**TRABAJO FIN DE GRADO**


UCAM
UNIVERSIDAD CATÓLICA
DE MURCIA

# ESCUELA POLITÉCNICA SUPERIOR

*Grado en Ingeniería Informática*

---

## MANUAL DE CÓDIGO RELEVANTE

*Autor:*

*D. Marc Hernández Montesinos*

*Directora:*

Dra. Dña. *Angélica Guzmán Ponce*

*Murcia, Junio de 2026*

**TRABAJO FIN DE GRADO**


UCAM
UNIVERSIDAD CATÓLICA
DE MURCIA

# ESCUELA POLITÉCNICA SUPERIOR

*Grado en Ingeniería Informática*

## MANUAL DE CÓDIGO RELEVANTE

*Autor:*

*D. Marc Hernández Montesinos*

*Directora:*

Dra. Dña. *Angélica Guzmán Ponce*

*Murcia, Junio de 2026*

*https://tfg.marchernandez.es/videos/video-demostrativo.mp4*

# ÍNDICE

## MANUAL DE CÓDIGO RELEVANTE

**Catálogo completo de fragmentos de código del sistema PKI + SSO**

Documento separado del Trabajo de Fin de Grado *"Diseño e implementación de un sistema integrado de PKI y SSO para organizaciones pequeñas"* de Marc Hernández Montesinos (UCAM, Grado en Ingeniería Informática).

| Campo | Valor |
|---|---|
| Documento | Manual de Código Relevante |
| Versión | 1.0 |
| Fecha | 1 junio 2026 |
| URL pública | https://tfg.marchernandez.es/manuales/Manual_Codigo_Relevante.pdf |
| TFG asociado | https://tfg.marchernandez.es |

Este manual conserva la numeración interna original (sección A.x) por trazabilidad con las versiones previas del documento. Las referencias cruzadas que apuntan al cuerpo del TFG (capítulos 1-10, anexos A-D) se mantienen tal cual y son válidas frente al PDF principal del TFG.

**Documento complementario al TFG.** Reproduce íntegramente el catálogo de fragmentos de código seleccionados del sistema PKI + SSO. En el TFG impreso se conserva únicamente un subconjunto reducido (Anexo A) con los fragmentos más representativos en términos de seguridad y arquitectura; el resto se documenta aquí para mantener manejable el cuerpo principal sin perder trazabilidad técnica.

Este manual **no** reproduce el código completo del sistema. El repositorio total supera las 20.000 líneas de PHP, SQL y configuración, y reproducirlas íntegramente haría inmanejable la lectura sin aportar valor adicional al lector.

En su lugar se han seleccionado **fragmentos representativos** que cumplen al menos uno de estos criterios:

| Criterio | Por qué interesa |
|---|---|
| **Seguridad** | Implementa una defensa concreta del modelo STRIDE (Sección 7.9) o de las 9 capas (Sección 7.8) |
| **Arquitectura propia** | Refleja una decisión de diseño documentada en el cuerpo (Sección 7.x) |
| **Interoperabilidad estándar** | Implementa un RFC explícito (RFC 6749, 6238, 7636, 5280, 6960, 7517, ...) |
| **Lógica criptográfica nuclear** | PKI, firma JWT, cifrado de passphrases, generación de claves |
| **Gestión de revocación** | Flujos CRL/OCSP, vinculación cruzada certificado↔BD |
| **Hardening transversal** | CSRF, rate limiting, validación de redirect_uri, security headers |

Se han omitido:

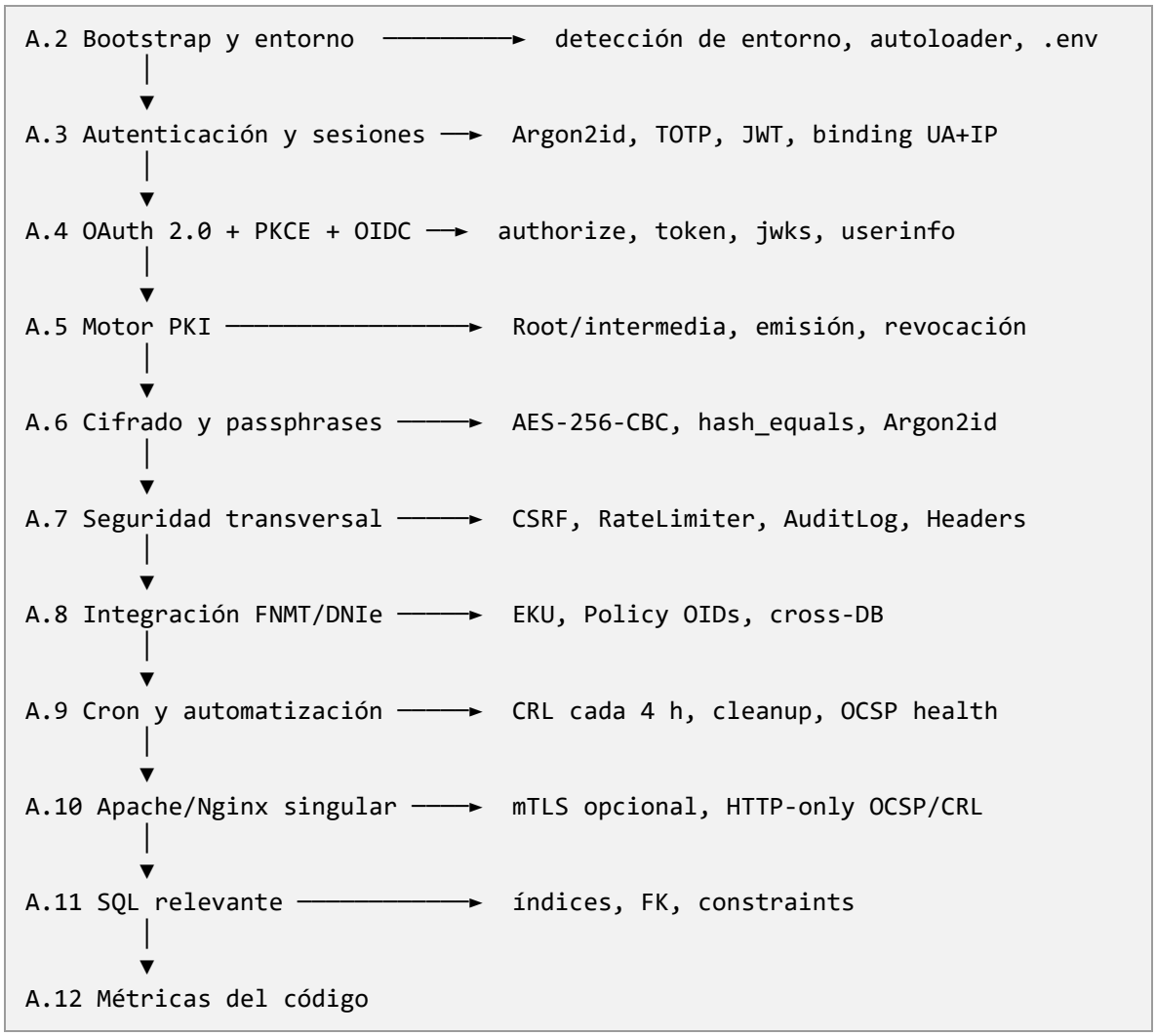
- Vistas HTML repetitivas (formularios, tablas, modales): boilerplate que no aporta diferencial.
- Configuración de Plesk / Apache rutinaria (sí se incluyen los snippets singulares en Sección A.10).
- Migraciones SQL ya documentadas íntegramente en el documento Modelo de Datos (Sección E.5).
- Tests unitarios y de integración (resumidos en Sección 8.1 del cuerpo).

- Código generado por instaladores (autoloaders, fixtures de prueba).

Cuando un fragmento ha sido editado para entrar en una página, se marca con `// ... más código ...` y se referencia la ruta exacta del archivo en el repositorio.

### A.1 Mapa de selección

La selección sigue el orden lógico que el lector necesita para entender el sistema desde abajo hacia arriba:




### A.2 Bootstrap y carga segura de entorno

#### *A.2.1 Detección de entorno del PKI*

El sistema funciona tanto en desarrollo local (Windows) como en producción (AlmaLinux). El bootstrap detecta el entorno y resuelve rutas dinámicamente, sin depender de variables hardcodeadas.

```
// --- Detección de entorno (local vs producción) ---
$isProduction = (PHP_OS_FAMILY !== 'Windows' &&
is_dir('/var/www/vhosts/marchernandez.es'));
define('PKI_ENV', $isProduction ? 'production' : 'local');

if ($isProduction) {
    define('PKI_ROOT',
'/var/www/vhosts/marchernandez.es/pki.marchernandez.es');
    define('PKI_PUBLIC', PKI_ROOT . '/public');
    define('PKI_CA_STORAGE', PKI_ROOT . '/ca');
    define('PKI_LOGS', PKI_ROOT . '/logs');
    define('PKI_LIB', PKI_ROOT . '/lib');

    define('CRL_WEB_DIR',
'/var/www/vhosts/marchernandez.es/crl.marchernandez.es');
    define('CA_WEB_DIR',
'/var/www/vhosts/marchernandez.es/ca.marchernandez.es');
    define('OCSP_WEB_DIR',
'/var/www/vhosts/marchernandez.es/ocsp.marchernandez.es');
    define('SSO_PRIVATE', '/var/www/vhosts/marchernandez.es/private/sso');
} else {
    define('PKI_ROOT', __DIR__);
    define('PKI_PUBLIC', PKI_ROOT . '/public');
    define('PKI_CA_STORAGE', PKI_ROOT . '/ca');
    define('PKI_LIB', PKI_ROOT . '/lib');

    $projectRoot = dirname(PKI_ROOT);
    define('CRL_WEB_DIR', $projectRoot . '/crl');
    define('CA_WEB_DIR', $projectRoot . '/ca');
    define('OCSP_WEB_DIR', $projectRoot . '/ocsp');
}
```

**Decisión arquitectónica.** Las rutas no se leen de configuración externa: se calculan en `bootstrap.php` y se exponen como constantes globales. Esto elimina un fallo común en despliegues PHP - paths hardcodeados - y permite que el mismo *codebase* corra en Windows local y AlmaLinux producción sin cambios.

### *A.2.2 Carga de variables de entorno (`.env` minimalista)*

El PKI no usa ninguna librería externa para leer el `.env`. La función completa cabe en 12 líneas:

```
<?php
$envFile = __DIR__ . '/.env';
if (file_exists($envFile)) {
    $lines = file($envFile, FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
    foreach ($lines as $line) {
        if (str_starts_with(trim($line), '#')) continue;
        $parts = explode('=', $line, 2);
        if (count($parts) === 2) {
            $key = trim($parts[0]);
            $val = trim($parts[1]);
            putenv("{$key}={$val}");
            $_ENV[$key] = $val;
        }
    }
}
```

**Por qué tan sencillo.** La librería `vlucas/phpdotenv` añade ~20 dependencias transitivas para cubrir un caso de uso que aquí no se necesita (interpolación, validación de tipos, exports a `$_SERVER`). El TFG defiende reducir superficie de dependencias y la curva de auditoría: 12 líneas que hacen lo justo son más auditables que un paquete completo.

### *A.2.3 Autoloader PSR-4 manual*

```
spl_autoload_register(function (string $class): void {
    $prefix = 'MaHerMo\\PKI\\';
    $baseDir = PKI_LIB . '/';

    $len = strlen($prefix);
    if (strncmp($prefix, $class, $len) !== 0) {
        return;
    }

    $relativeClass = substr($class, $len);
    $file = $baseDir . str_replace('\\', '/', $relativeClass) . '.php';

    if (file_exists($file)) {
        require $file;
    }
});
```

**Decisión.** El estándar PSR-4 cabe en 15 líneas. Tomarlas de Composer arrastraría ~600 archivos de `vendor/` solo para autoload. Esto enlaza con Sección 6 del cuerpo: control sobre dependencias en organizaciones pequeñas.

## A.3 Autenticación y sesiones SSO

### *A.3.1 Hash de contraseñas con Argon2id*

```
public static function hashPassword(string $password): string
{
    return password_hash($password, PASSWORD_ARGON2ID, [
        'memory_cost' => 65536,  // 64 MiB
        'time_cost'   => 4,
        'threads'     => 1,
    ]);
}

public static function verifyPassword(string $password, string $hash): bool
{
    return password_verify($password, $hash);
}
```

**Decisión.** Argon2id con 64 MiB / 4 iteraciones es el preset recomendado por OWASP Password Storage Cheat Sheet (2025). Es mucho más costoso para un atacante con GPU que bcrypt y `password_verify` ya es resistente a *timing attacks* internamente (Sección 7.6.1, Sección 7.9 STRIDE: Spoofing).

### *A.3.2 TOTP RFC 6238: cifrado del secreto + ventana de tolerancia*

El secreto TOTP **nunca** se almacena en claro. Se cifra con AES-256-GCM antes de guardarlo en `sso_users.totp_secret`:

```
public static function encryptSecret(string $plainSecret): string
{
    $key = self::getEncryptionKey();
    $iv = random_bytes(self::IV_LENGTH);
    $tag = '';

    $ciphertext = openssl_encrypt($plainSecret, self::CIPHER, $key,
OPENSSL_RAW_DATA, $iv, $tag, '', self::TAG_LENGTH);
    if ($ciphertext === false) {
        throw new \RuntimeException('Error cifrando secreto TOTP: ' .
openssl_error_string());
    }

    return self::ENC_PREFIX . base64_encode($iv . $tag . $ciphertext);
}

public static function decryptSecret(string $stored): string
{
    if (substr($stored, 0, strlen(self::ENC_PREFIX)) !== self::ENC_PREFIX) {
        return $stored;   // compatibilidad con secretos antiguos sin cifrar
    }

    $key = self::getEncryptionKey();
    $raw = substr($stored, strlen(self::ENC_PREFIX));
    $data = base64_decode($raw, true);

    $iv = substr($data, 0, self::IV_LENGTH);
    $tag = substr($data, self::IV_LENGTH, self::TAG_LENGTH);
    $ciphertext = substr($data, self::IV_LENGTH + self::TAG_LENGTH);

    $plain = openssl_decrypt($ciphertext, self::CIPHER, $key,
OPENSSL_RAW_DATA, $iv, $tag);
    if ($plain === false) {
        throw new \RuntimeException('Error descifrando secreto TOTP.');
    }
    return $plain;
}
```

**Decisión.** AES-256-GCM aporta cifrado **autenticado** (AEAD): si un atacante manipula el ciphertext, el `tag` no verifica y el descifrado falla - propiedad imposible con AES-CBC sin un HMAC añadido manualmente. El TFG usa CBC para las passphrases de CA (Sección A.6.1) porque el cifrado se hizo antes que esta decisión, pero los flujos nuevos usan GCM.

La verificación TOTP usa una ventana ±30 s y comparación en tiempo constante:

```
public static function verifyCode(string $storedSecret, string $code, int
$window = -1): bool
{
    if ($window < 0) {
        $window = self::totpWindow();
    }
    $secret    = self::decryptSecret($storedSecret);
    $period    = self::totpPeriod();
    $timestamp = time();
    for ($i = -$window; $i <= $window; $i++) {
        $checkTime = $timestamp + ($i * $period);
        if (hash_equals(self::generateCode($secret, $checkTime), $code)) {
            return true;
        }
    }
    return false;
}
```

**Decisión.** `hash_equals` evita ataques de tiempo (RFC 6238 no obliga, pero OWASP lo recomienda). La ventana de ±1 períodos (30 s) compensa desincronización de reloj entre el servidor y el dispositivo del usuario sin abrir una ventana de replay significativa.

### *A.3.3 Firma JWT RS256 desde clave en disco (rotación por `kid`)*

La clave privada **no** se almacena en BD; solo el `key_file_path` apunta al archivo `.pem` en `sso/keys/`. Esto permite rotación de claves desactivando filas en BD sin tocar el sistema de archivos, y permite proteger el archivo con permisos `0600`.

```
public static function encode(array $payload): string
{
    $keyPair = self::getActiveKeyPair();

    $header = [
        'typ' => 'JWT',
        'alg' => JWT_ALGORITHM,
        'kid' => $keyPair['kid'],
    ];

    $headerEncoded = self::base64UrlEncode(json_encode($header));
    $payloadEncoded = self::base64UrlEncode(json_encode($payload));
    $data = $headerEncoded . '.' . $payloadEncoded;

    $privateKeyResource = openssl_pkey_get_private($keyPair['private_key']);
    if ($privateKeyResource === false) {
        throw new \RuntimeException('Error cargando clave privada para firma
JWT.');
    }

    $signature = '';
    $signed = openssl_sign($data, $signature, $privateKeyResource,
OPENSSL_ALGO_SHA256);
    if (!$signed) {
        throw new \RuntimeException('Error firmando JWT.');
    }

    return $data . '.' . self::base64UrlEncode($signature);
}
```

**Decisión.** `kid` en el header del JWT permite que el lado verificador identifique exactamente qué clave usar, lo que permite tener múltiples claves activas durante un periodo de gracia (Manual del Administrador (Sección C.8)). Cualquier cliente que verifique con el JWKS endpoint puede gestionar la rotación sin intervención humana.

### *A.3.4 Validación de sesión: binding UA + IP como señal de riesgo*

```
public static function validateSession(string $accessToken): ?array
{
    $payload = JWT::decode($accessToken);
    if (!$payload) {
        return null;
    }

    $db = Database::getInstance();
    $stmt = $db->prepare(
        'SELECT s.*, u.username, u.email, u.display_name, u.role, u.is_active
         FROM sso_sessions s JOIN sso_users u ON s.user_id = u.id
         WHERE s.token_hash = :hash AND s.is_active = 1 AND s.expires_at >
NOW()'
    );
    $stmt->execute(['hash' => hash('sha256', $accessToken)]);
    $session = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$session || !$session['is_active']) {
        return null;
    }

    $currentIp = self::getClientIP();
    $currentFp = self::generateFingerprint($_SERVER['HTTP_USER_AGENT'] ??
'');
    $violation = '';

    if (!empty($session['ip_address']) && $session['ip_address'] !==
$currentIp) {
        $violation = “IP changed: {$session['ip_address']} -> {$currentIp}”;
    }
    if (!empty($session['ua_fingerprint']) &&
!hash_equals($session['ua_fingerprint'], $currentFp)) {
        $violation .= ($violation ? ' | ' : '') . 'UA fingerprint mismatch';
    }

    if ($violation !== '') {
        self::destroySession($session['id']);
        AuditLog::log('session_binding_violation', $session['user_id'],
$currentIp, [
            'session_id' => $session['id'], 'reason' => $violation,
            'stored_ip' => $session['ip_address'], 'current_ip' =>
$currentIp,
        ]);
        return null;
    }
    // ... actualizar last_activity y devolver datos de sesión ...
}
```

**Decisión.** La sesión se valida por tres niveles (firma JWT → hash en BD → binding contextual). El binding UA+IP no se propone como garantía criptográfica en el TFG (Sección 7.5.3): se documenta explícitamente como “señal contextual de riesgo”. Una IP cambiada por CGNAT o un UA actualizado pueden dar falsos

positivos; cuando ocurren, la sesión se destruye y el evento se audita como `critical`. El usuario simplemente tiene que volver a autenticarse.

### *A.3.5 Fingerprint estable del User-Agent*

El UA completo cambia constantemente (versiones puntuales del navegador). Se reduce a un fingerprint estable que solo incluye SO + familia de navegador + versión mayor:

```
public static function generateFingerprint(string $userAgent): string
{
    $ua = strtolower(trim($userAgent));
    $c = [];
    if (preg_match('/windows nt [\d.]+/', $ua, $m)) { $c[] = $m[0]; }
    elseif (preg_match('/mac os x [\d_.]+/', $ua, $m)) { $c[] = $m[0]; }
    elseif (str_contains($ua, 'linux')) { $c[] = 'linux'; }
    elseif (preg_match('/android [\d.]+/', $ua, $m)) { $c[] = $m[0]; }
    elseif (preg_match('/iphone os [\d_]+/', $ua, $m)) { $c[] = $m[0]; }

    if (preg_match('/chrome\/([\d]+)/', $ua, $m)) { $c[] = 'chrome/' . $m[1];
}
    elseif (preg_match('/firefox\/([\d]+)/', $ua, $m)) { $c[] = 'firefox/' .
$m[1]; }
    elseif (preg_match('/safari\/([\d]+)/', $ua, $m)) { $c[] = 'safari/' .
$m[1]; }
    elseif (preg_match('/edg\/([\d]+)/', $ua, $m)) { $c[] = 'edge/' . $m[1];
}

    return hash('sha256', empty($c) ? $ua : implode('|', $c));
}
```

**Decisión.** Usar el UA completo provocaría sesiones rotas cada vez que el navegador se actualiza (típicamente 1-2 veces por semana). Reducir a SO + familia + major version es un compromiso entre seguridad y experiencia. Documentado en Sección 7.5.3.

## A.4 Implementación OAuth 2.0 + PKCE + OIDC

Esta sección es el núcleo “académicamente fuerte” del SSO. Implementa fielmente RFC 6749 (OAuth 2.0), RFC 7636 (PKCE), RFC 7519 (JWT), RFC 7517 (JWKS) y la especificación OpenID Connect Discovery 1.0 (compatible con RFC 8414, *OAuth 2.0 Authorization Server Metadata*).

### *A.4.1 Validación estricta del `redirect_uri` (anti Open Redirect)*

```
private static function isRedirectUriAllowed(string $redirectUri, array
$app): bool
{
    $parsed = parse_url($redirectUri);
    if ($parsed === false || empty($parsed['host'] ?? '')) {
        return false;
    }

    $scheme = strtolower($parsed['scheme'] ?? '');
    $host = strtolower($parsed['host'] ?? '');

    // Solo HTTPS permitido (excepción: localhost para desarrollo)
    if ($scheme !== 'https') {
        if (!($scheme === 'http' && in_array($host, ['localhost',
'127.0.0.1'], true))) {
            return false;
        }
    }

    // Prohibir fragmentos (previene ataques de redirección)
    if (isset($parsed['fragment'])) {
        return false;
    }

    // Prohibir credenciales en la URL
    if (!empty($parsed['user'] ?? '') || !empty($parsed['pass'] ?? '')) {
        return false;
    }

    // Prohibir IPs directas (excepto localhost)
    if (filter_var($host, FILTER_VALIDATE_IP) && $host !== '127.0.0.1') {
        return false;
    }

    return InputValidator::isRedirectSafe($redirectUri,
$app['allowed_domains']);
}
```

**Decisión.** Cinco filtros encadenados antes de comparar con la whitelist de dominios. Esto cubre el OWASP Top 10 A05 (Security Misconfiguration) y mitiga la amenaza T-OAuth-04 del threat model STRIDE (Sección 7.9). Una redirección a `https://attacker.com#valid=https://app.com` o a `https://user:pass@evil.com` no pasaría: el fragmento y las credenciales se rechazan antes.

### *A.4.2 PKCE: validación del `code_challenge` (RFC 7636)*

```
private static function validatePkceParams(string $codeChallenge, string
$codeChallengeMethod): array
{
    if ($codeChallenge === '') {
        return [
            'valid' => true,
            'code_challenge' => null,
            'code_challenge_method' => null,
        ];   // PKCE opcional para compatibilidad con clientes legados
    }

    $method = $codeChallengeMethod !== '' ? strtoupper($codeChallengeMethod)
: 'S256';
    if ($method !== 'S256') {
        return ['valid' => false, 'error' => 'code_challenge_method debe ser
“S256” (plain no permitido).'];
    }

    $length = strlen($codeChallenge);
    if ($length < 43 || $length > 128) {
        return ['valid' => false, 'error' => 'code_challenge debe tener entre
43 y 128 caracteres.'];
    }

    if (!preg_match('/^[A-Za-z0-9\-._~]+$/', $codeChallenge)) {
        return ['valid' => false, 'error' => 'code_challenge contiene
caracteres no permitidos (solo BASE64URL).'];
    }

    return [
        'valid' => true,
        'code_challenge' => $codeChallenge,
        'code_challenge_method' => $method,
    ];
}
```

**Decisión.** El método `plain` (challenge == verifier) está rechazado explícitamente, aunque el RFC lo permita para compatibilidad. Cualquier cliente moderno (incluido el portal PKI propio del TFG) debe usar S256. Esto previene que un atacante MITM que capture el `code_challenge` pueda intercambiar el code por sí mismo.

### *A.4.3 PKCE: verificación del `code_verifier` en `/token`*

```
private static function verifyCodeVerifier(string $codeVerifier, string
$codeChallenge): bool
{
    $verifierLength = strlen($codeVerifier);
    if ($verifierLength < 43 || $verifierLength > 128) {
        return false;
    }

    if (!preg_match('/^[A-Za-z0-9\-._~]+$/', $codeVerifier)) {
        return false;
    }

    $computed = rtrim(strtr(base64_encode(hash('sha256', $codeVerifier,
true)), '+/', '-_'), '=');
    return hash_equals($codeChallenge, $computed);
}
```

**Decisión.** Comparación con `hash_equals` (constant-time) tras calcular `BASE64URL(SHA256(verifier))`. La condición de longitud y charset se aplica también al verifier, no solo al challenge, para evitar que un atacante use un verifier malformado que pase la regex débil, pero produzca una colisión SHA-256.

### *A.4.4 Intercambio code↔token con todas las verificaciones encadenadas*

```
$stmt = $db->prepare(
    'SELECT ac.*, a.client_secret_hash, a.client_id as app_client_id
     FROM sso_authorization_codes ac
     JOIN sso_applications a ON ac.application_id = a.id
     WHERE ac.code = :code LIMIT 1'
);
$stmt->execute(['code' => $codeHash]);
$authCode = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$authCode) {
    return ['error' => 'invalid_grant', 'http_status' => 400, /* ... */];
}

// Código ya utilizado (posible replay attack)
if ((int)$authCode['used'] === 1) {
    return ['error' => 'invalid_grant', 'http_status' => 400, /* ... */];
}

// Código expirado
if (strtotime($authCode['expires_at']) < time()) {
    $db->prepare('UPDATE sso_authorization_codes SET used = 1 WHERE id =
:id')
       ->execute(['id' => $authCode['id']]);
    return ['error' => 'invalid_grant', 'http_status' => 400, /* ... */];
}

// Validar client_id y client_secret (argon2id)
if (!hash_equals($authCode['app_client_id'], $clientId)) {
    return ['error' => 'invalid_client', 'http_status' => 401, /* ... */];
}
if (!password_verify($clientSecret, $authCode['client_secret_hash'])) {
    return ['error' => 'invalid_client', 'http_status' => 401, /* ... */];
}

// Validar redirect_uri (binding exacto al code emitido)
if ($authCode['redirect_uri'] !== $redirectUri) {
    return ['error' => 'invalid_grant', 'http_status' => 400, /* ... */];
}

// Validación PKCE: si el código se emitió con code_challenge, se exige
code_verifier
$storedChallenge = $authCode['code_challenge'] ?? null;
if ($storedChallenge !== null && $storedChallenge !== '') {
    if ($codeVerifier === null || $codeVerifier === '') {
        return ['error' => 'invalid_grant', 'http_status' => 400, /* ... */];
    }
    if (!self::verifyCodeVerifier($codeVerifier, $storedChallenge)) {
        return ['error' => 'invalid_grant', 'http_status' => 400, /* ... */];
    }
}

// Marcar código como usado (single-use)
$db->prepare('UPDATE sso_authorization_codes SET used = 1 WHERE id = :id')
   ->execute(['id' => $authCode['id']]);
```

**Decisión.** Las siete verificaciones (existencia, no-usado, no-expirado, `client_id`, `client_secret`, `redirect_uri` exacta, PKCE) se ejecutan antes de marcar el code como usado. El orden importa: por ejemplo, expirado se marca como `used=1` para evitar reintentos, pero `client_id` inválido **no** lo marca (sería otro cliente legítimo el que podría haber recibido ese code; aunque esto rompe el binding cliente↔code que precisamente queremos hacer cumplir).

Implementa fielmente RFC 6749 Sección 4.1.3 y RFC 7636 Sección 4.6.

### *A.4.5 JWKS endpoint (RFC 7517) en 12 líneas*

```
<?php
declare(strict_types=1);
require_once dirname(__DIR__, 2) . '/src/bootstrap.php';

use MaHerMoSSO\Security\SecurityHeaders;
use MaHerMoSSO\Session\JWT;

SecurityHeaders::jsonHeaders();
SecurityHeaders::corsHeaders();
header('Cache-Control: public, max-age=3600');

if ($_SERVER['REQUEST_METHOD'] === 'OPTIONS') {
    http_response_code(204);
    exit;
}

echo json_encode(JWT::getJWKS());
```

```
public static function getJWKS(): array
{
    $stmt = Database::getInstance()->prepare(
        'SELECT kid, public_key_pem, algorithm FROM sso_keypairs WHERE
is_active = 1'
    );
    $stmt->execute();

    $jwks = ['keys' => []];
    foreach ($stmt->fetchAll(PDO::FETCH_ASSOC) as $key) {
        $publicKeyResource = openssl_pkey_get_public($key['public_key_pem']);
        if ($publicKeyResource === false) continue;

        $details = openssl_pkey_get_details($publicKeyResource);
        if (!$details || !isset($details['rsa'])) continue;

        $jwks['keys'][] = [
            'kty' => 'RSA',
            'use' => 'sig',
            'alg' => $key['algorithm'],
            'kid' => $key['kid'],
            'n' => self::base64UrlEncode($details['rsa']['n']),
            'e' => self::base64UrlEncode($details['rsa']['e']),
        ];
    }
    return $jwks;
}
```

**Decisión.** El endpoint expone todas las claves activas, no solo la última. Durante una rotación (Manual del Administrador (Sección C.8)), el cliente que verifica un JWT antiguo todavía encuentra su clave aquí. Esto convierte la rotación en una operación sin downtime, lo cual es importante para mantener la disponibilidad (RNF-08).

### A.5 Motor PKI

Es el “corazón del proyecto”, según el cuerpo (Sección 7.4).

### *A.5.1 Generación de la Root CA (subproceso OpenSSL)*

```
public static function generateRootCA(array $params): array
{
    $isFirstRoot = !Database::fetchOne('SELECT id FROM
certificate_authorities WHERE type = ? LIMIT 1', ['root']);
    $keySize = $params['key_size'] ?? 4096;
    $days = $params['validity_days'] ?? (20 * 365);
    $digest = $params['digest'] ?? 'sha384';
    $passphrase = $params['passphrase'] ?? bin2hex(random_bytes(32));
    $subject = self::buildSubjectString($params);

    // 1. Reservar ID en BD insertando placeholder inactivo
    //    (la DPC del cert necesita conocer su propio ID antes de generarse)
    $caId = Database::insert('certificate_authorities', [
        'name' => $params['name'], 'common_name' => $params['common_name'],
        'type' => 'root', 'parent_ca_id' => null, 'serial_counter' => 1,
        'certificate_pem' => '', 'key_file_path' => '',
        'is_active' => 0, /* placeholders ... */
    ]);

    $dirName = $isFirstRoot ? 'root' : 'root_' . $caId;
    $caDir = pkiConfig('paths.ca_storage') . '/' . $dirName;
    self::ensureDirectory($caDir);

    // 2. Construir el comando OpenSSL con todos los argumentos escapados
    $cmd = sprintf(
        '%s req -x509 -new -utf8 -newkey rsa:%d -keyout %s -out %s -days %d '
        . '-%s -subj %s -passout pass:%s -config %s 2>&1',
        escapeshellcmd(self::getOpenSSLBin()),
        $keySize,
        escapeshellarg($keyFile), escapeshellarg($certFile),
        $days, escapeshellarg($digest),
        escapeshellarg($subject), escapeshellarg($passphrase),
        escapeshellarg($configFile)
    );

    $output = [];
    exec($cmd, $output, $returnCode);

    if ($returnCode !== 0) {
        self::deletePlaceholder($caId);
        return ['success' => false, 'error' => 'Error generando CA raíz: ' .
implode("\n", $output)];
    }

    // 3. La passphrase no se guarda en BD: solo cifrada en passphrase.key
(chmod 600)
    file_put_contents($caDir . '/passphrase.key',
self::encryptPassphrase($passphrase));
    chmod($caDir . '/passphrase.key', 0600);

    // 4. Actualizar el placeholder con los datos reales del certificado
emitido
    $certPem = file_get_contents($certFile);
    $fingerprint = openssl_x509_fingerprint($certPem, 'sha256');
    Database::update('certificate_authorities', [
        'certificate_pem' => $certPem,
        'key_file_path' => $dirName . '/root_ca.key',
```

```
        'is_active' => 1,
        'fingerprint_sha256' => $fingerprint,
        /* ... */
    ], 'id = ?', [$caId]);

    return ['success' => true, 'ca_id' => $caId, 'fingerprint' =>
$fingerprint, /* ... */];
}
```

**Decisión.** Se delega a `openssl` CLI (no a `ext-openssl`) porque la extensión PHP de OpenSSL no expone todos los atributos X.509 necesarios - concretamente `pathlen`, `crlDistributionPoints` con SAN URI, `certificatePolicies` con OIDs y CPS - sin recurrir a parches sucios sobre el resultado. La línea de comandos garantiza interoperabilidad bit-a-bit con OpenSSL stand-alone (clientes externos pueden reproducir la operación). Discutido en Sección 7.4.3 del cuerpo.

El patrón “placeholder + actualizar” resuelve un problema típico de PKI: la DPC del propio certificado debe contener la URL que incluye el ID de la CA, lo que crea una dependencia circular (necesito el ID antes de firmar; pero el ID se asigna al insertar). La solución: insertar placeholder con `is_active=0`, leer el ID, generar el cert con el ID ya conocido, actualizar la fila con el cert real.

### *A.5.2 `pathlen:0` estructural para intermedias*

```
// pathlen:0 estructural en toda intermedia: la jerarquía operativa es
// raíz -> intermedia -> certificado final (no se emiten sub-intermedias).
// El parámetro $chainDepth se conserva por compatibilidad con el cálculo
// de profundidad, pero el resultado se fija a 0 para cerrar la cadena.
$pathlenStr = 'pathlen:0';
```

**Decisión.** Aunque el código todavía calcula `$chainDepth`, el `pathlen` se fija a 0 independientemente de la profundidad. Esto traduce a la salida del certificado X.509 una propiedad estructural: aunque alguien comprometa una intermedia y consiga firmar otra “intermedia” debajo, esa nueva intermedia tendría `CA:TRUE, pathlen:0`, pero sería rechazada por cualquier validador conforme a RFC 5280 Sección 6.1.4(k) porque su padre ya tenía `pathlen:0`. Documentado en Sección 7.4.1 y documento Plantillas OpenSSL.

### *A.5.3 Revocación + regeneración automática de CRL*

```
public static function revokeCertificate(int $certId, string $reason, int
$userId): array
{
    $cert = Database::fetchOne('SELECT * FROM certificates WHERE id = ? AND
status = ?', [$certId, 'valid']);
    if (!$cert) {
        return ['success' => false, 'error' => 'Certificado no encontrado o
ya revocado'];
    }

    Database::update('certificates', [
        'status'            => 'revoked',
        'revocation_date'   => date('Y-m-d H:i:s'),
        'revocation_reason' => $reason,
        'revoked_by'        => $userId,
    ], 'id = ?', [$certId]);

    $user = Database::fetchOne('SELECT username FROM pki_users WHERE id = ?',
[$userId]);
    AuditLogger::certRevoked($userId, $user['username'] ?? 'unknown',
$certId, $cert['serial_number'], $reason);

    // Regenerar CRL automáticamente para que validadores externos vean la
revocación
    CRLManager::generate((int)$cert['ca_id'], 'revocation');

    return ['success' => true, 'serial' => $cert['serial_number']];
}
```

**Decisión.** Una revocación dispara inmediatamente la regeneración de la CRL (sin esperar al cron de 4 h). El OCSP responder consulta directamente la BD, así que ya está enterado al instante; pero los clientes que solo aceptan CRL (Windows en algunos casos) necesitan el fichero actualizado. La regeneración no bloquea al usuario: si falla, la siguiente ejecución del cron lo arreglará. El motivo se etiqueta como `revocation` en `crl_records.source` para distinguirlo de las ejecuciones automáticas.

### *A.5.4 Generación de CRL vía OpenSSL CLI*

```
public static function generate(int $caId, string $source = 'manual'): array
{
    $ca = Database::fetchOne('SELECT * FROM certificate_authorities WHERE id
= ? AND is_active = 1', [$caId]);
    if (!$ca) return ['success' => false, 'error' => 'CA no encontrada'];

    $caDir = PKIEngine::resolveCADir($ca);
    $passphrase = PKIEngine::decryptPassphrase(file_get_contents($caDir .
'/passphrase.key'));

    // Reconstruir index.txt desde la BD antes de regenerar la CRL
    self::updateOpenSSLIndex($caDir, $caId);

    $validityHours = (int)(pkiConfig('crl.validity_hours') ?? 24);
    $crlFile = $caDir . '/crl.pem';
    $crlDerFile = $caDir . '/crl.der';

    $cmd = sprintf(
        '%s ca -gencrl -config %s -out %s -passin pass:%s -cert %s -keyfile
%s -crldays %d 2>&1',
        escapeshellcmd(PKIEngine::getOpenSSLBin()),
        escapeshellarg($configFile), escapeshellarg($crlFile),
        escapeshellarg($passphrase), escapeshellarg($certFile),
        escapeshellarg($keyFile),
        (int)ceil($validityHours / 24) ?: 1
    );

    exec($cmd, $output, $returnCode);
    if ($returnCode !== 0) {
        return ['success' => false, 'error' => 'Error generando CRL: ' .
implode("\n", $output)];
    }

    // Convertir a DER y calcular SHA-256 para auditar la integridad
    exec(sprintf('%s crl -in %s -outform DER -out %s 2>&1', /* ... */));
    $crlDer = file_get_contents($crlDerFile);
    $sha256 = hash('sha256', $crlDer);

    $crlNumber = (Database::fetchOne(/* último crl_number */)['crl_number']
?? 0) + 1;

    Database::insert('crl_records', [
        'ca_id' => $caId, 'crl_number' => $crlNumber,
        'this_update' => date('Y-m-d H:i:s'),
        'next_update' => date('Y-m-d H:i:s', time() + ($validityHours *
3600)),
        'crl_pem' => $crlPem, 'crl_der' => $crlDer,
        'sha256_hash' => $sha256, 'source' => $source,
        /* ... */
    ]);

    self::publishCRL($ca, $crlDer, $crlPem);
    AuditLogger::crlGenerated($caId, $crlNumber);
    return ['success' => true, 'crl_number' => $crlNumber, /* ... */];
}
```

**Decisión 1:** BD como fuente de verdad**.** El `index.txt` de OpenSSL se reconstruye desde la BD cada vez (`updateOpenSSLIndex`), no se mantiene incrementalmente. Esto evita *drift* entre la fuente de verdad (BD) y los ficheros de OpenSSL: ningún script externo puede romper la consistencia, porque cada CRL parte de cero. El coste es despreciable (cientos de seriales como máximo en el horizonte de uso del TFG).

**Decisión 2:** regeneración 4 h con **`nextUpdate`** 24 h**.** La línea `(int)ceil($validityHours / 24) ?: 1` redondea `validity_hours = 4` a `-crldays 1`, lo que produce un DER cuyo `nextUpdate` está 24 h por delante. Esto es intencionado: el comando `openssl ca -gencrl -crldays N` solo acepta granularidad de días enteros, y emitir CRLs con `nextUpdate` < 24 h requeriría calcular `-startdate`/`-enddate` manualmente (no portable y susceptible a *clock drift*). El resultado práctico es favorable: con regeneración cada 4 h y validez nominal de 24 h, el sistema absorbe hasta seis rotaciones perdidas antes de quedar sin CRL válida en circulación. Documentado en Sección 7.8 del cuerpo y marcado como vía futura en Sección 9.4.

Independientemente del `nextUpdate` del DER, el repositorio público se refresca cada 4 h y la revocación dispara una regeneración inmediata bajo demanda (A.5.3): un cliente respetuoso del HTTP que valide la cabecera `Last-Modified` o `ETag` recibirá siempre la CRL más reciente.

### *A.5.5 OCSP responder con cache TTL y auditoría*

```
public static function handleRequest(string $rawBody): string
{
    $startTime = microtime(true);
    $serialNumber = null;

    try {
        if (empty($rawBody)) return self::buildMalformedResponse();

        $tempDir = pkiConfig('paths.private_base') . '/temp/ocsp_' .
bin2hex(random_bytes(8));
        PKIEngine::ensureDirectory($tempDir);

        $reqFile = $tempDir . '/request.der';
        file_put_contents($reqFile, $rawBody);

        $serialNumber = self::extractSerialFromRequest($reqFile);
        if (!$serialNumber) {
            self::cleanupTemp($tempDir);
            return self::buildMalformedResponse();
        }

        $cert = Database::fetchOne(
            'SELECT c.*, ca.type as ca_type FROM certificates c
             JOIN certificate_authorities ca ON ca.id = c.ca_id
             WHERE c.serial_number = ?',
            [$serialNumber]
        );

        $responseStatus = 'unknown';
        if ($cert) {
            if ($cert['status'] === 'valid' && strtotime($cert['not_after'])
>= time()) {
                $responseStatus = 'good';
            } elseif ($cert['status'] === 'revoked') {
                $responseStatus = 'revoked';
            }
        }

        $response = $cert
            ? self::buildOCSPResponseViaOpenSSL($reqFile, $respFile, $cert,
$tempDir)
            : self::buildBasicResponse($serialNumber, 'unknown', $tempDir,
$reqFile);

        // Auditar cada consulta OCSP: serie, estado, IP, latencia
        Database::insert('ocsp_queries', [
            'serial_number'    => $serialNumber,
            'ca_id'            => $cert['ca_id'] ?? null,
            'request_ip'       => AuditLogger::getClientIp(),
            'response_status'  => $responseStatus,
            'response_time_ms' => (int)((microtime(true) - $startTime) *
1000),
        ]);

        self::cleanupTemp($tempDir);
        return $response ?: self::buildMalformedResponse();
```

```
    } catch (\Throwable $e) {
        // ... logging del error ...
        return self::buildErrorResponse(2);
    }
}
```

**Decisión.** Cada consulta OCSP se audita con su `response_time_ms`, lo que permite medir el percentil 95 de latencia en el panel admin (Sección 7.7 del cuerpo). Cuando la BD del cert existe, OCSP se construye via OpenSSL CLI firmando preferentemente con un OCSP signer dedicado (delegated responder, RFC 6960 Sección 2.2.2), un certificado emitido específicamente con EKU `OCSPSigning` y `id-pkix-ocsp-nocheck` (cf. documento Plantillas OpenSSL (Sección F.4)) cuya clave privada no requiere passphrase para reducir la latencia de firma. Cuando el cert no se conoce, devolvemos `unknown` en una respuesta básica.

**Fallback explícito.** `PKIEngine::getOCSPSignerForCA()` devuelve `null` si la CA aún no tiene un signer dedicado emitido. En ese caso, el handler firma directamente con la clave privada de la propia CA tras descifrar su passphrase:

```
$ocspSigner = PKIEngine::getOCSPSignerForCA($ca);
if ($ocspSigner) {
    $signerCert = $ocspSigner['cert_file'];
    $signerKey  = $ocspSigner['key_file'];
    $passinArg  = '';                       // signer dedicado: sin
passphrase
} else {
    // Fallback: firmar con la propia CA (compatible con OpenSSL puro,
    // pero Windows rechaza este modo y exige delegated responder)
    $signerCert = $caCertFile;
    $signerKey  = $caDir . '/' . ($caType === 'root' ? 'root_ca.key' :
'intermediate_ca.key');
    $passphrase = PKIEngine::decryptPassphrase(file_get_contents($caDir .
'/passphrase.key'));
    $passinArg  = sprintf('-passin pass:%s', escapeshellarg($passphrase));
}
```

**Por qué documentar el fallback.** Es un compromiso operativo honesto: una CA recién creada todavía no tiene su OCSP signer emitido y, sin este fallback, las respuestas OCSP fallarían hasta que un administrador lo crease desde el panel. El precio es que en ese intervalo Windows rechazará las respuestas (Windows valida que el firmante de OCSP sea distinto de la propia CA, RFC 6960 Sección 4.2.2.2). Una vez emitido el signer, el flujo principal toma el control automáticamente.

## A.6 Gestión de passphrases y cifrado

### *A.6.1 Cifrado de passphrases con AES-256-CBC*

Cada CA tiene su propia passphrase aleatoria de 256 bits de entropía, codificada como 64 caracteres hexadecimales (`bin2hex(random_bytes(32))`). Esta passphrase nunca se almacena en BD: se guarda cifrada en `passphrase.key` en disco con permisos `0600`.

```
private static function encryptPassphrase(string $passphrase): string
{
    $key = pkiConfig('security.encryption_key');
    if (empty($key)) {
        return base64_encode($passphrase);   // modo desarrollo sin clave (no
usar en prod)
    }
    $iv = random_bytes(16);
    $encrypted = openssl_encrypt($passphrase, 'aes-256-cbc', $key, 0, $iv);
    // Usamos bin2hex para el IV para evitar conflictos con el separador '::'
    return base64_encode(bin2hex($iv) . '::' . $encrypted);
}
```

**Decisión. Dos niveles de protección**:

1. La clave privada de la CA (`*.key`) está cifrada por OpenSSL con la passphrase de la CA.
2. Esa passphrase no se guarda en claro: se cifra con AES-256-CBC usando `PKI_ENCRYPTION_KEY` (del `.env`).

Para descifrar y emitir un certificado, ambos secretos deben estar disponibles: la clave de cifrado del `.env` (fuera del repositorio) y el fichero `passphrase.key` (en disco con `chmod 600`). Esto se denomina en Sección 7.4.4 del cuerpo “cifrado jerárquico” (no “doble cifrado”) para distinguirlo de un HSM real.

El IV se codifica en hex dentro del formato porque un IV binario aleatorio puede contener bytes `0x3a 0x3a` (“`::`”), lo que rompería el parser ingenuo con `explode('::')`. Es un detalle pequeño pero crítico: hubo bugs reales antes del cambio a hex.

### *A.6.2 Comparación segura con `hash_equals`*

`hash_equals` se usa en todas las comparaciones de secretos del sistema:

| Uso | Ubicación |
|---|---|
| Comparar CSRF token | `sso/src/Security/CSRF.php` línea 37 |
| Comparar TOTP code | `sso/src/Auth/TOTPAuth.php` línea 164 |
| Verificar PKCE code_verifier | `sso/src/SSO/SSOProtocol.php` línea 138 |
| Comparar client_id en token exchange | `sso/src/SSO/SSOProtocol.php` línea 295 |
| Comparar fingerprint UA | `sso/src/Session/SessionManager.php` línea 69 |

**Decisión.** Las comparaciones de hashes y tokens deben ser constant-time. Una comparación con `===` puede revelar el secreto a un atacante con muchas peticiones por simple medición de tiempo. `hash_equals` recorre los dos strings hasta el final, aunque ya haya detectado una diferencia.

### *A.6.3 Recuperabilidad: re-cifrado tras rotación de clave maestra*

Cuando se rota `PKI_ENCRYPTION_KEY`, las passphrases existentes deben re-cifrarse:

```
public static function reEncryptPassphrase(string $caDir, string $oldKey):
bool
{
    $passphraseFile = $caDir . '/passphrase.key';
    if (!file_exists($passphraseFile)) return false;

    $data = file_get_contents($passphraseFile);
    $decoded = base64_decode($data);
    $passphrase = null;

    // Formato NUEVO: hex_iv(32 chars) + '::' + ciphertext
    if (strlen($decoded) > 34 && substr($decoded, 32, 2) === '::') {
        $hexIv = substr($decoded, 0, 32);
        if (ctype_xdigit($hexIv)) {
            $iv = hex2bin($hexIv);
            $enc = substr($decoded, 34);
            $passphrase = openssl_decrypt($enc, 'aes-256-cbc', $oldKey, 0,
$iv);
        }
    }

    // Formato ANTIGUO: raw_iv(16 bytes) + '::' + ciphertext
    if ($passphrase === false || $passphrase === null) {
        if (strlen($decoded) >= 18 && substr($decoded, 16, 2) === '::') {
            $iv = substr($decoded, 0, 16);
            $enc = substr($decoded, 18);
            $passphrase = openssl_decrypt($enc, 'aes-256-cbc', $oldKey, 0,
$iv);
        }
    }
    if ($passphrase === false || $passphrase === null || $passphrase === '')
{
        return false;
    }

    // Re-cifrar con la clave actual y formato nuevo
    $newEncrypted = self::encryptPassphrase($passphrase);
    copy($passphraseFile, $passphraseFile . '.bak');
    file_put_contents($passphraseFile, $newEncrypted);
    chmod($passphraseFile, 0600);
    return true;
}
```

**Decisión.** El sistema soporta migración entre formato**s** (IV binario antiguo → IV hex nuevo) sin perder passphrases. Se hace `.bak` del fichero original antes de sobrescribir, lo que da una vía de recuperación si el re-cifrado falla. Este código permite rotar la clave maestra del PKI sin pedir a un operador que reintroduzca todas las passphrases una a una.

## A.7 Middleware y seguridad transversal

### *A.7.1 CSRF: token rotado en cada validación*

```
public static function validate(?string $token): bool
{
    if (empty($token) || empty($_SESSION['csrf_token'])) {
        return false;
    }

    $csrfTime = $_SESSION['csrf_time'] ?? 0;
    if ((time() - $csrfTime) > 3600) {
        unset($_SESSION['csrf_token'], $_SESSION['csrf_time']);
        return false;
    }

    $valid = hash_equals($_SESSION['csrf_token'], $token);

    if ($valid) {
        self::generate();   // rotar token tras uso exitoso (single-use)
    }

    return $valid;
}
```

**Decisión.** Tres propiedades simultáneas:

1. Comparación constant-time con `hash_equals` (A.6.2).
2. Caducidad del token a la hora (mitiga sesiones zombi).
3. Single-use: tras una validación exitosa se rota. Esto bloquea ataques en los que un atacante recupera un CSRF token (XSS auto-cerrado, herramientas de debugging) e intenta reusarlo más tarde.

### *A.7.2 RateLimiter con cleanup periódico*

```
public static function check(string $identifier, string $action, int
$maxAttempts = RATE_LIMIT_MAX_ATTEMPTS, int $windowSeconds =
RATE_LIMIT_WINDOW): bool
{
    self::cleanup();

    $windowStart = date('Y-m-d H:i:s', time() - $windowSeconds);
    $stmt = Database::getInstance()->prepare(
        'SELECT SUM(attempts) as total FROM sso_rate_limits
         WHERE identifier = :id AND action = :action AND window_start >=
:window'
    );
    $stmt->execute(['id' => $identifier, 'action' => $action, 'window' =>
$windowStart]);
    $total = (int)($stmt->fetch(PDO::FETCH_ASSOC)['total'] ?? 0);

    return $total < $maxAttempts;
}

public static function hit(string $identifier, string $action): void
{
    Database::getInstance()->prepare(
        'INSERT INTO sso_rate_limits (identifier, action, attempts,
window_start)
         VALUES (:id, :action, 1, NOW())'
    )->execute(['id' => $identifier, 'action' => $action]);
}

private static function cleanup(): void
{
    static $lastCleanup = 0;
    if (time() - $lastCleanup < 300) return;   // máximo 1 cleanup cada 5 min
    $lastCleanup = time();

    Database::getInstance()->prepare(
        'DELETE FROM sso_rate_limits WHERE window_start < :cutoff'
    )->execute(['cutoff' => date('Y-m-d H:i:s', time() - 3600)]);
}
```

**Decisión.** El rate limiter es distributed-safe: el estado está en MariaDB, no en memoria de PHP. Esto significa que en un futuro despliegue con varios nodos PHP-FPM, el contador es consistente entre todos. La función `cleanup` se llama desde cada `check()` pero usa una variable estática para no ejecutar más de una vez cada 5 minutos: garantía probabilística de que la tabla nunca crece sin control, sin necesidad de un cron específico.

### *A.7.3 Security Headers: CSP con nonce por request*

```
public static function apply(): void
{
    $nonce = self::getNonce();

    header('X-Content-Type-Options: nosniff');
    header('X-Frame-Options: DENY');
    header('X-XSS-Protection: 0');
    header('Referrer-Policy: strict-origin-when-cross-origin');
    header('Permissions-Policy: camera=(), microphone=(), geolocation=()');
    header("Content-Security-Policy: default-src 'self'; "
        . "script-src 'self' 'nonce-{$nonce}' 'unsafe-hashes'; "
        . "style-src 'self' 'nonce-{$nonce}' https://fonts.googleapis.com; "
        . "style-src-attr 'unsafe-inline'; "
        . "font-src 'self' https://fonts.gstatic.com; "
        . "img-src 'self' data:; connect-src 'self'; "
        . "frame-ancestors 'none'; base-uri 'self'; "
        . "form-action 'self' https://*.marchernandez.es;");
    header('Strict-Transport-Security: max-age=31536000; includeSubDomains;
preload');
}
```

**Decisión.** CSP con nonce por request evita la necesidad de `'unsafe-inline'` para scripts: cada `<script>` legítimo lleva un atributo `nonce="{$nonce}"` que solo el servidor conoce. Un XSS reflejado o almacenado no podrá inyectar scripts ejecutables porque no conoce el nonce de esa petición.

El `frame-ancestors 'none'` y `X-Frame-Options: DENY` mitigan clickjacking. HSTS con `preload` registra el dominio para que los navegadores rechacen HTTP antes incluso de la primera conexión.

### *A.7.4 AuditLog con severidad automática por evento*

```
private const SEVERITY_MAP = [
    'auth_failed' => 'warning',
    'session_binding_violation' => 'critical',
    'refresh_binding_violation' => 'critical',
    'csrf_violation' => 'critical',
    'redirect_uri_blocked' => 'warning',
    'suspicious_ip_change' => 'critical',
    'rate_limit_exceeded' => 'warning',
    /* ... */
];

public static function log(string $event, ?string $userId, string $ipAddress,
array $details = [], ?string $severity = null): void
{
    $severity = $severity ?? (self::SEVERITY_MAP[$event] ?? 'info');

    try {
        Database::getInstance()->prepare(
            'INSERT INTO sso_audit_log (event_type, severity, user_id,
ip_address, details, created_at)
             VALUES (:event, :severity, :uid, :ip, :details, NOW())'
        )->execute([
            'event' => $event,
            'severity' => $severity,
            'uid' => $userId,
            'ip' => $ipAddress,
            'details' => json_encode($details, JSON_UNESCAPED_UNICODE),
        ]);
    } catch (\Throwable $e) {
        // Fallback: si la BD falla, al menos dejamos rastro en error_log de
PHP
        error_log("[MaHerMoSSO AuditLog] event={$event} severity={$severity}
user={$userId} ip={$ipAddress}");
    }
}
```

**Decisión.** Tres puntos importantes:

1. **Severidad automática:** cada evento crítico ya está mapeado en `SEVERITY_MAP`, así que un desarrollador que invoque `AuditLog::log('csrf_violation', ...)` no puede olvidar marcarlo como `critical` por accidente.

2. **Tolerancia a fallo de BD:** si la BD está caída, el evento se escribe a `error_log` para no perder trazabilidad. El cleanup automático (Manual del Administrador (Sección C.9)) preserva siempre los `critical` aunque hayan pasado 90 días.

3. `details` **JSON:** estructurado para hacer queries con `JSON_EXTRACT` (MariaDB 10.5+).

## A.8 Integración FNMT/DNIe (`CertificateAuth`)

### *A.8.1 EKU permitidos vs bloqueados*

```
private const BLOCKED_EKU_OIDS = [
    '1.3.6.1.5.5.7.3.1',       // TLS Web Server Authentication
    '1.3.6.1.5.5.7.3.3',       // Code Signing
    '1.3.6.1.5.5.7.3.8',       // Time Stamping
    '1.3.6.1.5.5.7.3.9',       // OCSP Signing
];

private const ALLOWED_EKU_OIDS = [
    '1.3.6.1.5.5.7.3.2',       // TLS Web Client Authentication
    '1.3.6.1.5.5.7.3.4',       // Email Protection
    '1.3.6.1.4.1.311.20.2.2',  // Smart Card Logon (Microsoft)
];

private const PERSON_CERT_POLICY_OIDS = [
    '2.16.724.1.2.2.4.1',      // FNMT persona física
    '2.16.724.1.2.2.4.2',      // FNMT persona jurídica
    '2.16.724.1.3.5.5.1',      // DNIe - Autenticación
    '2.16.724.1.3.5.5.2',      // DNIe - Firma
    '1.3.6.1.4.1.5734.3.10.1', // Certificado cualificado FNMT
    '1.3.6.1.4.1.55032.1.1',   // Omnipresence TrustCA
];
```

**Decisión.** El TFG bloquea explícitamente cuatro EKU que algunos atacantes podrían intentar usar para hacer login con certificados emitidos para otros fines:

- **TLS Server Auth:** un atacante con un cert de servidor TLS no puede usarlo para hacer login (suplantar a un usuario).
- **Code Signing:** evitar que una organización con CA propia para firmar binarios pueda colar el cert para SSO.
- **OCSP Signing / Time Stamping:** roles operacionales especializados que no deben confundirse con identidad de usuario.

Es una defensa antes que la confianza: aunque el certificado esté firmado por una CA que el sistema confía, el EKU lo rechaza si no es de tipo cliente o email.

### *A.8.2 Verificación cruzada de revocación contra BD del PKI*

```
// Verificar revocación contra la BD de PKI para certificados Omnipresence
if (!empty($serialNumber)) {
    $revocationCheck = self::checkPKIRevocation($serialNumber);
    if ($revocationCheck['revoked']) {
        return [
            'success' => false,
            'error' => 'El certificado ha sido revocado: ' .
$revocationCheck['reason'],
        ];
    }
}
```

**Decisión (Omnipresence TrustCA).** El método `checkPKIRevocation()` ejecuta una consulta de sólo lectura contra la tabla de certificados emitidos (`certificates`) del PKI usando la misma conexión PDO del SSO. Para que esa comprobación inmediata en el login por certificado funcione, debe existir uno de estos despliegues:

- el usuario MariaDB del SSO tiene **`SELECT`** explícito sobre la tabla `certificates` de la base de datos PKI (p. ej. `GRANT SELECT ON marchernandezmo_pki_nuevo.certificates TO 'admin_sso_nuevo_2'@'localhost'`), documentado en el Manual de Instalación (Sección A.4.2); o bien
- una vista (o recurso equivalente) en la BD del SSO que exponga sólo los campos necesarios y delegue en los datos PKI.

Si ese permiso cruzado no existe, las validaciones basadas en la cadena de confianza del cliente (consultas CRL/OCSP según CDP/AIA del propio certificado) siguen siendo válidas desde el navegador o el sistema operativo, pero esta ruta aplicativa de “revocado ya en BD” no puede aplicarse en el servidor SSO (los errores de consulta pueden quedar absorbidos sin bloquear el login). Para FNMT/DNIe la revocación no reside en esa tabla; una consulta OCSP contra el responder del emisor público permanece como línea futura (Sección 9.4).

### *A.8.3 Validación encadenada: EKU + Policy + Issuer en BD*

```
public static function validateCertificateType(string $certPem, string
$issuerDN): array
{
    if (empty($certPem)) {
        return ['valid' => false, 'error' => 'No se pudo leer el
certificado.', 'reason' => 'empty_cert'];
    }

    $certInfo = openssl_x509_parse($certPem);
    if ($certInfo === false) {
        return ['valid' => false, 'error' => 'No se pudo parsear el
certificado.', 'reason' => 'parse_error'];
    }

    $certEkus = self::extractEKUs($certInfo);
    $certPolicies = self::extractCertificatePolicies($certInfo);

    // 1. Bloquear OIDs explícitamente prohibidos para login
    foreach (self::BLOCKED_EKU_OIDS as $blockedOid) {
        if (in_array($blockedOid, $certEkus, true)) {
            return [
                'valid' => false,
                'error' => “Este tipo de certificado no puede usarse para
iniciar sesión.”,
                'reason' => 'blocked_eku',
            ];
        }
    }

    // 2. Verificar que al menos un EKU permitido esté presente
    if (!empty($certEkus)) {
        $hasAllowedEku = false;
        foreach (self::ALLOWED_EKU_OIDS as $allowedOid) {
            if (in_array($allowedOid, $certEkus, true)) {
                $hasAllowedEku = true;
                break;
            }
        }
        if (!$hasAllowedEku) {
            return ['valid' => false, 'error' => 'EKU inválido.', 'reason' =>
'no_allowed_eku'];
        }
    }

    // 3. Verificar restricciones del issuer en BD (allowed_eku,
allowed_policy_oids)
    $issuerRestriction = self::checkIssuerRestrictions($issuerDN, $certEkus,
$certPolicies);
    if ($issuerRestriction !== null) return $issuerRestriction;

    return ['valid' => true];
}
```

**Decisión.** Validación en tres capas:

1. **Lista negra de EKU** (defensa antes que la confianza).
2. **Lista blanca de EKU** (un cert sin ningún uso reconocido se rechaza).
3. **Restricciones por emisor en BD** (`sso_trusted_issuers.allowed_eku` y `allowed_policy_oids`).

Esto permite, por ejemplo, configurar que la FNMT solo se acepte para certificados con Policy OID de persona física o jurídica (no para certificados de empresa o de componente informático), sin tocar el código.

## A.9 Cron y automatización

### *A.9.1 Generación automática de CRL con lock*

```
$lockFile = $logDir . '/crl_generate.lock';
$lockFp = fopen($lockFile, 'c');
if (!$lockFp || !flock($lockFp, LOCK_EX | LOCK_NB)) {
    crlLog('SKIP: Otra instancia ya está ejecutándose (lock activo).',
$logFile);
    exit(0);
}

crlLog('=== Inicio generación automática de CRL ===', $logFile);
$exitCode = 0;

try {
    $cas = Database::fetchAll('SELECT id, name FROM certificate_authorities
WHERE is_active = 1');

    $generated = 0;
    $errors = 0;

    foreach ($cas as $ca) {
        crlLog("Generando CRL para CA #{$ca['id']} ({$ca['name']})...",
$logFile);
        try {
            $result = CRLManager::generate($ca['id'], 'cron');
            if ($result['success']) {
                crlLog("  OK: CRL #{$result['crl_number']} generada -
{$result['entries']} entradas", $logFile);
                $generated++;
            } else {
                crlLog("  ERROR: {$result['error']}", $logFile);
                $errors++;
            }
        } catch (\Throwable $e) {
            crlLog("  EXCEPTION: {$e->getMessage()}", $logFile);
            $errors++;
        }
    }

    crlLog("Resumen: {$generated} CRL generadas, {$errors} errores de " .
count($cas) . " CAs.", $logFile);
    $exitCode = $errors > 0 ? 1 : 0;
} finally {
    flock($lockFp, LOCK_UN);
    fclose($lockFp);
    @unlink($lockFile);
}

exit($exitCode);
```

**Decisión.** El lock con `flock()` evita que dos ejecuciones simultáneas del cron (cron solapado con cron manual desde el panel) corrompan los `index.txt` y `serial` de las CAs. Si el lock ya está tomado, el segundo proceso sale silenciosamente

con código 0 (no es un error: es “ya hay alguien haciendo el trabajo”). El exit code 1 se reserva para errores reales (alguna CA no se pudo regenerar), lo que permite que `systemd` o `cron` envíen alertas Telegram.

### *A.9.2 Cleanup periódico del SSO (orientado a privacidad)*

```
// 1. Sesiones expiradas
$sessionsDeleted = SessionManager::cleanExpiredSessions();

// 2. Códigos de autorización OAuth2 expirados (>24 h)
$db->prepare(
    'DELETE FROM sso_authorization_codes
     WHERE (expires_at < NOW() OR used = 1)
     AND created_at < DATE_SUB(NOW(), INTERVAL 24 HOUR)'
)->execute();

// 3. Tokens de reset de contraseña (>7 días)
$db->prepare(
    'DELETE FROM sso_password_resets
     WHERE (expires_at < NOW() OR used = 1)
     AND created_at < DATE_SUB(NOW(), INTERVAL 7 DAY)'
)->execute();

// 4. Rate limits antiguos (>24 h)
// ...

// 5. Audit logs >90 días - preservando críticos
$auditLogsDeleted = AuditLog::cleanup(90);

// 6. Login logs >180 días
// ...

// 7. Keypairs inactivos antiguos (>30 días) - borrar archivos físicos
también
$oldKeypairs = $db->query(
    'SELECT kid, key_file_path FROM sso_keypairs
     WHERE is_active = 0 AND created_at < DATE_SUB(NOW(), INTERVAL 30 DAY)'
)->fetchAll();

foreach ($oldKeypairs as $keypair) {
    $keyFilePath = PRIVATE_PATH . '/keys/' . $keypair['key_file_path'];
    if (file_exists($keyFilePath)) unlink($keyFilePath);
    $db->prepare('DELETE FROM sso_keypairs WHERE kid = :kid')->execute(['kid'
=> $keypair['kid']]);
}
```

**Decisión.** El cleanup cumple con RGPD art. 5 (minimización): cada categoría de datos tiene un periodo de retención justificado:

- 24 h para códigos OAuth (vida útil < 60 s, pero conservados un día para auditoría inmediata).

- 7 días para tokens de reset.
- 30 días para keypairs inactivos (margen para rollback tras una rotación).
- 90 días para audit logs excepto los críticos, que se conservan indefinidamente.
- 180 días para login logs.

Los eventos `critical` no se borran nunca (Manual del Administrador (Sección C.9)). Esto es importante para investigaciones forenses posteriores.

### *A.9.3 OCSP health check (script bash)*

```
check_ocsp() {
    local ca_slug="$1"
    local ca_dir="$PKI_DIR/$ca_slug"
    local ca_cert="$ca_dir/certs/ca.crt"
    local ocsp_url="http://ocsp.marchernandez.es/$ca_slug"

    if [[ ! -f "$ca_cert" ]]; then
        log "${RED}ERROR: No existe el certificado de la CA: $ca_cert${NC}"
        return 1
    fi

    local test_cert=$(find "$ca_dir/newcerts" -name "*.pem" -type f | head -n
1)

    if [[ -z "$test_cert" ]]; then
        log "${YELLOW}WARNING: No hay certificados emitidos para probar OCSP
de $ca_slug${NC}"
        return 0
    fi

    if timeout 10 openssl ocsp \
        -issuer "$ca_cert" -cert "$test_cert" \
        -url "$ocsp_url" -CAfile "$ca_cert" -noverify 2>&1 \
        | grep -q "Response verify OK"; then
        log "${GREEN}SUCCESS: OCSP responder operativo para $ca_slug${NC}"
        return 0
    else
        log "${RED}ERROR: OCSP responder no responde para $ca_slug${NC}"
        return 1
    fi
}
```

**Decisión.** Un script bash de 30 líneas, agendable por systemd timer cada 5 minutos. Verifica que **el** responder real responde a peticiones reales (no solo que el proceso esté vivo). Si una intermedia no tiene certificados emitidos todavía, el chequeo se salta sin marcar error.

## A.10 Configuración Apache/Nginx singular

### *A.10.1 Nginx con `ssl_verify_client optional` para subdominio cert-auth*

```
ssl_client_certificate
/var/www/vhosts/marchernandez.es/sso.marchernandez.es/config/trusted_ca_bundl
e.pem;
ssl_verify_depth 4;
ssl_verify_client optional;

proxy_set_header X-SSL-Client-Verify   $ssl_client_verify;
proxy_set_header X-SSL-Client-S-DN     $ssl_client_s_dn;
proxy_set_header X-SSL-Client-I-DN     $ssl_client_i_dn;
proxy_set_header X-SSL-Client-M-Serial $ssl_client_serial;
proxy_set_header X-SSL-Client-Cert     $ssl_client_escaped_cert;
proxy_set_header X-SSL-Client-V-End    $ssl_client_v_end;
```

**Decisión.** `ssl_verify_client optional` permite que el navegador presente certificado si quiere: usuarios con FNMT o DNIe lo pueden usar, los demás no ven popup. La validación criptográfica de la cadena la hace Nginx (capa TLS); la lógica de qué hacer con el cert válido la hace PHP (capa aplicación) - la defensa en capas entre TLS y lógica mencionada en Sección 7.6.2 del cuerpo.

Los headers `X-SSL-Client-*` reenvían los detalles del cert al backend PHP, que los lee desde `$_SERVER['SSL_CLIENT_VERIFY']`, etc.

### *A.10.2 `.htaccess` del SSO: cabeceras + bloqueo de directorios sensibles*

```
<IfModule mod_rewrite.c>
    RewriteEngine On

    # Forzar HTTPS
    RewriteCond %{HTTPS} off
    RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI} [L,R=301]

    # Bloquear acceso a archivos sensibles
    RewriteRule ^\.env$ - [F,L]
    RewriteRule ^\.git/ - [F,L]
    RewriteRule ^config/credentials\.php$ - [F,L]
</IfModule>

<IfModule mod_headers.c>
    Header always set Strict-Transport-Security “max-age=31536000;
includeSubDomains; preload”
    Header always set X-Content-Type-Options “nosniff”
    Header always set X-Frame-Options “DENY”
    Header always set Referrer-Policy “strict-origin-when-cross-origin”
    Header always set Permissions-Policy “camera=(), microphone=(),
geolocation=()”
</IfModule>

# Bloquear listado de directorios
Options -Indexes
```

### *A.10.3 Subdominios OCSP/CRL/CA explícitamente HTTP (no Let's Encrypt)*

En Plesk, los subdominios `ocsp`, `crl` y `ca` no se marcan para Let's Encrypt y no se activa “Forzar HTTPS”. El motivo, RFC 6960 Sección A.1.1 y RFC 5280 Sección 4.2.1.13:

A client that needs to validate a certificate's revocation status may need to fetch CRLs or contact an OCSP responder. If those endpoints use TLS, the client would have to validate that endpoint's certificate too, creating a circular dependency. Using HTTP for CRL/OCSP avoids this deadlock; integrity is provided by the cryptographic signatures on the responses themselves.

Esto está documentado en el `.htaccess` de cada subdominio público:

```
# crl.marchernandez.es/.htaccess
#
# Este subdominio se sirve OBLIGATORIAMENTE por HTTP, no por HTTPS.
# RFC 5280 Sección 4.2.1.13: las CRLs deben ser accesibles sin trust anchor
previo.
# La integridad la aporta la firma de la CA sobre la CRL, no TLS.

<IfModule mod_headers.c>
    Header set Cache-Control “public, max-age=3600”
    Header set Content-Type “application/pkix-crl”
</IfModule>
```

## A.11 Fragmentos SQL relevantes

El esquema completo está en el documento Modelo de Datos. Aquí solo se muestran las constraints e índices que reflejan decisiones de diseño concretas.

### *A.11.1 Single-source-of-truth para revocación*

```
-- Tabla certificates: el estado autoritativo de revocación está aquí.
CREATE TABLE certificates (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ca_id INT NOT NULL,
    serial_number VARCHAR(40) NOT NULL,
    common_name VARCHAR(255),
    status ENUM('valid','revoked','expired') NOT NULL DEFAULT 'valid',
    revocation_date DATETIME NULL,
    revocation_reason ENUM('unspecified','keyCompromise','cACompromise',

'affiliationChanged','superseded','cessationOfOperation',
                           'certificateHold','removeFromCRL',
                           'privilegeWithdrawn','aACompromise') NULL,
    revoked_by INT NULL,
    /* ... */
    UNIQUE KEY uq_ca_serial (ca_id, serial_number),
    INDEX idx_status (status),
    INDEX idx_serial (serial_number),
    CONSTRAINT fk_cert_ca FOREIGN KEY (ca_id)
        REFERENCES certificate_authorities(id) ON DELETE RESTRICT,
    CONSTRAINT fk_cert_revoked_by FOREIGN KEY (revoked_by)
        REFERENCES pki_users(id) ON DELETE SET NULL
);
```

**Decisión.** Cuatro puntos:

1. `UNIQUE (ca_id, serial_number)`: dos CAs distintas pueden emitir el mismo serial 1001 sin colisión (cada una mantiene su contador independiente).
2. `ENUM` **con los 10 motivos de revocación de RFC 5280 Sección 5.3.1**: si el formulario web envía un valor fuera del enum, MySQL lo rechaza.

3. `ON DELETE RESTRICT` **desde la CA**: no se puede borrar una CA mientras tenga certificados emitidos (preserva integridad referencial; el operador debe revocar todo y desactivar la CA, no borrarla).

4. `ON DELETE SET NULL` **desde** `revoked_by`: si un admin se borra, no se pierden los registros de revocación que él hizo (su autoría se anonimiza).

### *A.11.2 Índices para queries operativas*

```
-- Búsqueda de certificados activos próximos a expirar (dashboard PKI)
ALTER TABLE certificates ADD INDEX idx_status_not_after (status, not_after);

-- Consultas OCSP por serial (hot path)
ALTER TABLE certificates ADD INDEX idx_serial (serial_number);

-- Auditoría SSO filtrada por severidad + tiempo (queries del Manual del
Administrador (Sección C.9))
ALTER TABLE sso_audit_log ADD INDEX idx_severity_created (severity,
created_at);

-- Sesiones activas de un usuario
ALTER TABLE sso_sessions ADD INDEX idx_user_active (user_id, is_active,
expires_at);
```

**Decisión.** Cada índice corresponde a una query concreta que se ejecuta con frecuencia. Por ejemplo, `idx_severity_created` permite la query del Manual del Administrador “top IPs con eventos críticos en la última semana” en tiempo constante respecto al tamaño total de la tabla.

### *A.11.3 PKCE: columnas y constraints*

```
-- Migración 011: soporte PKCE (documento Modelo de Datos (Sección E.5.1))
ALTER TABLE sso_authorization_codes
    ADD COLUMN IF NOT EXISTS code_challenge VARCHAR(128) DEFAULT NULL
        COMMENT 'PKCE code_challenge (RFC 7636,
BASE64URL(SHA256(verifier)))',
    ADD COLUMN IF NOT EXISTS code_challenge_method VARCHAR(8) DEFAULT NULL
        COMMENT 'PKCE method: S256 (plain rechazado por seguridad)';

CREATE INDEX IF NOT EXISTS idx_pkce_method
    ON sso_authorization_codes (code_challenge_method);
```

**Decisión.** Migraciones idempotentes con `IF NOT EXISTS`. Esto permite que el script de instalación o el equipo de operaciones ejecuten la misma migración varias veces sin error 1060 (columna duplicada). Documentado en documento Modelo de Datos (Sección E.5).

## A.12 Métricas del código

### *A.12.1 Tamaño aproximado por componente*

| Componente | Archivos PHP | LOC (aprox.) | Notas |
|---|---|---|---|
| **SSO -** `src/` (núcleo, sin vistas) | 17 | 3.500 | Auth, Session, SSO, Security, Admin, Database, Utils |
| **SSO -** `public/api/` (endpoints OAuth/OIDC) | 6 | 700 | authorize, token, userinfo, jwks, logout, verify |
| **SSO -** `public/admin/` (panel) | 5 | 1.500 | users, applications, sessions, logs, dashboard |
| **SSO - vistas + templates** | ~10-15 | 2.500 | Plantillas Twig-like manuales (mayoría HTML) |
| **PKI -** `lib/` (motor + servicios) | 9 | 5.000 | PKIEngine es ~1.600 LOC por sí solo |
| **PKI -** `public/admin/` (panel) | 9 | 2.500 | ca, requests, certificates, crl, users, audit, ... |
| **PKI -** `public/` (portal usuario) | 11 | 2.000 | request, revoke, certificates, profile, ... |
| **PKI -** `cron/` **+** `bin/` (automatización) | 4 | 400 | generate_crl.php + 3 scripts bash |
| **OCSP responder** (`ocsp.marchernandez.es/`) | 1 | 150 | Index simple que delega en OCSPHandler |
| **CRL/CA repositorios** (`crl/` + `ca/`) | 4 | 400 | Listados públicos + download.php |

| Componente | Archivos PHP | LOC (aprox.) | Notas |
|---|---|---|---|
| **SQL - schema + 12 migraciones** | 13 | 1.500 | documento Modelo de Datos |
| **OpenSSL `.cnf` (Root + 6 perfiles)** | 7 | 300 | documento Plantillas OpenSSL |
| **Configuración Apache/Nginx/`.htaccess`** | ~12 | 250 | Sección A.10 |
| **Tests** (`sso/tests/`) | 7 | 1.200 | PKCE, JWT, smoke tests |
| **Documentación TFG** (Anexos + capítulos) | 25 | 13.000 | Markdown - este propio documento |
| **TOTAL código operativo** | ~110 | 22.000 | sin contar `vendor/`, backups ni snapshots |

### *A.12.2 Otras métricas*

| Métrica | Valor |
|---|---|
| Endpoints HTTP expuestos (documento API REST) | ~30 |
| Tablas en BD (SSO + PKI) | 24 |
| Migraciones SQL | 12 |
| Plantillas OpenSSL `.cnf` | 7 |
| Clases PHP (namespaces `MaHerMoSSO\*` + `MaHerMo\PKI\*`) | ~30 |
| Constantes globales en bootstrap | ~25 |
| Tests unitarios (PKCE solo) | 29 |

| Métrica | Valor |
|---|---|
| Idiomas soportados | 1 (es) |
| Dependencias Composer en producción | **0** |
| Dependencias Composer dev | 1 (PHPUnit) |
| Líneas de código por archivo (mediana) | ~180 |
| LOC por archivo más grande (`PKIEngine.php`) | ~1.600 |

**Lectura.** Tres datos pueden parecer pequeños, pero son decisiones intencionadas:

- **0 dependencias de runtime:** el sistema entero (autoloader, OAuth, JWT, CSRF, rate limiting, OCSP) está hecho a mano. Esto se discute extensamente en Sección 6 del cuerpo: para una organización pequeña, controlar la cadena de suministro vale más que la velocidad de desarrollo.
- **1 idioma:** español como única lengua soportada. La i18n es una vía futura (Sección 9.4) - no se estableció como objetivo estricto del TFG.
- **PKIEngine.php tiene 1.600 LOC:** monolítico a propósito. Cualquier intento prematuro de “dividir en clases más pequeñas” sin un dominio realmente estable acaba en un over-engineering peor que el monolito. Una refactorización futura aparece en Sección 9.4.

## A.13 Referencias internas

Cada fragmento de este anexo se referencia directamente desde el cuerpo del TFG:

| Fragmento | Sección del cuerpo |
|---|---|
| A.2 Bootstrap | Sección 7.10 (despliegue), Sección 5 (tecnologías) |

| Fragmento | Sección del cuerpo |
|---|---|
| A.3 Auth + JWT + binding | Sección 7.5 (SSO), Sección 7.6 (métodos de autenticación), Sección 7.9 (STRIDE Spoofing) |
| A.4 OAuth + PKCE + OIDC | Sección 7.5.1, Sección 7.5.2 |
| A.5 Motor PKI | Sección 7.4.1, Sección 7.4.3, Sección 7.4.5 |
| A.6 Cifrado / passphrases | Sección 7.4.4 (claves CA), Sección 7.9 (STRIDE Tampering) |
| A.7 Hardening transversal | Sección 7.8 (9 capas), Sección 7.9 (STRIDE) |
| A.8 FNMT/DNIe | Sección 7.6.2 |
| A.9 Cron | Sección 7.4.5 (CRL), Sección 7.7 (OCSP), Sección 7.10 (operación) |
| A.10 Apache/Nginx | Sección 7.5.2, Sección 7.10 |
| A.11 SQL | documento Modelo de Datos |
| A.12 Métricas | Sección 6 (planificación), Sección 9.2 (conclusiones técnicas) |

**Nota final.** El código completo del proyecto está disponible en el repositorio Git interno (consultar al autor para acceso). Los fragmentos seleccionados representan menos del 15 % del volumen total, pero concentran la mayor parte de las decisiones técnicas que sostienen los argumentos del TFG. Una lectura combinada de Sección 7 (Análisis y diseño) y este anexo debería ser suficiente para que un revisor entienda qué hace el sistema, por qué lo hace así, y dónde está el código que lo demuestra.