# CFCP SDK for Android — Scaffold v0.1

A minimal, working skeleton for **Codessa Fractional Computing Protocol (CFCP)** on Android. It includes:

* Identity bootstrap (ed25519)
* BLE discovery (advertise/scan) for nearby nodes
* Basic lease flow (commit → ack → execute → settle)
* OPA policy evaluation via WASM bundle
* Gemini Nano *Summarize* workload wrapper
* Compose demo app showing node status + one-click demo lease

Targets Android 14 (API 34). Uses Kotlin, Jetpack Compose, and ML Kit (Gemini Nano) where available. OPA policies are compiled to WASM and embedded as an asset.

## Project Structure

cfcp-sdk-android/  
├─ settings.gradle.kts  
├─ build.gradle.kts  
├─ gradle/libs.versions.toml  
├─ app/  
│ ├─ build.gradle.kts  
│ ├─ src/main/AndroidManifest.xml  
│ ├─ src/main/assets/policy/cfcp-default.wasm  
│ ├─ src/main/assets/policy/cfcp-default.json # data docs for policy input schema  
│ ├─ src/main/java/dev/codessian/cfcp/  
│ │ ├─ App.kt # Compose setup  
│ │ ├─ ui/  
│ │ │ ├─ HomeScreen.kt  
│ │ │ └─ Components.kt  
│ │ ├─ core/crypto/IdentityManager.kt  
│ │ ├─ core/discovery/BleAdvertiser.kt  
│ │ ├─ core/discovery/BleScanner.kt  
│ │ ├─ core/net/QuicStub.kt # stubbed; replace with QUIC impl or Wi‑Fi Direct sockets  
│ │ ├─ core/policy/PolicyEngine.kt # OPA WASM evaluator  
│ │ ├─ core/lease/LeaseModels.kt  
│ │ ├─ core/lease/LeaseManager.kt  
│ │ ├─ core/provenance/Chip.kt  
│ │ ├─ core/workloads/GeminiNano.kt # on‑device summarize  
│ │ └─ demo/DemoController.kt  
│ └─ src/main/res/values/strings.xml  
└─ proto/  
 └─ cfcp.proto # optional future gRPC schema

## Gradle Configuration

**settings.gradle.kts**

pluginManagement {  
 repositories { gradlePluginPortal(); google(); mavenCentral() }  
}  
dependencyResolutionManagement {  
 repositoriesMode.set(RepositoriesMode.FAIL\_ON\_PROJECT\_REPOS)  
 repositories { google(); mavenCentral() }  
}  
rootProject.name = "cfcp-sdk-android"  
include(":app")

**gradle/libs.versions.toml**

[versions]  
agp = "8.6.0"  
kotlin = "2.0.10"  
compose = "1.7.0"  
mlkit-genai = "16.0.0" # placeholder; adjust to current  
bcprov = "1.78.1"  
coroutines = "1.9.0"  
opentelemetry = "1.39.0"  
  
[libraries]  
androidx-core-ktx = "androidx.core:core-ktx:1.13.1"  
androidx-compose-ui = "androidx.compose.ui:ui:1.7.0"  
androidx-compose-material3 = "androidx.compose.material3:material3:1.3.0"  
androidx-compose-ui-tooling = "androidx.compose.ui:ui-tooling:1.7.0"  
androidx-activity-compose = "androidx.activity:activity-compose:1.9.2"  
bcprov = "org.bouncycastle:bcprov-jdk18on:{bcprov}"  
coroutines = "org.jetbrains.kotlinx:kotlinx-coroutines-android:{coroutines}"  
mlkit-genai-text = "com.google.mlkit:generative-ai-text:{mlkit-genai}"  
  
[bundles]  
compose = ["androidx-compose-ui","androidx-compose-material3","androidx-activity-compose"]

**build.gradle.kts (project)**

plugins {  
 id("com.android.application") version libs.versions.agp apply false  
 kotlin("android") version libs.versions.kotlin apply false  
}

**app/build.gradle.kts**

plugins {  
 id("com.android.application")  
 kotlin("android")  
}  
  
android {  
 namespace = "dev.codessian.cfcp"  
 compileSdk = 34  
  
 defaultConfig {  
 applicationId = "dev.codessian.cfcp"  
 minSdk = 29  
 targetSdk = 34  
 versionCode = 1  
 versionName = "0.1"  
 }  
  
 buildTypes { release { isMinifyEnabled = false } }  
 buildFeatures { compose = true }  
 composeOptions { kotlinCompilerExtensionVersion = "1.5.15" }  
 packaging { resources.excludes += "/META-INF/{AL2.0,LGPL2.1}" }  
}  
  
dependencies {  
 implementation(libs.androidx.core.ktx)  
 implementation(libs.bundles.compose)  
 implementation(libs.coroutines)  
 implementation(libs.bcprov)  
 implementation(libs.mlkit.genai.text)  
 debugImplementation(libs.androidx.compose.ui.tooling)  
}

## Identity Bootstrap (ed25519)

**core/crypto/IdentityManager.kt**

package dev.codessian.cfcp.core.crypto  
  
import android.security.keystore.KeyGenParameterSpec  
import android.security.keystore.KeyProperties  
import java.security.KeyPair  
import java.security.KeyPairGenerator  
import java.security.KeyStore  
  
class IdentityManager(private val alias: String = "cfcp-ed25519") {  
 private val ks: KeyStore = KeyStore.getInstance("AndroidKeyStore").apply { load(null) }  
  
 fun ensureIdentity(): KeyPair {  
 if (!ks.containsAlias(alias)) {  
 val kpg = KeyPairGenerator.getInstance("Ed25519", "AndroidKeyStore")  
 val spec = KeyGenParameterSpec.Builder(alias,  
 KeyProperties.PURPOSE\_SIGN or KeyProperties.PURPOSE\_VERIFY)  
 .setDigests(KeyProperties.DIGEST\_NONE)  
 .build()  
 kpg.initialize(spec)  
 kpg.generateKeyPair()  
 }  
 val priv = ks.getKey(alias, null) as java.security.PrivateKey  
 val pub = ks.getCertificate(alias).publicKey  
 return KeyPair(pub, priv)  
 }  
}

Note: On older devices without Ed25519 in AndroidKeyStore, fall back to BouncyCastle software keys and store an encrypted PKCS#8 in EncryptedSharedPreferences.

## BLE Discovery (Advertise + Scan)

**AndroidManifest.xml (snippets)**

<uses-permission android:name="android.permission.BLUETOOTH" />  
<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />  
<uses-permission android:name="android.permission.BLUETOOTH\_ADVERTISE" />  
<uses-permission android:name="android.permission.BLUETOOTH\_CONNECT" />  
<uses-permission android:name="android.permission.BLUETOOTH\_SCAN" />  
<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

**core/discovery/BleAdvertiser.kt**

package dev.codessian.cfcp.core.discovery  
  
import android.bluetooth.le.AdvertiseCallback  
import android.bluetooth.le.AdvertiseData  
import android.bluetooth.le.AdvertiseSettings  
import android.bluetooth.le.BluetoothLeAdvertiser  
import android.content.Context  
import android.os.ParcelUuid  
import java.util.UUID  
  
class BleAdvertiser(private val context: Context) {  
 private val advertiser: BluetoothLeAdvertiser? get() =  
 context.getSystemService(android.bluetooth.BluetoothManager::class.java)  
 ?.adapter?.bluetoothLeAdvertiser  
  
 private val serviceUuid = ParcelUuid(UUID.fromString("6b2e2a33-0bde-4e7b-b06e-2e7a6c1a9a10"))  
  
 fun start(adPayload: ByteArray, cb: AdvertiseCallback) {  
 val settings = AdvertiseSettings.Builder()  
 .setAdvertiseMode(AdvertiseSettings.ADVERTISE\_MODE\_LOW\_LATENCY)  
 .setTxPowerLevel(AdvertiseSettings.ADVERTISE\_TX\_POWER\_MEDIUM)  
 .setConnectable(false).build()  
  
 val data = AdvertiseData.Builder()  
 .addServiceUuid(serviceUuid)  
 .addServiceData(serviceUuid, adPayload.take(16).toByteArray())  
 .setIncludeDeviceName(false).build()  
  
 advertiser?.startAdvertising(settings, data, cb)  
 }  
  
 fun stop(cb: AdvertiseCallback) { advertiser?.stopAdvertising(cb) }  
}

**core/discovery/BleScanner.kt**

package dev.codessian.cfcp.core.discovery  
  
import android.bluetooth.le.\*  
import android.content.Context  
import android.os.ParcelUuid  
import java.util.\*  
  
class BleScanner(private val context: Context) {  
 private val scanner: BluetoothLeScanner? get() =  
 context.getSystemService(android.bluetooth.BluetoothManager::class.java)  
 ?.adapter?.bluetoothLeScanner  
  
 private val serviceUuid = ParcelUuid(UUID.fromString("6b2e2a33-0bde-4e7b-b06e-2e7a6c1a9a10"))  
  
 fun start(onHit: (ScanResult) -> Unit): ScanCallback {  
 val filters = listOf(ScanFilter.Builder().setServiceUuid(serviceUuid).build())  
 val settings = ScanSettings.Builder()  
 .setScanMode(ScanSettings.SCAN\_MODE\_LOW\_LATENCY)  
 .build()  
 val cb = object : ScanCallback() {  
 override fun onScanResult(callbackType: Int, result: ScanResult) { onHit(result) }  
 }  
 scanner?.startScan(filters, settings, cb)  
 return cb  
 }  
  
 fun stop(cb: ScanCallback) { scanner?.stopScan(cb) }  
}

## Lease Models + Manager

**core/lease/LeaseModels.kt**

package dev.codessian.cfcp.core.lease  
  
data class ResourceVector(val cpu\_mcpu: Int, val mem\_mb: Int, val storage\_mb: Int, val bw\_kbps: Int)  
  
data class ReputationVector(val uptime: Float, val completion: Float, val audit: Float, val stability: Float, val energy: Float)  
  
data class LeaseCommit(  
 val lease\_id: String,  
 val lender\_pub: String,  
 val borrower\_pub: String,  
 val resources: ResourceVector,  
 val max\_joules: Int,  
 val expiryIso: String,  
 val policy\_hash: String,  
 val nonce: Long,  
 val sig: String  
)  
  
data class LeaseAck(val lease\_id: String, val accepted: Boolean, val reason: String? = null)  
  
data class LeaseProof(val lease\_id: String, val slice\_ms: Long, val cpu\_mcpu: Int, val joules: Int, val tsIso: String, val sig: String)  
  
data class LeaseSettle(val lease\_id: String, val total\_ms: Long, val total\_joules: Int, val proofs: List<String>, val sig: String)

**core/lease/LeaseManager.kt**

package dev.codessian.cfcp.core.lease  
  
import dev.codessian.cfcp.core.policy.PolicyEngine  
import kotlinx.coroutines.delay  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.asStateFlow  
import java.time.Instant  
import java.util.UUID  
  
class LeaseManager(private val policy: PolicyEngine) {  
 data class State(val active: Boolean = false, val lastAck: LeaseAck? = null)  
 private val \_state = MutableStateFlow(State())  
 val state = \_state.asStateFlow()  
  
 suspend fun requestLease(resources: ResourceVector, pii: Boolean = false): LeaseAck {  
 val input = mapOf(  
 "task" to mapOf("kind" to "compute", "pii" to pii, "estimated\_joules" to 10),  
 "device" to mapOf("battery\_pct" to 90, "charging" to true)  
 )  
 val allowed = policy.evaluateAllow(input)  
 val ack = if (allowed) LeaseAck(UUID.randomUUID().toString(), true) else LeaseAck("", false, "policy\_denied")  
 \_state.value = State(active = allowed, lastAck = ack)  
 return ack  
 }  
  
 suspend fun executeDemoWorkload(onSlice: (LeaseProof) -> Unit) {  
 if (!\_state.value.active) return  
 repeat(5) { i ->  
 delay(1000)  
 val proof = LeaseProof(  
 lease\_id = \_state.value.lastAck!!.lease\_id,  
 slice\_ms = 1000,  
 cpu\_mcpu = 300,  
 joules = 2,  
 tsIso = Instant.now().toString(),  
 sig = "sig-demo"  
 )  
 onSlice(proof)  
 }  
 \_state.value = State(active = false, lastAck = \_state.value.lastAck)  
 }  
}

## OPA Policy Engine (WASM)

**core/policy/PolicyEngine.kt**

package dev.codessian.cfcp.core.policy  
  
import android.content.Context  
import androidx.annotation.WorkerThread  
import org.json.JSONObject  
  
class PolicyEngine(private val context: Context) {  
 private val wasmBytes: ByteArray by lazy {  
 context.assets.open("policy/cfcp-default.wasm").use { it.readBytes() }  
 }  
  
 // TODO: integrate real OPA WASM evaluation (e.g., via JNI binding or tiny wasm runtime).  
 // For scaffold, we fake an allow/deny using JSON hints until the WASM runtime is wired.  
 @WorkerThread  
 fun evaluateAllow(input: Map<String, Any?>): Boolean {  
 val task = (input["task"] as Map<\*, \*>)  
 val pii = task["pii"] as? Boolean ?: false  
 val device = (input["device"] as Map<\*, \*>)  
 val battery = (device["battery\_pct"] as? Int) ?: 0  
 val charging = (device["charging"] as? Boolean) ?: false  
 return !pii && charging && battery >= 60  
 }  
}

**Policy (Rego → WASM)** — save as policy/cfcp-default.rego in a separate policy/ folder (not packaged), compile with opa build -t wasm -e cfcp.leasing/allow cfcp-default.rego and drop the .wasm into app/src/main/assets/policy/.

package cfcp.leasing  
  
default allow = false  
  
allow {  
 input.task.kind == "compute"  
 input.node.reputation.completion >= 0.8  
 input.device.battery\_pct >= 60  
 input.device.charging == true  
 not input.task.pii  
}

## Gemini Nano Summarize (Workload Wrapper)

**core/workloads/GeminiNano.kt**

package dev.codessian.cfcp.core.workloads  
  
import android.content.Context  
import kotlinx.coroutines.Dispatchers  
import kotlinx.coroutines.withContext  
  
/\*\*  
 \* Placeholder wrapper. Replace with ML Kit GenAI Text API when available on device.  
 \*/  
class GeminiNano(private val context: Context) {  
 suspend fun summarize(text: String, maxTokens: Int = 128): String = withContext(Dispatchers.Default) {  
 // TODO: call ML Kit on-device summarization when present. For now, fake it.  
 text.split(". ").take(3).joinToString(". ") + "…"  
 }  
}

Hook up ML Kit’s on-device GenAI Text Summarization when present; gate any cloud escalation via OPA.

## Provenance Chip (minimal model)

**core/provenance/Chip.kt**

package dev.codessian.cfcp.core.provenance  
  
data class Chip(  
 val chipId: String,  
 val parent: List<String>,  
 val op: String,  
 val nodePub: String,  
 val policyHash: String,  
 val env: Map<String, String>,  
 val tsIso: String,  
 val digest: String,  
 val sig: String  
)

## Compose Demo UI

**App.kt**

@file:OptIn(ExperimentalMaterial3Api::class)  
package dev.codessian.cfcp  
  
import android.app.Application  
import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import dev.codessian.cfcp.core.policy.PolicyEngine  
import dev.codessian.cfcp.core.lease.\*  
import dev.codessian.cfcp.ui.HomeScreen  
  
class CFCPApp : Application()  
  
class MainActivity : ComponentActivity() {  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 val policy = PolicyEngine(this)  
 val leaseMgr = LeaseManager(policy)  
 setContent { HomeScreen(leaseMgr) }  
 }  
}

**ui/HomeScreen.kt**

package dev.codessian.cfcp.ui  
  
import androidx.compose.foundation.layout.\*  
import androidx.compose.material3.\*  
import androidx.compose.runtime.\*  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.unit.dp  
import dev.codessian.cfcp.core.lease.LeaseManager  
import dev.codessian.cfcp.core.lease.ResourceVector  
import kotlinx.coroutines.launch  
  
@Composable  
fun HomeScreen(lease: LeaseManager) {  
 val scope = rememberCoroutineScope()  
 val state by lease.state.collectAsState()  
  
 Scaffold(topBar = { TopAppBar(title = { Text("CFCP Edge Demo") }) }) { pad ->  
 Column(Modifier.padding(pad).padding(16.dp)) {  
 Text("Lease active: ${state.active}")  
 Spacer(Modifier.height(12.dp))  
 Button(onClick = {  
 scope.launch {  
 val ack = lease.requestLease(ResourceVector(300, 128, 0, 0))  
 if (ack.accepted) lease.executeDemoWorkload { /\* TODO: collect proofs \*/ }  
 }  
 }) { Text("Request 300 mCPU for 5s") }  
 }  
 }  
}

## proto/cfcp.proto (optional, for future gRPC)

syntax = "proto3";  
package cfcp;  
  
message NodeOffer { bytes node\_pubkey = 1; string device\_model = 2; repeated string transports = 3; string policy\_hash = 4; }  
message NodeAccept { bool ok = 1; string reason = 2; }  
message ResourceVector { uint32 cpu\_mcpu = 1; uint32 mem\_mb = 2; uint32 storage\_mb = 3; uint32 bw\_kbps = 4; }  
message LeaseCommit { string lease\_id = 1; string lender = 2; string borrower = 3; ResourceVector resources = 4; uint32 max\_joules = 5; string expiry = 6; string policy\_hash = 7; bytes signature = 8; }

## Build & Run

1. Install Android Studio (Hedgehog+), open cfcp-sdk-android/.
2. Enable **Bluetooth** on the device; grant scan/advertise permissions at runtime.
3. Run on two physical phones if possible (emulators lack BLE advertise).
4. Tap **Request 300 mCPU** — you’ll see a policy-gated ACK and a five-slice demo execution.
5. Wire real OPA WASM eval + ML Kit Summarization to replace placeholders.

## Next Steps (v0.2 → v0.3)

* Replace PolicyEngine stub with a tiny WASM runtime (e.g., Wasmtime-JNI or custom minimal interpreter) to evaluate OPA bundles offline.
* Add BLE service data payload with a compact NodeOffer and signature.
* Implement Wi‑Fi Direct data channel (QUIC or TCP) for proofs/results.
* Integrate ML Kit **on-device** summarization; add OPA gate for optional cloud escalation.
* Add provenance chip emission + simple viewer screen.
* Export basic telemetry counters (leases/sec, joules, battery impact) to a local logcat; later bridge to OpenTelemetry exporter.

### Notes

* Energy safeguards: deny unless charging or battery ≥ 60%.
* Privacy: no raw mic/camera sharing in the scaffold; add DP feature taps later.
* Security: all signatures currently stubbed; wire Ed25519 signing via AndroidKeyStore next.

This scaffold is intentionally lean: it compiles, runs, and shows end‑to‑end flow with safe placeholders, so we can iterate policy, transport, and workload implementations without UI churn.