Seguridad y Privacidad en PAE

Principios de seguridad

1. **Privacy-first**: datos sensibles nunca salen del dispositivo
2. **Encryption**: datos en reposo encriptados
3. **Offline-first**: no depender de conectividad para funciones críticas
4. **Minimal permissions**: solo solicitar permisos necesarios
5. **Defense in depth**: múltiples capas de protección

Gestión de permisos

Android Manifest

```
<!-- AndroidManifest.xml -->

<!-- Permisos normales (automáticos) -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<!-- Permisos peligrosos (solicitar en runtime) -->
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!-- Hardware específico de Machine -->
<uses-permission android:name="android.permission.MODIFY_SYSTEM_SETTINGS" />
```

Solicitud de permisos en runtime

```
// Machine/src/main/java/co/ada/paemachine/permissions/PermissionManager.kt

class PermissionManager(private val activity: Activity) {
    fun requestCameraPermission() {
        val permission = Manifest.permission.CAMERA

        if (ContextCompat.checkSelfPermission(activity, permission)
            != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(
                activity,
                arrayOf(permission),
                CAMERA_PERMISSION_CODE
            )
```

```
        }
    }

    fun requestLocationPermission() {
        val permission = Manifest.permission.ACCESS_FINE_LOCATION

        if (ContextCompat.checkSelfPermission(activity, permission)
            != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(
                activity,
                arrayOf(permission),
                LOCATION_PERMISSION_CODE
            )
        }
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<String>,
        grantResults: IntArray
    ) {
        when (requestCode) {
            CAMERA_PERMISSION_CODE -> {
                if ((grantResults.isNotEmpty() &&
                     grantResults[0] == PackageManager.PERMISSION_GRANTED))
{
                    onCameraPermissionGranted()
                } else {
                    onCameraPermissionDenied()
                }
            }
        }
    }
}
```

Encriptación de datos

Encriptación en reposo (EncryptedSharedPreferences)

```
// Credenciales sensibles
val encryptedSharedPrefs = EncryptedSharedPreferences.create(
    context,
    "secret_shared_prefs",
    MasterKey.Builder(context)
        .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
        .build(),
    EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
```

```
    EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
)

// Guardar
encryptedSharedPrefs.edit().apply {
    putString("api_token", "secreto")
    putString("user_id", "12345")
    apply()
}

// Leer
val token = encryptedSharedPrefs.getString("api_token", null)
val userId = encryptedSharedPrefs.getString("user_id", null)
```

Encriptación en base de datos (EncryptedFile)

```
// Fotos de entregas y beneficiarios
val file = File(context.filesDir, "delivery_photo.jpg")

val encryptedFile = EncryptedFile.Builder(
    context,
    file,
    MasterKey.Builder(context)
        .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
        .build(),
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()

// Escribir
encryptedFile.openFileOutput().use { outputStream ->
    val photoBytes = bitmap.toByteArray()
    outputStream.write(photoBytes)
}

// Leer
val decryptedBytes = encryptedFile.openFileInput().use { inputStream ->
    inputStream.readBytes()
}
val bitmap = BitmapFactory.decodeByteArray(decryptedBytes, 0,
decryptedBytes.size)
```

SQLite con encriptación (SQLCipher)

```
// Database encriptada con contraseña
val dbPassword = "secret_database_passphrase"

val database = Room.databaseBuilder(
    context,
```

```
    DeliveryDatabase::class.java,
    "delivery_db.db"
)
.openHelperFactory(
    SQLCipherOpenHelperFactory { SQLiteDatabase.loadLibs(context);
dbPassword }
)
.build()
```

Almacenamiento seguro de credenciales

Keystore de Android

```
// Generar y guardar clave privada
val keyGenParameterSpec = KeyGenParameterSpec.Builder(
    "delivery_key",
    KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
).apply {
    setBlockModes(KeyProperties.BLOCK_MODE_GCM)
    setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
    setKeySize(256)
    setUserAuthenticationRequired(true)
    setUserAuthenticationValidityDurationSeconds(30)
}.build()

val keyGenerator = KeyGenerator.getInstance(
    KeyProperties.KEY_ALGORITHM_AES,
    "AndroidKeyStore"
)
keyGenerator.init(keyGenParameterSpec)
val secretKey = keyGenerator.generateKey()

// Recuperar clave
val keyStore = KeyStore.getInstance("AndroidKeyStore")
keyStore.load(null)
val key = keyStore.getKey("delivery_key", null) as SecretKey
```

Comunicación segura (HTTPS + Certificate Pinning)

Configuración de HTTP client

```
// MachineEndpoints.kt
val httpClient = HttpClientConfig()
    .socketTimeout(20000)
    .connectTimeout(20000)
```

```
    .setHOST("backend.api.com")
    .setProtocol("https")
    .build()

val httpClientFactory = FuelManager()
httpClientFactory.apply {
    basePath = "https://backend.api.com"
    client.setDefault(
        FuelManager.getInstance().client.setConfigBlock()
    )
}
```

Certificate Pinning

```
val certificatePinner = CertificatePinner.Builder()
    .add(
        "backend.api.com",
        "sha256/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="
    )
    .add(
        "backend.api.com",
        "sha256/BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB="
    )
    .build()

val okHttpClient = OkHttpClient.Builder()
    .certificatePinner(certificatePinner)
    .build()
```

Sanitización de datos

Prevención de SQL Injection

```
// MALO 
val query = "SELECT * FROM Beneficiary WHERE name = '$inputName'"

// BUENO 
val beneficiaries = repository.beneficiaries
    .where { name = inputName }
    .toList()
```

Prevención de XSS (si hay web component)

```
// MALO 
webView.loadData("<script>alert('hacked')</script>", "text/html", "utf-8")
```

```
// BUENO 
val escapedHtml = Html.escapeHtml(userInput)
webView.loadData(escapedHtml, "text/html; charset=utf-8", null)
```

Logging seguro

NO registrar datos sensibles

```
// MALO 
Log.d("API", "User token: $token")
Log.d("DB", "Password: $password")
Log.d("Sync", "Delivery data: $delivery")

// BUENO 
Log.d("API", "User authenticated successfully")
Log.d("DB", "Beneficiary record accessed")
Log.d("Sync", "Delivery synced - ID: ${delivery.id}")
```

Logger personalizado que oculta datos sensibles

```
object SecureLogger {
    fun d(tag: String, message: String) {
        if (BuildConfig.DEBUG) {
            Log.d(tag, message)
        }
    }

    fun logDelivery(delivery: Delivery) {
        d("SYNC", "Delivery ID: ${delivery.id}, Weight: ${delivery.weight}")
        // nunca loguear servicioDeliveryId o fotos
    }

    fun logToken(token: String) {
        d("AUTH", "Token length: ${token.length}, starts with:
${token.take(5)}")
    }
}
```

Protección contra tampering

Code Obfuscation (ProGuard/R8)

```
# proguard-rules.pro
```

```
-keep class co.ada.contract.models.** {
    public <init>(...);
}

-keepclassmembers class co.ada.domain.state.** {
    public <methods>;
}

-repackageclasses
-dontoptimize
```

Detección de root/jailbreak

```kotlin
// Machine/src/main/java/co/ada/paemachine/security/

object SecurityChecker {
    fun isDeviceSecure(context: Context): Boolean {
        return !isRooted() && !isDebuggerConnected()
    }

    private fun isRooted(): Boolean {
        val paths = arrayOf(
            "/system/app/Superuser.apk",
            "/sbin/su",
            "/system/bin/su",
            "/system/xbin/su",
            "/data/local/xbin/su"
        )
        return paths.any { File(it).exists() }
    }

    private fun isDebuggerConnected(): Boolean {
        return Debug.isDebuggerConnected()
    }
}
```

Data Privacy

Cumplimiento GDPR

1. Consentimiento

```kotlin
// Solicitar consentimiento inicial
if (!hasUserConsent()) {
    showPrivacyDialog {
        onAccepted = {
```

```
            savePrivacyConsent(true)
        }
        onDeclined = {
            exitApplication()
        }
    }
}
```

2. Transparencia

```
PRIVACY_POLICY.MD
- Qué datos se recopilan
- Cómo se almacenan
- Cuánto tiempo se retienen
- Derechos del usuario
```

3. Derecho al olvido

```
object DataDeletion {
    fun deleteUserData(userId: Long): Boolean {
        // Borrar de base de datos
        repository.deleteUserDeliveries(userId)
        repository.deleteUserBeneficiaries(userId)
        repository.deleteUserShifts(userId)

        // Borrar fotos
        deleteUserPhotoFiles(userId)

        // Borrar caché
        deleteUserCache(userId)

        return true
    }
}
```

Network Security

Network Security Configuration

```
<!-- res/xml/network_security_config.xml -->
<network-security-config>
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">backend.api.com</domain>
        <trust-anchors>
            <certificates src="@raw/backend_ca" />
```

```
        </trust-anchors>
    </domain-config>

    <!-- Denegar traffic en claro -->
    <domain-config cleartextTrafficPermitted="false">
        <domain includeSubdomains="true">.</domain>
    </domain-config>
</network-security-config>
```

AndroidManifest.xml

```
<application
    android:networkSecurityConfig="@xml/network_security_config"
    ...>
</application>
```

Verificación de integridad

Google Play Integrity API

```
// Machine/src/main/java/co/ada/paemachine/security/IntegrityChecker.kt

class IntegrityChecker(private val context: Context) {
    private val integrityClient = IntegrityClientFactory.create(context)

    suspend fun verifyAppIntegrity(): Boolean = withContext(Dispatchers.IO)
{
        return@withContext try {
            val response = integrityClient.requestIntegrityToken(
                IntegrityTokenRequest.Builder()
                    .setCloudProjectNumber(PROJECT_NUMBER)
                    .build()
            ).await()

            val tokenResponse = response.token()
            // Enviar tokenResponse a backend para verificar
            true
        } catch (e: Exception) {
            false
        }
    }
}
```

Checklist de seguridad ante deployment

- [ ] No hay secrets en código
- [ ] Permisos mínimos solicitados
- [ ] Encriptación habilitada para datos sensibles
- [ ] ProGuard/R8 habilitado en release
- [ ] HTTPS en todas las comunicaciones
- [ ] Certificate pinning configurado
- [ ] Logs no contienen datos sensibles
- [ ] Verifica OWASP Top 10 Mobile
- [ ] Pruebas de penetración completadas
- [ ] Privacy policy actualizado
- [ ] Code review completado
- [ ] Security patches aplicados

Referencias

- Android Security & Privacy Year in Review
- OWASP Mobile Top 10
- Google Play Security Best Practices
- Android Developers: Security