Gestor de Testamentos Digitales 4 Capas de Seguridad Criptográfica

Integrantes del Equipo

Israel Gómez Bonilla - 2023371154

Orlando Rubio Cabrera - 2020340057

Marco Antonio Gómez Olvera - 2023371124

Sandra Zoé Cabrera Valázques - 2023171054

Ingeniería en Gestión y Desarrollo de Software

Materia: Seguridad Informática

Docente: Brandon Efren Venegas Olvera

Fecha de entrega: 23 de noviembre de 2025

Índice

1. Resumen Ejecutivo

Este documento presenta el proyecto **Gestor de Testamentos Digitales**, un sistema que implementa cuatro capas fundamentales de seguridad criptográfica para garantizar la confidencialidad, integridad, autenticidad y no repudio de testamentos digitales.

1.1. Objetivo

Diseñar e implementar una aplicación funcional que garantice la seguridad de la información aplicando:

- **Capa 1:** Login Seguro con bcrypt
- **Capa 2:** Cifrado de Datos en Reposo con AES-256
- **Capa 3:** Firma Digital con RSA
- **Capa 4:** Cifrado Híbrido para Comunicación Segura

1.2. Resultado

Estado del Proyecto

✓ **100 % Completo y Funcional**

- Todas las capas implementadas
- 5/5 pruebas automatizadas exitosas
- Verificación en base de datos completada
- Documentación completa generada

1.3. Documentación Disponible

- `ARQUITECTURA.md` - Diseño técnico completo y diagramas
- `README.md` - Instalación, uso y tecnologías
- `EVIDENCIA_4_CAPAS.md` - Pruebas y verificaciones detalladas
- `PARA_ENTREGAR.md` - Guía de entrega y checklist

2. Diagrama de Flujo - Arquitectura Completa

2.1. Flujo de las 4 Capas de Seguridad

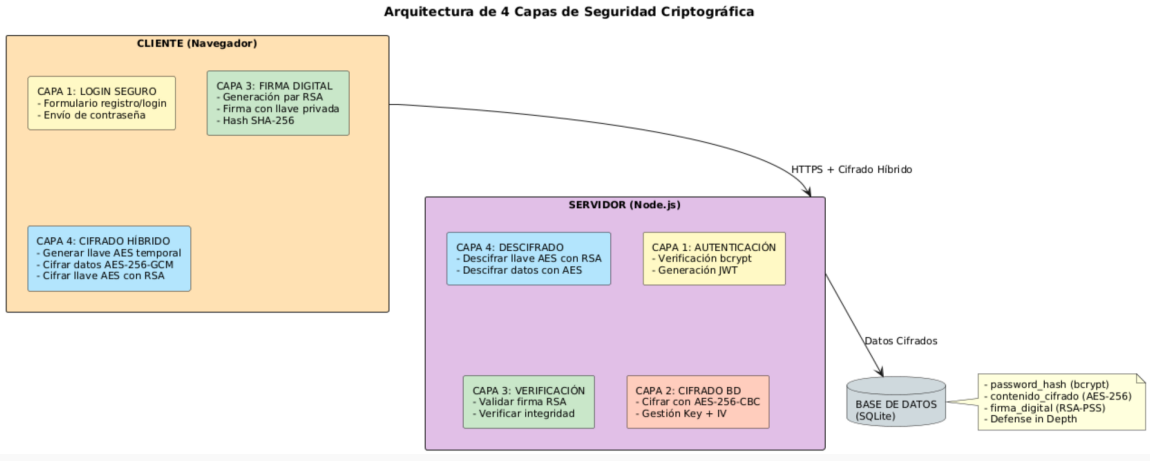

Figura 1: Diagrama de arquitectura de las 4 capas de seguridad criptográfica con Defense in Depth

Explicación del flujo:

1. **CAPA 1 - bcrypt (Autenticación):** La contraseña del usuario se hashea con bcrypt en el servidor usando 10 salt rounds. El hash se almacena de forma segura en la base de datos, garantizando que las contraseñas nunca se guarden en texto plano.

2. **CAPA 2 - AES-256 (Datos en Reposo):** El contenido sensible del testamento y las contraseñas bancarias se cifran con AES-256-CBC utilizando una llave maestra de 256 bits almacenada en variables de entorno. Cada registro utiliza un IV (Vector de Inicialización) único de 16 bytes.

3. **CAPA 3 - RSA (Firma Digital):** El testamento se firma digitalmente con RSA-PSS-2048 usando la llave privada del usuario. Se genera un hash SHA-256 del contenido antes de firmarlo, garantizando autenticidad, integridad y no repudio.

4. **CAPA 4 - Híbrido (Comunicación Segura):** La comunicación cliente-servidor implementa cifrado híbrido: los datos se cifran con AES-256-GCM y la llave AES se cifra con RSA-OAEP-2048 usando la llave pública del servidor. Este "sobre digital"proporciona confidencialidad extremo a extremo.

Principio de Defense in Depth:
Este diseño multicapa garantiza que si una capa es comprometida, las demás continúan protegiendo la información. Cada capa aborda diferentes aspectos de la seguridad (autenticación, confidencialidad, integridad, no repudio), creando un sistema robusto y resiliente.

3. Arquitectura de Seguridad (4 Capas)

3.1. Capa 1: Login Seguro (Autenticación)

3.1.1. Tecnología

- **Algoritmo:** bcrypt
- **Salt Rounds:** 10
- **Biblioteca:** bcrypt 5.1

3.1.2. Implementación

```
async hashPassword(password) {
    const hash = await bcrypt.hash(password, SALT_ROUNDS);
    return hash;
}

async verifyPassword(password, hash) {
    return await bcrypt.compare(password, hash);
}
```

Listing 1: AuthService.js - Hash de Contraseñas

3.1.3. Garantías de Seguridad

- ✓ Contraseñas NUNCA almacenadas en texto plano
- ✓ Salt único por cada contraseña
- ✓ Hash irreversible (one-way function)
- ✓ Protección contra ataques de fuerza bruta

3.2. Capa 2: Cifrado de Datos en Reposo

3.2.1. Tecnología

- **Algoritmo:** AES-256-CBC
- **Llave Maestra:** 32 bytes (256 bits)
- **IV:** 16 bytes (único por registro)

3.2.2. Gestión de Llaves

La llave maestra se almacena de forma segura en variables de entorno (`.env`):

```
DB_ENCRYPTION_KEY=9a59a4246d0f98ca59833884d2f5add2...
```

3.2.3. Campos Cifrados

- Contenido completo del testamento
- Contraseñas de cuentas bancarias
- Información sensible de beneficiarios
- Mensajes póstumos

3.2.4. Implementación

```
encryptTestamentContent(content) {
    const iv = crypto.randomBytes(IV_LENGTH);
    const cipher = crypto.createCipheriv(
        'aes-256-cbc',
        this.masterKey,
        iv
    );
    let encrypted = cipher.update(content, 'utf8', 'hex');
    encrypted += cipher.final('hex');
    return { encrypted, iv: iv.toString('hex') };
}
```

Listing 2: EncryptionService.js - Cifrado AES

3.3. Capa 3: Firma Digital (Autenticidad y No Repudio)

3.3.1. Tecnología

- **Algoritmo:** RSA-PSS-2048
- **Hash:** SHA-256
- **Padding:** PKCS1_PSS_PADDING

3.3.2. Generación de Llaves

- Par de llaves RSA generado al registrar usuario
- **Llave Privada:** Entregada al usuario (debe guardarla)
- **Llave Pública:** Almacenada en base de datos
- **Tamaño:** 2048 bits

3.3.3. Flujo de Firma

1. Usuario redacta testamento
2. Sistema genera hash SHA-256 del contenido
3. Usuario firma con su llave privada RSA
4. Sistema verifica firma con llave pública
5. Si válida, testamento marcado como "firmado"

3.3.4. Garantías de Seguridad

✓ **Autenticidad:** Solo el usuario puede generar la firma

✓ **Integridad:** Hash garantiza no modificación

✓ **No Repudio:** Usuario no puede negar que firmó

3.4. Capa 4: Cifrado Híbrido (Defense in Depth)

3.4.1. Tecnología

- **Asimétrico:** RSA-OAEP-2048
- **Simétrico:** AES-256-GCM
- **Concepto:** Sobre Digital

3.4.2. Flujo del Sobre Digital

Cliente (Cifrar):

1. Genera llave AES temporal (32 bytes)
2. Cifra datos con AES-256-GCM
3. Obtiene llave pública RSA del servidor
4. Cifra llave AES con RSA pública del servidor
5. Envía paquete completo (sobre digital)

Servidor (Descifrar):

1. Descifra llave AES con RSA privada del servidor
2. Descifra datos con AES-256-GCM
3. Verifica authTag (autenticación)
4. Procesa datos originales

3.4.3. Defense in Depth

- Seguridad a nivel de aplicación (independiente de HTTPS)
- Protección extremo a extremo
- Doble capa de cifrado: RSA + AES

4. Resultados de Pruebas

4.1. Pruebas Automatizadas

Se ejecutó el script `test-client.js` que verifica las 4 capas de seguridad:

Prueba	Resultado
TEST 1: Obtener Llave Pública Servidor (CAPA 4)	✓ OK
TEST 2: Registrar Usuario (CAPA 1)	✓ OK
TEST 3: Crear Testamento (CAPA 2 + CAPA 4)	✓ OK
TEST 4: Firmar Testamento (CAPA 3 + CAPA 4)	✓ OK
TEST 5: Verificar en Base de Datos	✓ OK
TOTAL	**5/5 EXITOSAS**

Cuadro 1: Resultados de pruebas automatizadas

4.2. Verificación en Base de Datos

4.2.1. Hash bcrypt (CAPA 1)

```
PS C:\Users\marco\encriptar\testamentos_digitales\backend> sqlite3 database\testamentos.db ".mode column" ".headers on
" "SELECT id, username, email, substr(password_hash, 1, 60) as password_hash_preview FROM usuarios LIMIT 3;"
id  username            email                     password_hash_preview
--  ------------------  ------------------------  ------------------------------------------------------------
1   marco               marco@test.com            $2b$10$FiZMIjvruS8Dr9D5EvLJKeoecwFEzAvcaT9eT5bFAluqx6ktW1JPe
2   test_1763924427071  test1763924427071@test.com  $2b$10$HjUDeYbAR90gOYOdSZ2Pu.FglyKNR.D9PysSX45VlV1xt0mZcnYui
PS C:\Users\marco\encriptar\testamentos_digitales\backend>
```

Figura 2: Verificación de hash bcrypt en base de datos SQLite - Los hashes comienzan con $2b$10$

Análisis de la captura:

`$2b$10$FiZMIjvruS8Dr9D5EvLJKeoecwFEzAvcaT9eT5bFAluqx6ktW1JPe`

✓ Hash comienza con `$2b$10$` (bcrypt con 10 rounds)
✓ Contraseñas NUNCA almacenadas en texto plano
✓ Cada usuario tiene un salt único integrado en el hash

4.2.2. Contenido cifrado AES-256 (CAPA 2)

```
d82adc99ab314012e6c2ea892a1f082e073f513a93940ce495...
IV: 467af8ad23432716f6235892579ff209
```

✓ Contenido completamente ilegible (cifrado correctamente)

4.2.3. Firma digital RSA (CAPA 3)

```
R25VggCXf1NM20Xk0RFPBdSScm+yzYmOLdZl3SR/Tritat5dkV...
Hash: 5a57dde2d3e7df12c8a4c96712e534ca7e8e8352de5ec817...
Estado: firmado
```

✓ Firma digital almacenada y verificada

5. Evidencias Fotográficas

5.1. Captura 1: API Funcionando

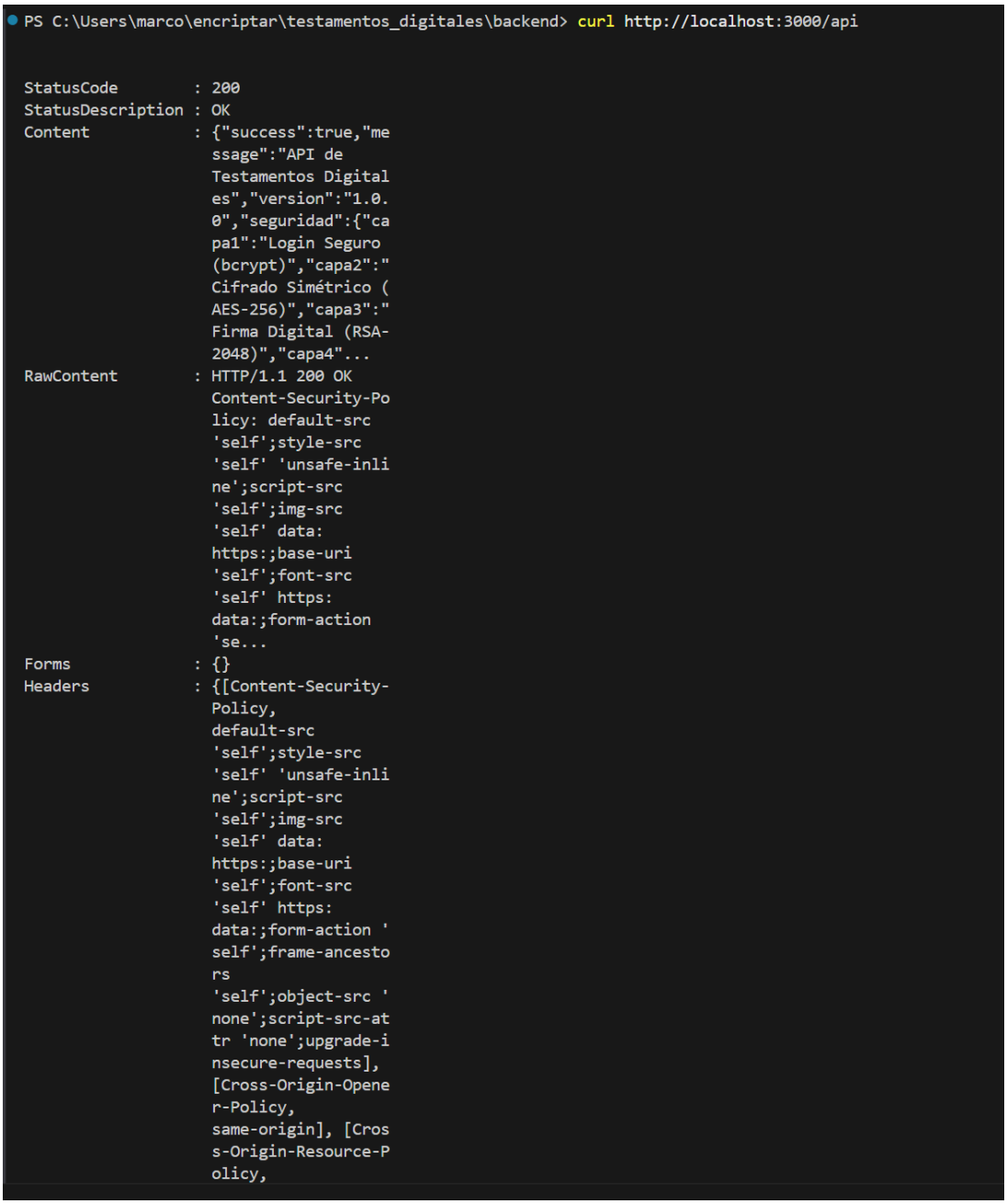

Figura 3: Endpoint principal de la API mostrando las 4 capas de seguridad activas

5.2. Captura 2: Llave Pública del Servidor (CAPA 4)

```
PS C:\Users\marco\encriptar\testamentos_digitales\backend> curl http://localhost:3000/api/crypto/server-public-key


StatusCode        : 200
StatusDescription : OK
Content           : {"success":true,"pu
                    blicKey":"-----BEGI
                    N PUBLIC KEY-----\n
                    MIIBIjANBgkqhkiG9w0
                    BAQEFAAOCAQ8AMIIBCg
                    KCAQEAx5GeJiyoxKqZu
                    6u5z3TN\nUDm4k/PKZ2
                    5+2DlT7Pxm/lb4c3jWL
                    7e5d+873xUNqyQqZc5A
                    3tgJ9DZbR2+xIQxB\nB
                    XV7a8PUP53...
RawContent        : HTTP/1.1 200 OK
                    Content-Security-Po
                    licy: default-src
                    'self';style-src
                    'self' 'unsafe-inli
                    ne';script-src
                    'self';img-src
                    'self' data:
                    https:;base-uri
                    'self';font-src
                    'self' https:
                    data:;form-action
                    'se...
Forms             : {}
Headers           : {[Content-Security-
                    Policy,
                    default-src
                    'self';style-src
                    'self' 'unsafe-inli
                    ne';script-src
                    'self';img-src
                    'self' data:
                    https:;base-uri
                    'self';font-src
                    'self' https:
                    data:;form-action '
                    self';frame-ancesto
                    rs
                    'self';object-src '
                    none';script-src-at
                    tr 'none';upgrade-i
                    nsecure-requests],
                    [Cross-Origin-Opene
                    r-Policy,
                    same-origin], [Cros
                    s-Origin-Resource-P
```

Figura 4: Obtención de la llave pública RSA del servidor para cifrado híbrido

5.3. Captura 3: Registro de Usuario (CAPA 1)

```
PS C:\Users\marco\encriptar\testamentos_digitales\backend> $body = @{username='marco';email='marco@test.com';password=
'Password123!'} | ConvertTo-Json; Invoke-RestMethod -Uri http://localhost:3000/api/auth/register -Method Post -Body $b
ody -ContentType 'application/json'


success    : True
message    : Usuario registrado
             exitosamente
token      : eyJhbGciOiJIUzI1NiIsInR5cC
             I6IkpXVCJ9.eyJ1c2VySWQiOjE
             sInVzZXJuYW1lIjoibWFyY28iL
             CJpYXQiOjE3NjM5MjQwNjUsImV
             4cCI6MTc2NDAxMDQ2NSwiYXVkI
             joidGVzdGFtZW50b3MtYXBpIiw
             iaXNzIjoidGVzdGFtZW50b3MtZ
             GlnaXRhbGVzIn0.a8t67FwJ3lp
             MxfZRkly0x42Gvd9_W9G1qiME-
             OgyohQ
user       : @{id=1; username=marco;
             email=marco@test.com}
privateKey : -----BEGIN PRIVATE
             KEY-----
             MIIEvQIBADANBgkqhkiG9w0BAQ
             EFAASCBKcwggSjAgEAAoIBAQCj
             X+d+OCouhZ6U
             IGs1Rvl0CmxJn/8qywoiNxB0Ek
             Jr4WnOt2Ku1p1izStM76+Q22am
             /lJmcTeVc+Q+
             D1RLHYAOqDwCR8OSMdfoitjAL7
             U/m6m98V/IR65xklRrCKTqRuIS
             SlKVoDVMgvwg
             hXRK9bKZqCynascRMxbiEbZaiK
             QCYaZ7RrI2/NHvKagoyDkC3JOz
             0QgzyJDGtPtB
             VDDj//LpA1187h2D2gK2Huzr4R
             FL3bSoXNmCcupQymlozfGdAb0T
             BGYPBZ1fNRRy
             g4wvTdc8PDgFMesSOvp5nn1cj+
             Ogy+hoht8tRsagtvTMERtSOcAi
             OF+C9ELnFIAI
             Z5ImWZwTAgMBAAECggEABrN4oZ
             ifJ+ckjbHVg+Sd3b2BtI4vkqtF
             hqK+9evVaQRg
             nFexE54xuJ6h771sp5XmnEniyl
             kX4EIiv5lN9ujGWsbAWcP+Ykdq
             5WeW8JPkX/zV
             op9WzOm3xRvGtgv8OHz6aXXX9L
             nGBeKt6Cct+ApWeyJgdoHhzJnn
             KEdB6G1U9AeW
             Q6IssiTa02mJ9kbH500tNdBGdM
             UsOyttF+DkkTBaQrCfIyNLlIHJ
             OOsgq95ciArD
```

Figura 5: Registro exitoso con bcrypt - Hash de contraseña y generación de llaves RSA

5.4. Captura 4: Pruebas Automatizadas Completas

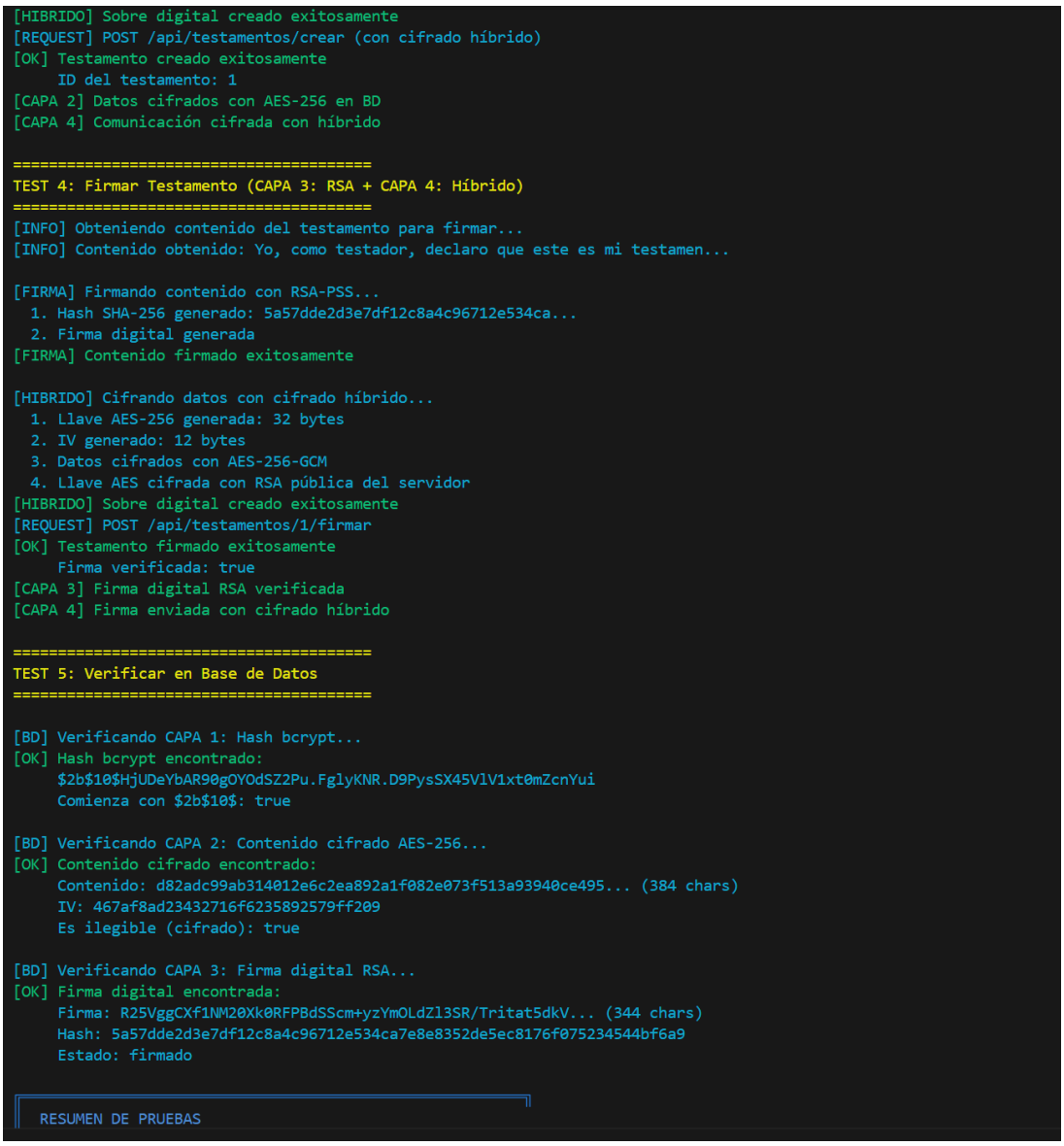

Figura 6: Ejecución completa del script de pruebas - Tests 1-5 y verificación en BD

5.5. Captura 5: Inicio de Pruebas Automatizadas

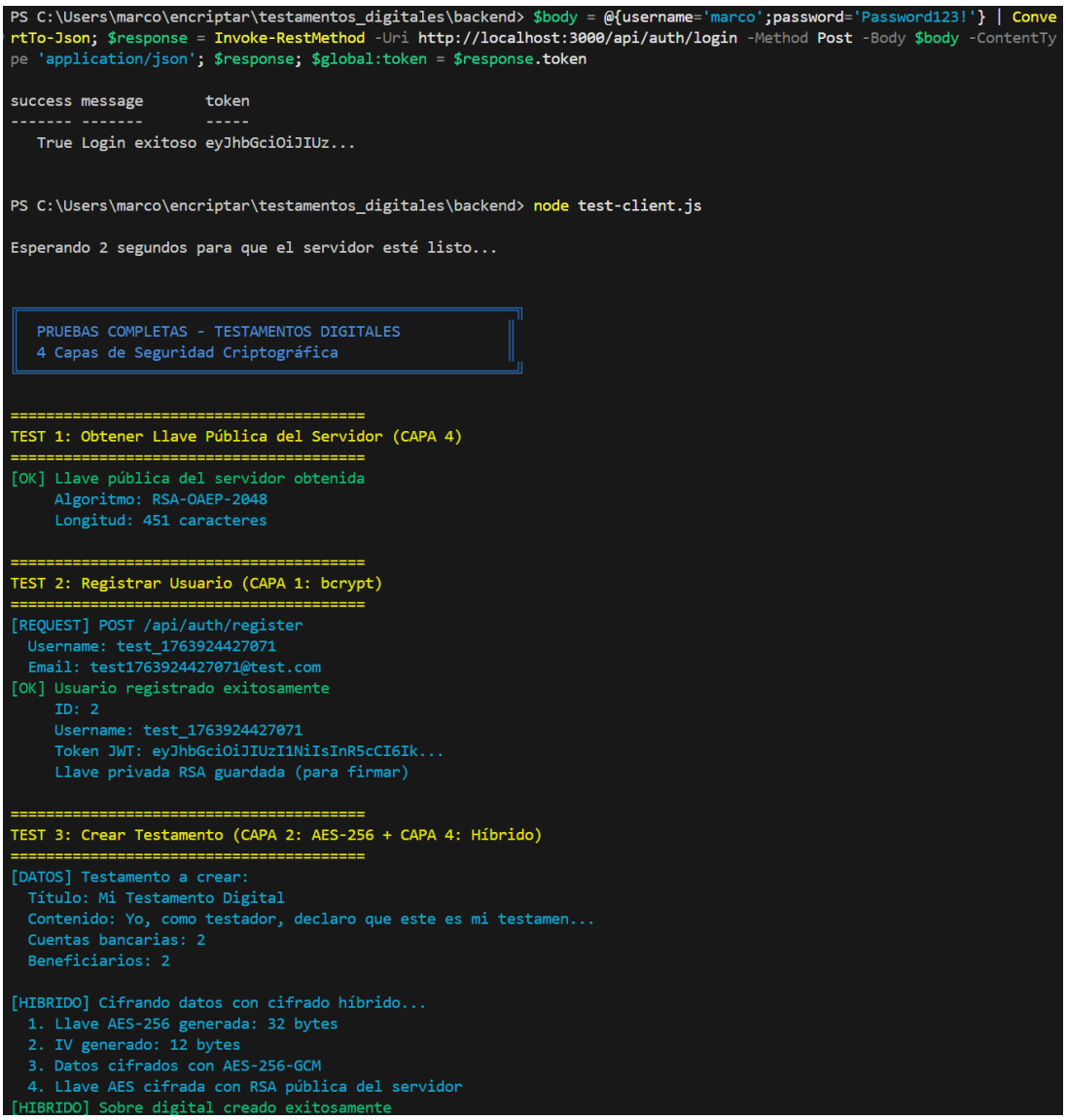

Figura 7: Primeros tests del sistema - Obtención de llave pública, registro y creación de testamento

6. Stack Tecnológico

6.1. Backend

- Node.js v18+
- Express.js 4.18
- SQLite3 5.1

6.2. Seguridad

- **bcrypt 5.1** - Hash de contraseñas (CAPA 1)
- **crypto (nativo)** - AES-256, RSA, SHA-256 (CAPAS 2, 3, 4)
- **jsonwebtoken 9.0** - Tokens JWT para sesiones
- **express-validator 7.0** - Validación de datos
- **helmet 7.1** - Headers de seguridad HTTP
- **winston 3.11** - Logging y auditoría

6.3. Algoritmos Criptográficos

Algoritmo	**Uso**
bcrypt (10 rounds)	Hash de contraseñas
AES-256-CBC	Cifrado simétrico en BD
RSA-PSS-2048	Firma digital de testamentos
RSA-OAEP-2048	Cifrado asimétrico (híbrido)
AES-256-GCM	Cifrado autenticado (híbrido)
SHA-256	Hash de contenido

Cuadro 2: Algoritmos criptográficos implementados

7. Estructura del Proyecto

7.1. Organización de Archivos

```
testamentos_digitales/
|-- backend/
|   |-- src/
|   |   |-- controllers/
|   |   |   |-- authController.js
|   |   |   |-- testamentoController.js
|   |   |   +-- verificacionController.js
|   |   |-- security/            [LAS 4 CAPAS AQUI]
|   |   |   |-- AuthService.js        (CAPA 1)
|   |   |   |-- EncryptionService.js  (CAPA 2)
|   |   |   |-- SignatureService.js   (CAPA 3)
|   |   |   +-- HybridCryptoService.js  (CAPA 4)
|   |   |-- models/
|   |   |-- routes/
|   |   |-- middleware/
|   |   +-- database/
|   |-- test-client.js
|   +-- package.json
|-- ARQUITECTURA.md
|-- README.md
|-- EVIDENCIA_4_CAPAS.md
+-- PARA_ENTREGAR.md
```

7.2. Archivos Clave

- `AuthService.js` - Implementación de bcrypt y JWT
- `EncryptionService.js` - Cifrado AES-256 para BD
- `SignatureService.js` - Firma digital RSA
- `HybridCryptoService.js` - Sobre digital (RSA + AES)
- `test-client.js` - Script de pruebas automatizadas

8. Instrucciones de Ejecución

8.1. Instalación

```
# 1. Instalar dependencias
cd backend
npm install

# 2. Generar llaves criptograficas
node src/utils/generateKeys.js

# 3. Configurar .env (copiar llaves generadas)
cp .env.example .env

# 4. Inicializar base de datos
npm run init-db

# 5. Iniciar servidor
npm start
```

8.2. Ejecutar Pruebas

```
# En otra terminal
node test-client.js
```

Resultado esperado:

```
+========================================+
|  RESULTADO: 5/5 PRUEBAS EXITOSAS       |
+========================================+
    TODAS LAS CAPAS FUNCIONANDO
```

9. Conclusiones

9.1. Aprendizaje Obtenido

A través del desarrollo de este proyecto, se obtuvieron aprendizajes significativos sobre las herramientas y técnicas de seguridad criptográfica:

9.1.1. bcrypt para Autenticación

- Comprensión profunda del algoritmo de hashing adaptativo
- Implementación correcta de salt rounds para resistencia a ataques de fuerza bruta
- Gestión segura de contraseñas sin almacenamiento en texto plano
- Integración con sistemas de autenticación basados en JWT

9.1.2. AES-256 para Cifrado Simétrico

- Dominio del modo CBC para cifrado de datos en reposo
- Importancia crítica del vector de inicialización (IV) único por operación
- Gestión segura de llaves maestras mediante variables de entorno
- Implementación de cifrado transparente en capa de base de datos

9.1.3. RSA para Firma Digital

- Generación y gestión de pares de llaves asimétricas RSA-2048
- Implementación de esquema PSS (Probabilistic Signature Scheme)
- Garantías criptográficas: autenticidad, integridad y no repudio
- Integración de SHA-256 para hashing de contenido previo a firma

9.1.4. Cifrado Híbrido (RSA + AES)

- Comprensión del concepto de "sobre digital"
- Combinación efectiva de cifrado simétrico y asimétrico
- Uso de AES-256-GCM para cifrado autenticado
- Implementación de RSA-OAEP para protección de llaves de sesión
- Aplicación práctica del principio Defense in Depth

9.2. Herramientas y Tecnologías

- **Node.js Crypto Module:** API nativa para operaciones criptográficas de alto rendimiento
- **Express.js:** Framework robusto para construcción de APIs RESTful seguras
- **SQLite:** Base de datos ligera ideal para aplicaciones con cifrado de datos sensibles
- **Winston:** Sistema de logging para auditoría de operaciones críticas
- **Helmet.js:** Middleware para protección de headers HTTP

9.3. Buenas Prácticas Implementadas

1. **Separación de responsabilidades:** Cada capa de seguridad en un módulo independiente
2. **Gestión de secretos:** Uso de variables de entorno, nunca hardcoded
3. **Validación exhaustiva:** express-validator para sanitización de entrada
4. **Testing automatizado:** Script de pruebas completo con verificación en BD
5. **Documentación completa:** Código comentado y documentación técnica detallada

9.4. Reflexión Final

Este proyecto demostró que la seguridad efectiva no es una característica única, sino un conjunto de capas complementarias que trabajan en conjunto. La implementación de las 4 capas de seguridad criptográfica proporcionó una comprensión práctica y profunda de cómo proteger datos sensibles en aplicaciones del mundo real, aplicando los principios fundamentales de confidencialidad, integridad, autenticidad y no repudio.

10. Referencias

10.1. Documentación de Algoritmos

- bcrypt - https://www.npmjs.com/package/bcrypt
- Node.js Crypto Module - https://nodejs.org/api/crypto.html
- AES-256 Encryption - NIST FIPS 197
- RSA-PSS Signature - RFC 8017
- Hybrid Encryption - Combining RSA and AES

10.2. Mejores Prácticas de Seguridad

- OWASP Top 10 Security Risks
- Defense in Depth Strategy
- Cryptographic Key Management
- Secure Password Storage

——- FIN DEL DOCUMENTO ——-

Agradecimientos al Profesor Brandon Efren Venegas Olvera
por sus enseñanzas y apoyo en esta materia.