity,
                    "No se puede completar la operación debido a restricciones de integridad referencial",
                    "ForeignKeyConstraint"
                ),

                // 500 - Internal Server Error
                DbUpdateException => (
                    HttpStatusCode.InternalServerError,
                    "Error al actualizar la base de datos",
                    "DatabaseError"
                ),

                IOException => (
                    HttpStatusCode.InternalServerError,
                    "Error al acceder al sistema de archivos",
                    "FileSystemError"
                ),

                // Excepciones criptográficas
                System.Security.Cryptography.CryptographicException => (
                    HttpStatusCode.InternalServerError,
                    "Error en operación criptográfica",
                    "CryptographicError"
                ),

                // Default - 500 Internal Server Error
                _ => (
                    HttpStatusCode.InternalServerError,
                    "Ha ocurrido un error interno en el servidor",
                    "InternalServerError"
                )
            };
        }
    }
```

```
    /// <summary>
    /// Modelo de respuesta de error estandarizado
    /// </summary>
    public class ErrorResponse
    {
        /// <summary>
        /// Indica si la operación fue exitosa (siempre false para errores)
        /// </summary>
        public bool Success { get; set; }

        /// <summary>
        /// Mensaje de error user-friendly
        /// </summary>
        public string Message { get; set; } = string.Empty;

        /// <summary>
        /// Tipo de error (para procesamiento en cliente)
        /// </summary>
        public string ErrorType { get; set; } = string.Empty;

        /// <summary>
        /// Código de estado HTTP
        /// </summary>
        public int StatusCode { get; set; }

        /// <summary>
        /// Timestamp del error (UTC)
        /// </summary>
        public DateTime Timestamp { get; set; }

        /// <summary>
        /// Path del request que causó el error
        /// </summary>
        public string Path { get; set; } = string.Empty;

        /// <summary>
        /// Método HTTP del request
        /// </summary>
        public string Method { get; set; } = string.Empty;

        /// <summary>
        /// Detalles adicionales (solo en Development)
        /// </summary>
        public string? Details { get; set; }

        /// <summary>
        /// Stack trace completo (solo en Development)
        /// </summary>
        public string? StackTrace { get; set; }

        /// <summary>
        /// Mensaje de excepción interna (solo en Development)
        /// </summary>
        public string? InnerException { get; set; }
    }

    /// <summary>
    /// Extension methods para registrar el middleware
    /// </summary>
    public static class ErrorHandlingMiddlewareExtensions
    {
        public static IApplicationBuilder UseErrorHandling(this IApplicationBuilder builder)
        {
            return builder.UseMiddleware<ErrorHandlingMiddleware>();
        }
    }
}
```

```
namespace SecureVideoStreaming.API.Middleware
{
    /// <summary>
    /// Excepciones personalizadas para el dominio de la aplicación
    /// </summary>
    public class VideoNotFoundException : KeyNotFoundException
    {
        public VideoNotFoundException(int videoId)
            : base($"El video con ID {videoId} no fue encontrado")
        {
            VideoId = videoId;
        }

        public int VideoId { get; }
    }

    public class UserNotFoundException : KeyNotFoundException
    {
        public UserNotFoundException(int userId)
            : base($"El usuario con ID {userId} no fue encontrado")
        {
            UserId = userId;
        }

        public UserNotFoundException(string email)
            : base($"El usuario con email '{email}' no fue encontrado")
        {
            Email = email;
        }

        public int? UserId { get; }
        public string? Email { get; }
    }

    public class VideoNotOwnedException : UnauthorizedAccessException
    {
        public VideoNotOwnedException(int videoId, int userId)
            : base($"El usuario {userId} no es propietario del video {videoId}")
        {
            VideoId = videoId;
            UserId = userId;
        }

        public int VideoId { get; }
        public int UserId { get; }
    }

    public class DuplicateEmailException : InvalidOperationException
    {
        public DuplicateEmailException(string email)
            : base($"El email '{email}' ya está registrado")
        {
            Email = email;
        }

        public string Email { get; }
    }

    public class DuplicateUsernameException : InvalidOperationException
    {
        public DuplicateUsernameException(string username)
            : base($"El nombre de usuario '{username}' ya está en uso")
        {
            Username = username;
        }

        public string Username { get; }
    }

    public class InvalidCredentialsException : UnauthorizedAccessException
    {
        public InvalidCredentialsException()
            : base("Email o contraseña incorrectos")
        {
        }
    }
```

```
    public class VideoIntegrityException : InvalidOperationException
    {
        public VideoIntegrityException(int videoId, string reason)
            : base($"La integridad del video {videoId} está comprometida: {reason}")
        {
            VideoId = videoId;
            Reason = reason;
        }

        public int VideoId { get; }
        public string Reason { get; }
    }

    public class CryptoKeyNotFoundException : KeyNotFoundException
    {
        public CryptoKeyNotFoundException(int userId, string keyType)
            : base($"No se encontró la clave {keyType} para el usuario {userId}")
        {
            UserId = userId;
            KeyType = keyType;
        }

        public int UserId { get; }
        public string KeyType { get; }
    }

    public class VideoProcessingException : InvalidOperationException
    {
        public VideoProcessingException(string message, Exception? innerException = null)
            : base($"Error al procesar el video: {message}", innerException)
        {
        }
    }

    public class InsufficientPermissionsException : UnauthorizedAccessException
    {
        public InsufficientPermissionsException(string requiredRole)
            : base($"Se requiere el rol '{requiredRole}' para realizar esta operación")
        {
            RequiredRole = requiredRole;
        }

        public string RequiredRole { get; }
    }

    public class VideoUploadException : InvalidOperationException
    {
        public VideoUploadException(string message, Exception? innerException = null)
            : base($"Error al subir el video: {message}", innerException)
        {
        }
    }

    public class FileStorageException : IOException
    {
        public FileStorageException(string operation, string filePath, Exception? innerException = null)
            : base($"Error al {operation} el archivo '{filePath}'", innerException)
        {
            Operation = operation;
            FilePath = filePath;
        }

        public string Operation { get; }
        public string FilePath { get; }
    }
}
```

b. Key Distribution Module

With PermissionService, I managed access control for videos through request, approval, revocation, and validation workflows, and with KeyDistributionService, I implemented the secure key distribution by decrypting KEK with server's private key and re-encrypting it with consumer's public key ensuring zero-knowledge proof. With VideoStreamingService, I enabled chunked streaming with HTTP Range request support for progressive video download. With KeyDistributionController, I implemented 6 endpoints to handle access requests, permission approvals, key package delivery, and permission management, and with StreamingController, I created the video streaming endpoint with Range support returning HTTP 206 Partial Content responses. For DTOs, I created AccessRequestDto to receive access requests with justification, KeyPackageRequest to receive consumer's public key, KeyPackageResponse to deliver the re-encrypted KEK with nonce and authTag, PermissionResponse to show permission status with expiration and access limits, StreamingTokenResponse to provide temporary streaming tokens with HMAC-SHA256 validation, and ApproveAccessRequest to configure max accesses and expiration dates when approving permissions.

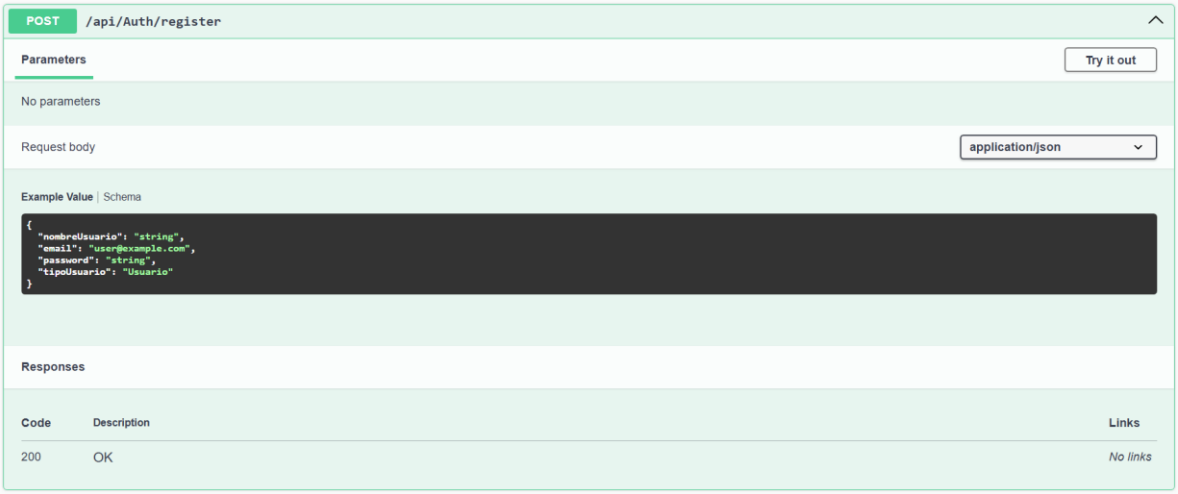
```
using Microsoft.EntityFrameworkCore;
using SecureVideoStreaming.Data.Context;
using SecureVideoStreaming.Models.DTOs.Response;
using SecureVideoStreaming.Services.Business.Interfaces;
using SecureVideoStreaming.Services.Cryptography.Interfaces;
using SecureVideoStreaming.Services.Exceptions;
using System.Security.Cryptography;
using System.Text;

namespace SecureVideoStreaming.Services.Business.Implementations
{
    public class KeyDistributionService : IKeyDistributionService
    {
        private readonly ApplicationDbContext _context;
        private readonly IPermissionService _permissionService;
        private readonly IRsaService _rsaService;
        private readonly IKeyManagementService _keyManagementService;

        public KeyDistributionService(
            ApplicationDbContext context,
            IPermissionService permissionService,
            IRsaService rsaService,
            IKeyManagementService keyManagementService)
        {
            _context = context;
            _permissionService = permissionService;
            _rsaService = rsaService;
            _keyManagementService = keyManagementService;
        }

        public async Task<ApiResponse<KeyPackageResponse>> GetKeyPackageAsync(int videoId, int userId, string userPublicKey)
        {
            try
            {
                // 1. Validar que el usuario tiene permiso activo
                var hasAccessResponse = await _permissionService.HasAccessAsync(videoId, userId);
                if (!hasAccessResponse.Success || !hasAccessResponse.Data)
                {
                    await _permissionService.RegisterAccessAsync(videoId, userId, false, "Permiso denegado o inexistente");
                    return ApiResponse<KeyPackageResponse>.ErrorResponse(
                        hasAccessResponse.Message ?? "No tiene permiso para acceder a este video");
                }

                // 2. Obtener datos criptográficos del video
                var cryptoData = await _context.DatosCriptograficosVideos
                    .FirstOrDefaultAsync(c => c.IdVideo == videoId)
                    ?? throw new KeyNotFoundException($"Datos criptográficos del video {videoId} no encontrados");

                // 3. Descifrar KEK usando la clave privada del servidor
                var serverPrivateKey = _keyManagementService.GetServerPrivateKey();
                var kekBytes = _rsaService.Decrypt(cryptoData.KEKCifrada, serverPrivateKey);

                // 4. Re-cifrar KEK con la clave pública del consumidor
                var rekencryptedKek = _rsaService.Encrypt(kekBytes, userPublicKey);

                // 5. Generar token de streaming
                var streamingToken = await GenerateStreamingTokenAsync(videoId, userId);

                // 6. Incrementar contador de accesos
                var permission = await _context.Permisos
                    .FirstOrDefaultAsync(p => p.IdVideo == videoId && p.IdUsuario == userId && p.TipoPermiso == "Aprobado");

                if (permission != null)
                {
                    await _permissionService.IncrementAccessCountAsync(permission.IdPermiso);
                }
```

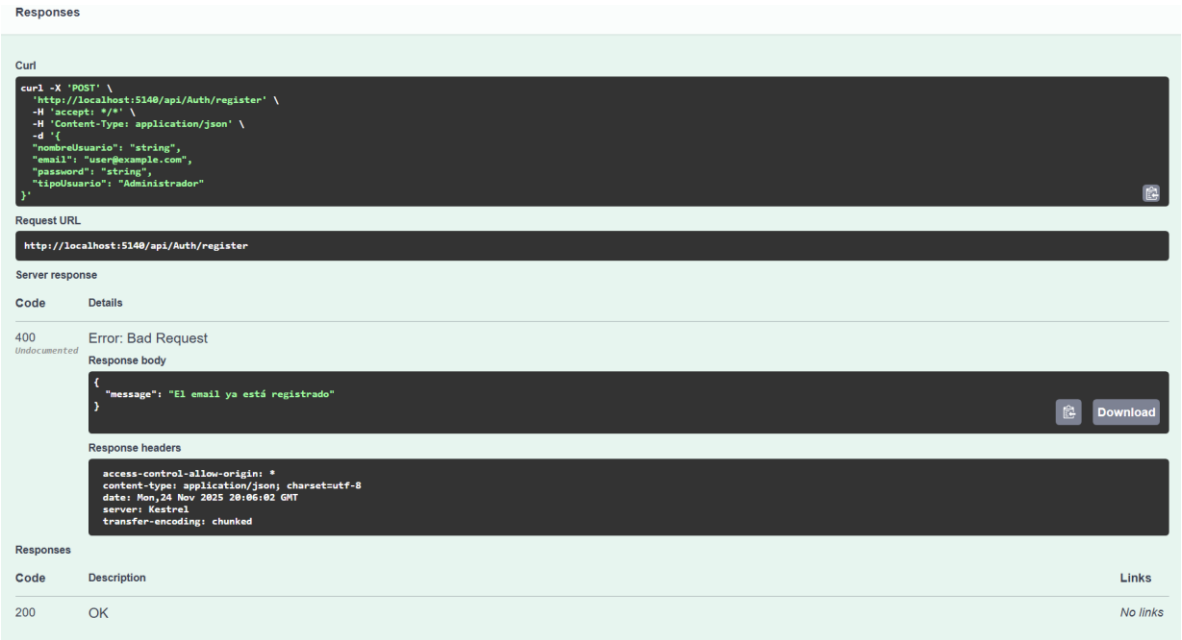
```
                // 7. Registrar acceso exitoso
                await _permissionService.RegisterAccessAsync(videoId, userId, true);

                // 8. Crear respuesta
                var response = new KeyPackageResponse
                {
                    EncryptedKekForUser = Convert.ToBase64String(rekencryptedKek),
                    Nonce = Convert.ToBase64String(cryptoData.Nonce),
                    AuthTag = Convert.ToBase64String(cryptoData.AuthTag),
                    Algorithm = "ChaCha20-Poly1305",
                    VideoId = videoId,
                    StreamingToken = streamingToken.Data?.Token ?? string.Empty,
                    GeneratedAt = DateTime.UtcNow,
                    ExpiresAt = DateTime.UtcNow.AddHours(1)
                };

                return ApiResponse<KeyPackageResponse>.SuccessResponse(
                    response,
                    "Paquete de claves generado exitosamente");
            }
            catch (Exception ex) when (ex is not VideoNotFoundException && ex is not UserNotFoundException)
            {
                await _permissionService.RegisterAccessAsync(videoId, userId, false, ex.Message);
                throw;
            }
        }

        public async Task<ApiResponse<StreamingTokenResponse>> GenerateStreamingTokenAsync(int videoId, int userId)
        {
            // Obtener información del video
            var video = await _context.Videos
                .FirstOrDefaultAsync(v => v.IdVideo == videoId)
                ?? throw new VideoNotFoundException($"Video con ID {videoId} no encontrado");

            // Generar token JWT-like con HMAC-SHA256
            var tokenData = $"{videoId}|{userId}|{DateTime.UtcNow.AddHours(1):O}";
            var key = _keyManagementService.GetServerPrivateKey();
            using var hmac = new HMACSHA256(Encoding.UTF8.GetBytes(key.Substring(0, Math.Min(64, key.Length))));
            var hash = hmac.ComputeHash(Encoding.UTF8.GetBytes(tokenData));
            var token = $"{Convert.ToBase64String(Encoding.UTF8.GetBytes(tokenData))}.{Convert.ToBase64String(hash)}";

            var response = new StreamingTokenResponse
            {
                Token = token,
                VideoId = videoId,
                StreamingUrl = $"/api/streaming/video/{videoId}",
                ExpiresAt = DateTime.UtcNow.AddHours(1),
                FileSizeBytes = video.TamañoArchivo,
                ContentType = "application/octet-stream"
            };

            return ApiResponse<StreamingTokenResponse>.SuccessResponse(response, "Token generado exitosamente");
        }

        public async Task<bool> ValidateStreamingTokenAsync(string token, int videoId, int userId)
        {
            try
            {
                // Separar token en datos y firma
                var parts = token.Split('.');
                if (parts.Length != 2)
                    return false;

                var tokenDataBase64 = parts[0];
                var signatureBase64 = parts[1];

                // Decodificar datos
                var tokenData = Encoding.UTF8.GetString(Convert.FromBase64String(tokenDataBase64));
                var tokenParts = tokenData.Split('|');

                if (tokenParts.Length != 3)
                    return false;

                var tokenVideoId = int.Parse(tokenParts[0]);
                var tokenUserId = int.Parse(tokenParts[1]);
                var expiresAt = DateTime.Parse(tokenParts[2]);

                // Validar contenido
                if (tokenVideoId != videoId || tokenUserId != userId)
                    return false;

                if (expiresAt < DateTime.UtcNow)
                    return false;

                // Validar firma
                var key = _keyManagementService.GetServerPrivateKey();
                using var hmac = new HMACSHA256(Encoding.UTF8.GetBytes(key.Substring(0, Math.Min(64, key.Length))));
                var expectedHash = hmac.ComputeHash(Encoding.UTF8.GetBytes(tokenData));
                var actualHash = Convert.FromBase64String(signatureBase64);

                if (!expectedHash.SequenceEqual(actualHash))
                    return false;

                return true;
            }
            catch
            {
                return false;
            }
        }
    }
}
```

```
using SecureVideoStreaming.Models.DTOs.Response;
using SecureVideoStreaming.Services.Business.Interfaces;

namespace SecureVideoStreaming.Services.Business.Implementations
{
    public class VideoStreamingService : IVideoStreamingService
    {
        public async Task<(Stream stream, long totalSize, long start, long end)> GetVideoChunkAsync(
            string videoPath,
            long rangeStart,
            long? rangeEnd = null)
        {
            // Validar que el archivo existe
            if (!File.Exists(videoPath))
                throw new FileNotFoundException("Video no encontrado", videoPath);

            var fileInfo = new FileInfo(videoPath);
            var totalSize = fileInfo.Length;

            // Ajustar rangeEnd si no está especificado o excede el tamaño
            var actualEnd = rangeEnd ?? totalSize - 1;
            if (actualEnd >= totalSize)
            {
                actualEnd = totalSize - 1;
            }

            // Validar rango
            if (rangeStart < 0 || rangeStart > actualEnd || actualEnd >= totalSize)
            {
                throw new ArgumentException(
                    $"Rango inválido: {rangeStart}-{actualEnd} (tamaño: {totalSize})");
            }

            // Calcular tamaño del chunk
            var chunkSize = actualEnd - rangeStart + 1;

            // Abrir stream y posicionar en el inicio del rango
            var fileStream = new FileStream(videoPath, FileMode.Open, FileAccess.Read, FileShare.Read);
            fileStream.Seek(rangeStart, SeekOrigin.Begin);

            // Crear stream limitado al chunk solicitado
            var limitedStream = new LimitedStream(fileStream, chunkSize);

            return (limitedStream, totalSize, rangeStart, actualEnd);
        }

        public async Task<(long fileSize, string contentType)> GetVideoInfoAsync(string videoPath)
        {
            if (!File.Exists(videoPath))
                throw new FileNotFoundException("Video no encontrado", videoPath);

            var fileInfo = new FileInfo(videoPath);
            var fileSize = fileInfo.Length;
            var contentType = "application/octet-stream"; // Videos cifrados

            return (fileSize, contentType);
        }

        public async Task<bool> ValidateVideoFileAsync(string videoPath)
        {
            if (string.IsNullOrWhiteSpace(videoPath))
                return false;

            if (!File.Exists(videoPath))
                return false;

            var fileInfo = new FileInfo(videoPath);
            if (fileInfo.Length == 0)
                return false;

            try
            {
                // Intentar abrir el archivo para verificar permisos
                using (var fs = File.OpenRead(videoPath))
                {
                    // Solo verificar que se puede abrir
                }
                return true;
            }
            catch
            {
                return false;
            }
        }
```

```
        /// <summary>
        /// Stream auxiliar que limita la lectura a un número especifico de bytes
        /// </summary>
        private class LimitedStream : Stream
        {
            private readonly Stream _baseStream;
            private readonly long _maxLength;
            private long _position;

            public LimitedStream(Stream baseStream, long maxLength)
            {
                _baseStream = baseStream;
                _maxLength = maxLength;
                _position = 0;
            }

            public override bool CanRead => _baseStream.CanRead;
            public override bool CanSeek => false;
            public override bool CanWrite => false;
            public override long Length => _maxLength;
            public override long Position
            {
                get => _position;
                set => throw new NotSupportedException();
            }

            public override int Read(byte[] buffer, int offset, int count)
            {
                var remainingBytes = _maxLength - _position;
                if (remainingBytes <= 0)
                    return 0;

                var bytesToRead = (int)Math.Min(count, remainingBytes);
                var bytesRead = _baseStream.Read(buffer, offset, bytesToRead);
                _position += bytesRead;
                return bytesRead;
            }

            public override async Task<int> ReadAsync(byte[] buffer, int offset, int count, CancellationToken cancellationToken)
            {
                var remainingBytes = _maxLength - _position;
                if (remainingBytes <= 0)
                    return 0;

                var bytesToRead = (int)Math.Min(count, remainingBytes);
                var bytesRead = await _baseStream.ReadAsync(buffer, offset, bytesToRead, cancellationToken);
                _position += bytesRead;
                return bytesRead;
            }

            public override void Flush() => _baseStream.Flush();
            public override long Seek(long offset, SeekOrigin origin) => throw new NotSupportedException();
            public override void SetLength(long value) => throw new NotSupportedException();
            public override void Write(byte[] buffer, int offset, int count) => throw new NotSupportedException();

            protected override void Dispose(bool disposing)
            {
                if (disposing)
                {
                    _baseStream?.Dispose();
                }
                base.Dispose(disposing);
            }
        }
    }
}
```

5. System Working





POST /api/Auth/login

Parameters | Try it out

No parameters

Request body | application/json

Example Value | Schema

```
{
  "email": "user@example.com",
  "password": "string"
}
```

Responses

Code	Description	Links
200	OK	*No links*

Curl

```
curl -X 'POST' \
  'http://localhost:5140/api/Auth/login' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "user@example.com",
  "password": "string"
}'
```

Request URL

```
http://localhost:5140/api/Auth/login
```

Server response

Code	Details
200	Response body

```
{
  "success": true,
  "userId": 1,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxIiwiZW1haWwiOiJ1c2VyQGV4YW1wbGUuY29tIiwiaHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvbmFtZSI6InN0cmluZyIsImh0dHA6Ly9zY2hlbWFzLmlyY3lvc29mdC5jb20vd3MvMjAwOC8wNi9pZGVudGl0eS9jbGFpbXMvcm9sZSI6IlVzdWFyaW8iLCJqdGkiOiJmNDVhNDU4OC1jMzEyLTQ2NmEtOTFlMC0wODE0NjgzMDF1OGUiLCJleHAiOjE3NjQwMTgwMzEsImlzcyI6IlN1Y3VyZVZpZGVvU3RyZWFtaWSnIiwiYXVkIjoiU2VjdXJlVmlkZW9TdHJlYW1pbmdVc2VycyJ9.BEwNQEFteU71X5Vq-2dJGYB4sStG-xccVLA5n3-ik7o",
  "email": "user@example.com",
  "username": "string",
  "userType": "Usuario",
  "message": "Login exitoso"
}
```

Download

Response headers

```
access-control-allow-origin: *
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:07:11 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	*No links*

CryptoTest

GET /api/CryptoTest/test-chacha20

Parameters | Try it out

No parameters

Responses

Code	Description	Links
200	OK	*No links*

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5140/api/CryptoTest/test-chacha20' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5140/api/CryptoTest/test-chacha20
```

Server response

Code	Details
200	Response body

```
{
  "algorithm": "ChaCha20-Poly1305",
  "original": "Hola, este es un mensaje de prueba con ChaCha20-Poly1305",
  "decrypted": "Hola, este es un mensaje de prueba con ChaCha20-Poly1305",
  "success": true,
  "ciphertextLength": 56,
  "nonceLength": 12,
  "authTagLength": 16
}
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:08:15 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	*No links*

GET /api/CryptoTest/test-rsa

Parameters | Try it out

No parameters

Responses

Code	Description	Links
200	OK	*No links*

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5140/api/CryptoTest/test-rsa' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5140/api/CryptoTest/test-rsa
```

Server response

Code	Details
200	Response body

```
{
  "algorithm": "RSA-2048-OAEP",
  "original": "Mensaje secreto",
  "decrypted": "Mensaje secreto",
  "encryptionSuccess": true,
  "signatureValid": true,
  "encryptedLength": 256,
  "signatureLength": 256
}
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:08:53 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	*No links*

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5140/api/CryptoTest/test-hash' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5140/api/CryptoTest/test-hash
```

Server response

Code	Details
200	Response body

```
{
  "algorithm": "SHA-256 & PBKDF2",
  "hashLength": 32,
  "hashHex": "78ae181207140114334 3cab7025a6f993feb23aedd57f569234abbba3554a3021",
  "saltLength": 32,
  "derivedKeyLength": 32,
  "derivedKeyHex": "f9be5b15f0ef650ca7be2247967161093cd241cfcd7df8dd6fd8e3af8eecf45a"
}
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:09:18 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	*No links*

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5140/api/CryptoTest/test-hmac' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5140/api/CryptoTest/test-hmac
```

Server response

Code	Details
200	Response body

```
{
  "algorithm": "HMAC-SHA256",
  "hmacLength": 32,
  "hmacHex": "c3b8872458b5f30246b54752b5a092c702b4761351a80e47ae3ea1608817bc0e",
  "verificationSuccess": true,
  "keyLength": 64
}
```

Download

Response headers

```
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:09:34 GMT
server: Kestrel
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	*No links*

Curl

```
curl -X 'GET' \
  'http://localhost:5140/api/CryptoTest/test-all' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:5140/api/CryptoTest/test-all
```

Server response

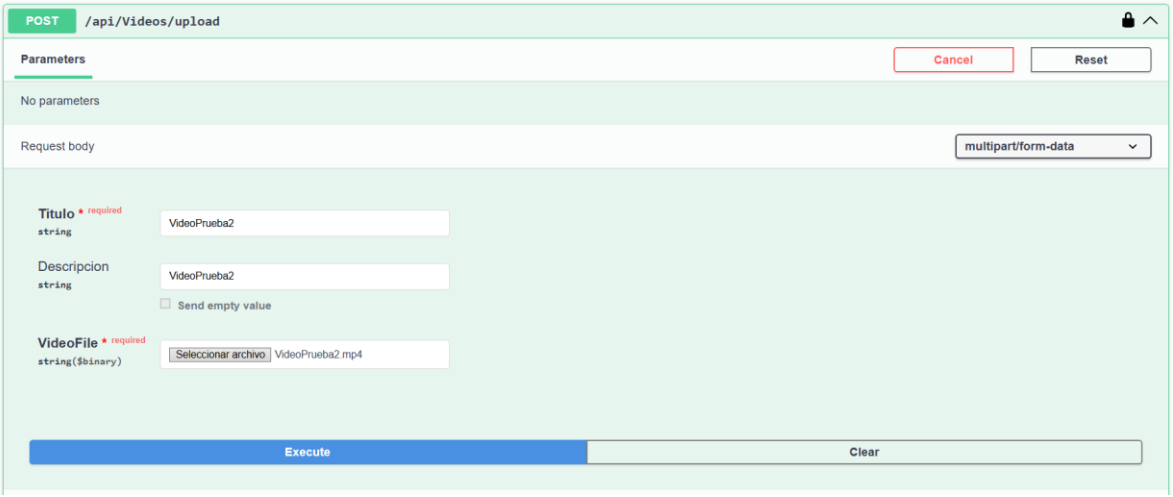
Code	Details
200	Response body

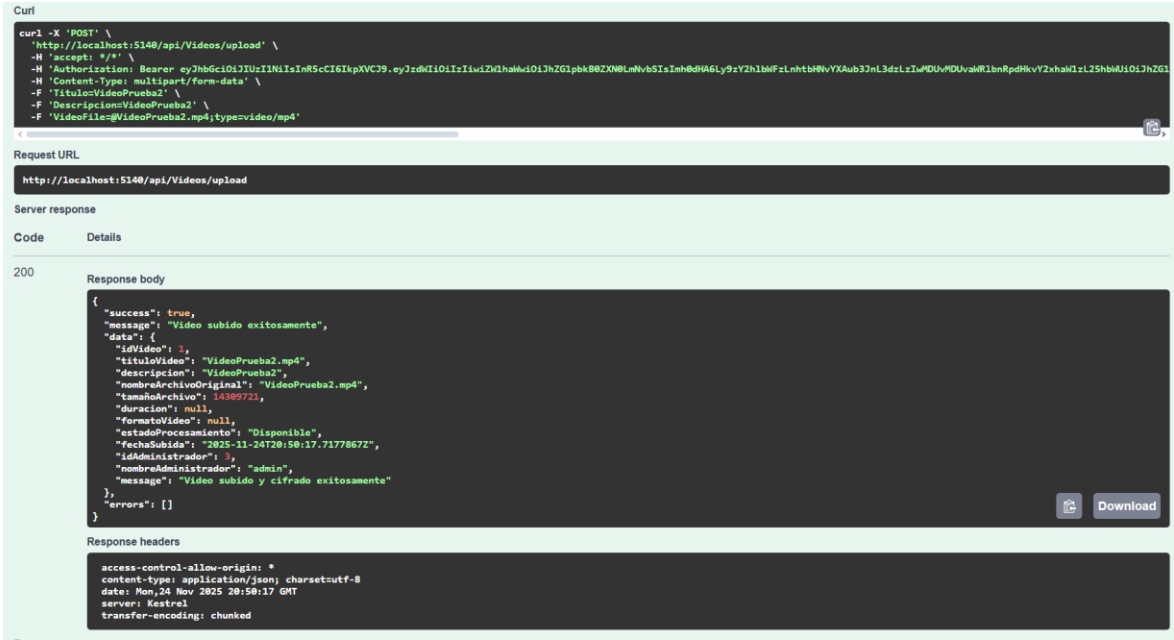

```
{
  "message": "Todas las pruebas criptográficas completadas",
  "results": {
    "chaCha20Poly1305": {
      "success": true,
      "status": "✓"
    },
    "rsa": {
      "success": true,
      "status": "✓"
    },
    "sha256": {
      "success": true,
      "status": "✓"
    },
    "hmac": {
      "success": true,
      "status": "✓"
    }
  },
  "allTestsPassed": true
}
```

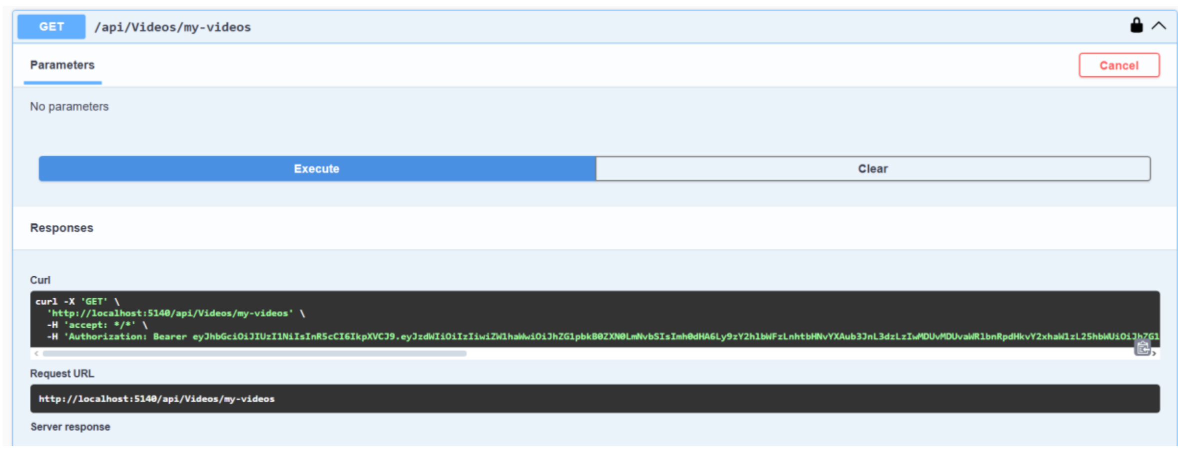

Download

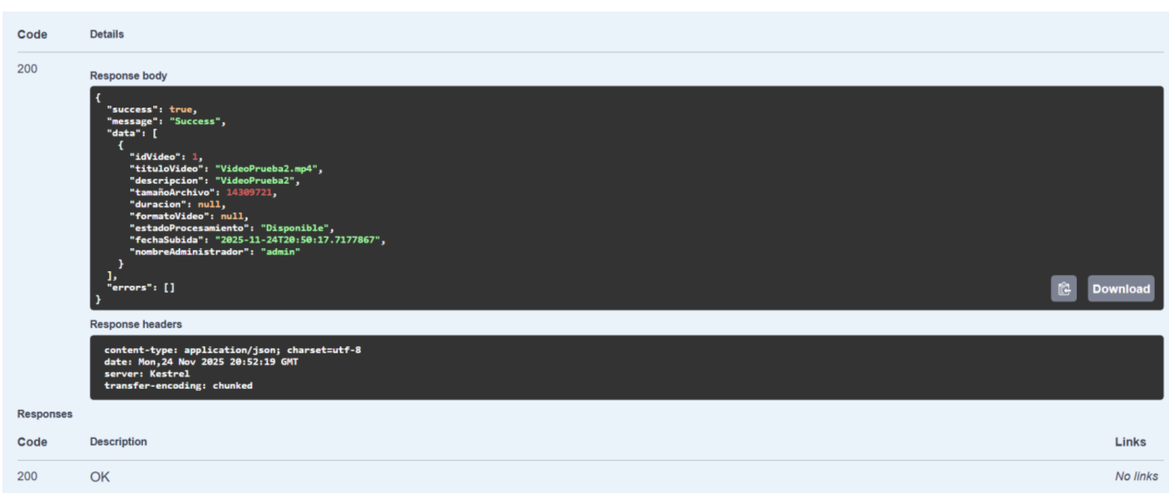
Response headers

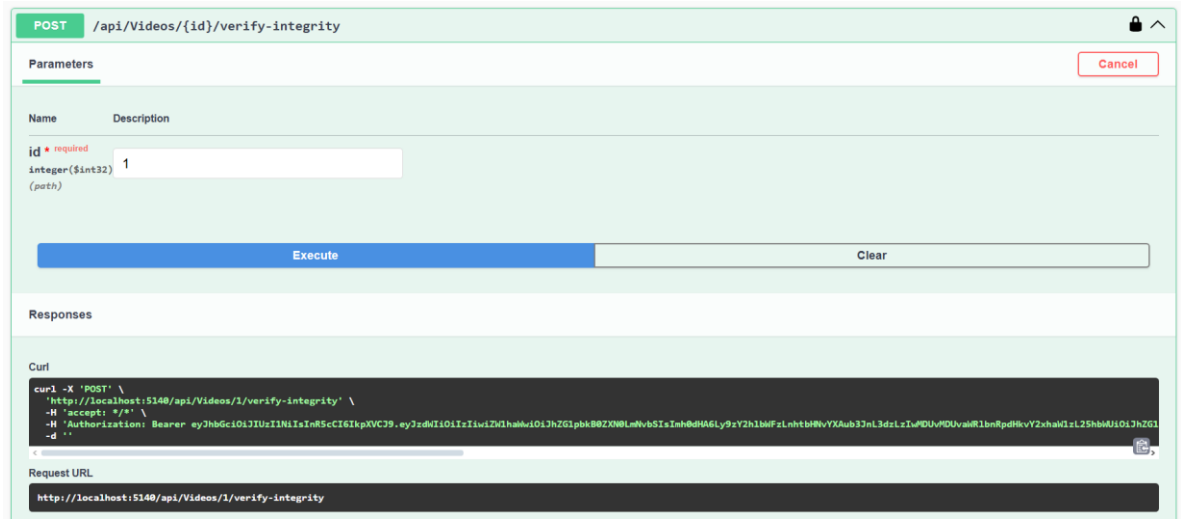
```
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:09:54 GMT
server: Kestrel
transfer-encoding: chunked
```

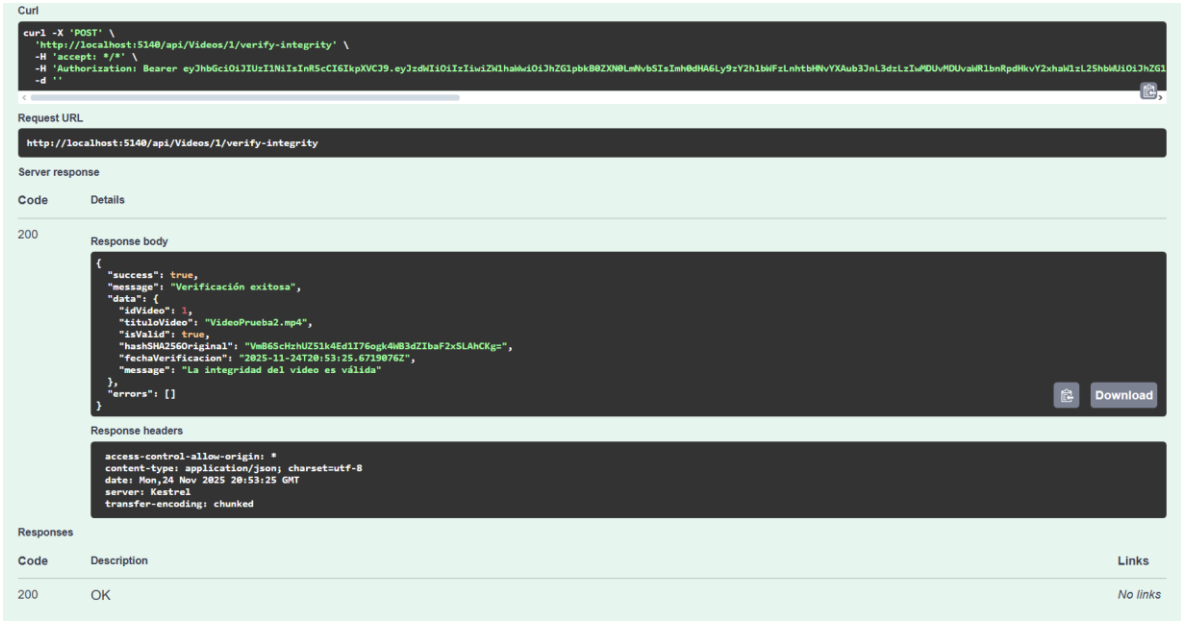


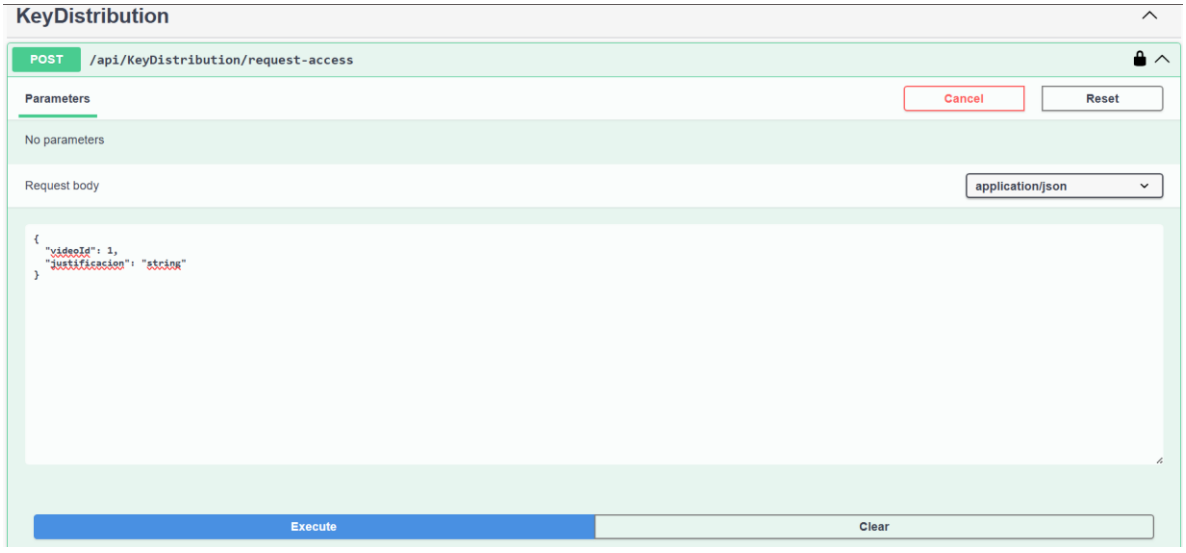

Curl
curl -X 'POST' \
'http://localhost:5140/api/Videos/upload' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIzIiwiZW1haWwiOiJhZG1pbkB0ZXN0LmNvbSIsImh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdHkvY2xhaW1zL25hbWUiOiJhZG1
-H 'Content-Type: multipart/form-data' \
-F 'Titulo=VideoPrueba2' \
-F 'Descripcion=VideoPrueba2' \
-F 'VideoFile=@VideoPrueba2.mp4;type=video/mp4'
Request URL
http://localhost:5140/api/Videos/upload
Server response
Code
Details
200
Response body
{
"success": true,
"message": "Video subido exitosamente",
"data": {
"idVideo": 1,
"tituloVideo": "VideoPrueba2.mp4",
"descripcion": "VideoPrueba2",
"nombreArchivoOriginal": "VideoPrueba2.mp4",
"tamañoArchivo": 14309721,
"duracion": null,
"formatoVideo": null,
"estadoProcesamiento": "Disponible",
"fechaSubida": "2025-11-24T20:50:17.7177867Z",
"idAdministrador": 3,
"nombreAdministrador": "admin",
"message": "Video subido y cifrado exitosamente"
},
"errors": []
}
Download
Response headers
access-control-allow-origin: *
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:50:17 GMT
server: Kestrel
transfer-encoding: chunked

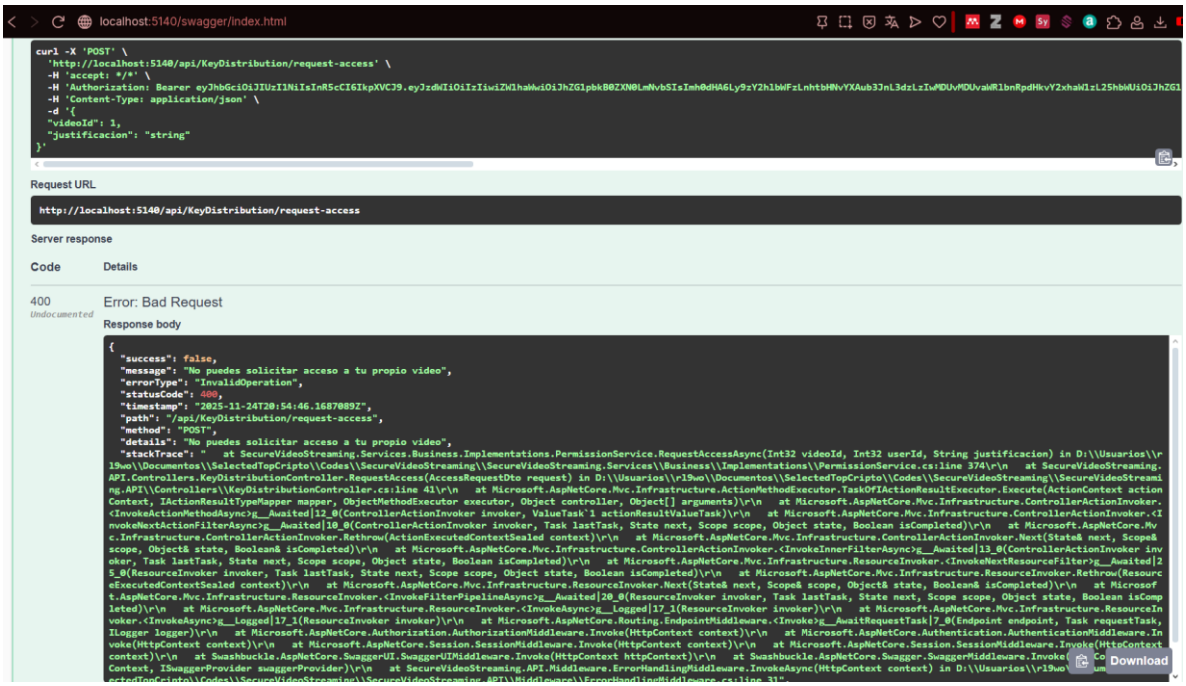

GET
/api/Videos/my-videos
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:5140/api/Videos/my-videos' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIzIiwiZW1haWwiOiJhZG1pbkB0ZXN0LmNvbSIsImh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdHkvY2xhaW1zL25hbWUiOiJhZG1
Request URL
http://localhost:5140/api/Videos/my-videos
Server response


Code
Details
200
Response body
{
"success": true,
"message": "Success",
"data": [
{
"idVideo": 1,
"tituloVideo": "VideoPrueba2.mp4",
"descripcion": "VideoPrueba2",
"tamañoArchivo": 14309721,
"duracion": null,
"formatoVideo": null,
"estadoProcesamiento": "Disponible",
"fechaSubida": "2025-11-24T20:50:17.7177867",
"nombreAdministrador": "admin"
}
],
"errors": []
}
Download
Response headers
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:52:19 GMT
server: Kestrel
transfer-encoding: chunked
Responses
Code
Description
Links
200
OK
No links


POST
/api/Videos/{id}/verify-integrity
Parameters
Cancel
Name
Description
id
required
integer($int32)
(path)
1
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:5140/api/Videos/1/verify-integrity' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIzIiwiZWlhaWwiOiJhZG1pbkB0ZXN0LmNvbSIsImh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdHkvY2xhaW1zL25hbWUiOiJhZG1
-d ''
Request URL
http://localhost:5140/api/Videos/1/verify-integrity


Curl
curl -X 'POST' \
'http://localhost:5140/api/Videos/1/verify-integrity' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIzIiwiZWlhaWwiOiJhZG1pbkB0ZXN0LmNvbSIsImh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdHkvY2xhaW1zL25hbWUiOiJhZG1
-d ''
Request URL
http://localhost:5140/api/Videos/1/verify-integrity
Server response
Code
Details
200
Response body
{
"success": true,
"message": "Verificación exitosa",
"data": {
"idVideo": 1,
"tituloVideo": "VideoPrueba2.mp4",
"isValid": true,
"hashSHA256Original": "VmB6ScHzhUZ51k4Ed1I76ogk4WB3dZIbaF2xSLAhCKg=",
"fechaVerificacion": "2025-11-24T20:53:25.6719076Z",
"message": "La integridad del video es válida"
},
"errors": []
}
Download
Response headers
access-control-allow-origin: *
content-type: application/json; charset=utf-8
date: Mon,24 Nov 2025 20:53:25 GMT
server: Kestrel
transfer-encoding: chunked
Responses
Code
Description
Links
200
OK
No links


KeyDistribution
POST
/api/KeyDistribution/request-access
Parameters
Cancel
Reset
No parameters
Request body
application/json
{
"videoId": 1,
"justificacion": "string"
}
Execute
Clear


localhost:5140/swagger/index.html
curl -X 'POST' \
'http://localhost:5140/api/KeyDistribution/request-access' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIzIiwiZW1haWwiOiJhZG1pbk80ZXN0LmNvbSIsImh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWRlbnRpdHkvY2xhaW1zL25hbWUiOiJhZG1
-H 'Content-Type: application/json' \
-d '{
"videoId": 1,
"justificacion": "string"
}'
Request URL
http://localhost:5140/api/KeyDistribution/request-access
Server response
Code
Details
400
Undocumented
Error: Bad Request
Response body
{
"success": false,
"message": "No puedes solicitar acceso a tu propio video",
"errorType": "InvalidOperation",
"statusCode": 400,
"timestamp": "2025-11-24T20:54:46.1687089Z",
"path": "/api/KeyDistribution/request-access",
"method": "POST",
"details": "No puedes solicitar acceso a tu propio video",
"stackTrace": "   at SecureVideoStreaming.Services.Business.Implementations.PermissionService.RequestAccessAsync(Int32 videoId, Int32 userId, String justificacion) in D:\\Usuarios\\r19wo\\Documentos\\SelectedTopCripto\\Codes\\SecureVideoStreaming\\SecureVideoStreaming.Services\\Business\\Implementations\\PermissionService.cs:line 374\r\n   at SecureVideoStreaming.API.Controllers.KeyDistributionController.RequestAccess(AccessRequestDto request) in D:\\Usuarios\\r19wo\\Documentos\\SelectedTopCripto\\Codes\\SecureVideoStreaming\\SecureVideoStreaming.API\\Controllers\\KeyDistributionController.cs:line 41\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor.TaskOfIActionResultExecutor.Execute(ActionContext actionContext, IActionResultTypeMapper mapper, ObjectMethodExecutor executor, Object controller, Object[] arguments)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeActionMethodAsync>g__Awaited|12_0(ControllerActionInvoker invoker, ValueTask`1 actionResultValueTask)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.<InvokeNextActionFilterAsync>g__Awaited|10_0(ControllerActionInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Rethrow(ActionExecutedContextSealed context)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.Next(State& next, Scope& scope, Object& state, Boolean& isCompleted)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeNextResourceFilter>g__Awaited|25_0(ResourceInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.Rethrow(ResourceExecutedContextSealed context)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.Next(State& next, Scope& scope, Object& state, Boolean& isCompleted)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeFilterPipelineAsync>g__Awaited|20_0(ResourceInvoker invoker, Task lastTask, State next, Scope scope, Object state, Boolean isCompleted)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g__Logged|17_1(ResourceInvoker invoker)\r\n   at Microsoft.AspNetCore.Mvc.Infrastructure.ResourceInvoker.<InvokeAsync>g__Logged|17_1(ResourceInvoker invoker)\r\n   at Microsoft.AspNetCore.Routing.EndpointMiddleware.<Invoke>g__AwaitRequestTask|7_0(Endpoint endpoint, Task requestTask, ILogger logger)\r\n   at Microsoft.AspNetCore.Authorization.AuthorizationMiddleware.Invoke(HttpContext context)\r\n   at Microsoft.AspNetCore.Authentication.AuthenticationMiddleware.Invoke(HttpContext context)\r\n   at Microsoft.AspNetCore.Session.SessionMiddleware.Invoke(HttpContext context)\r\n   at Microsoft.AspNetCore.Session.SessionMiddleware.Invoke(HttpContext context)\r\n   at Swashbuckle.AspNetCore.SwaggerUI.SwaggerUIMiddleware.Invoke(HttpContext httpContext)\r\n   at Swashbuckle.AspNetCore.Swagger.SwaggerMiddleware.Invoke(HttpContext httpContext, ISwaggerProvider swaggerProvider)\r\n   at SecureVideoStreaming.API.Middleware.ErrorHandlingMiddleware.InvokeAsync(HttpContext context) in D:\\Usuarios\\r19wo\\Documentos\\SelectedTopCripto\\Codes\\SecureVideoStreaming\\SecureVideoStreaming.API\\Middleware\\ErrorHandlingMiddleware.cs:line 31"
Download