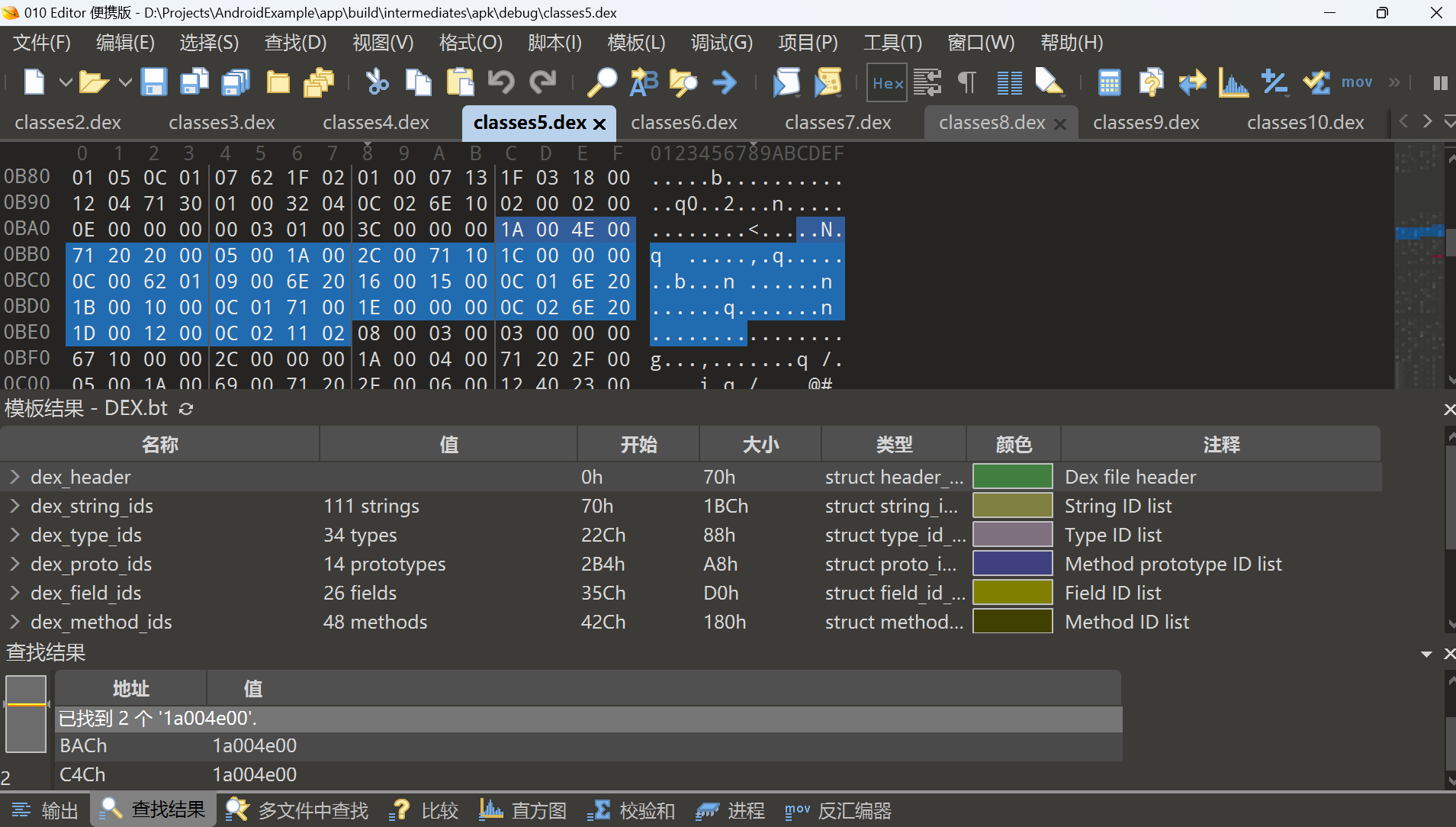
上一篇【[详解如何自定义 Android Dex VMP 保护壳](https://cyrus-studio.github.io/blog/posts/%E8%AF%A6%E8%A7%A3%E5%A6%82%E4%BD%95%E8%87%AA%E5%AE%9A%E4%B9%89-android-dex-vmp-%E4%BF%9D%E6%8A%A4%E5%A3%B3/)】实现了 VMP 保护壳。

为了进一步加强对 dex 指令的保护，实现指令流加密和动态加载，比如使用 AES 加密指令流，在运行时解密执行。

**保存指令流到文件**

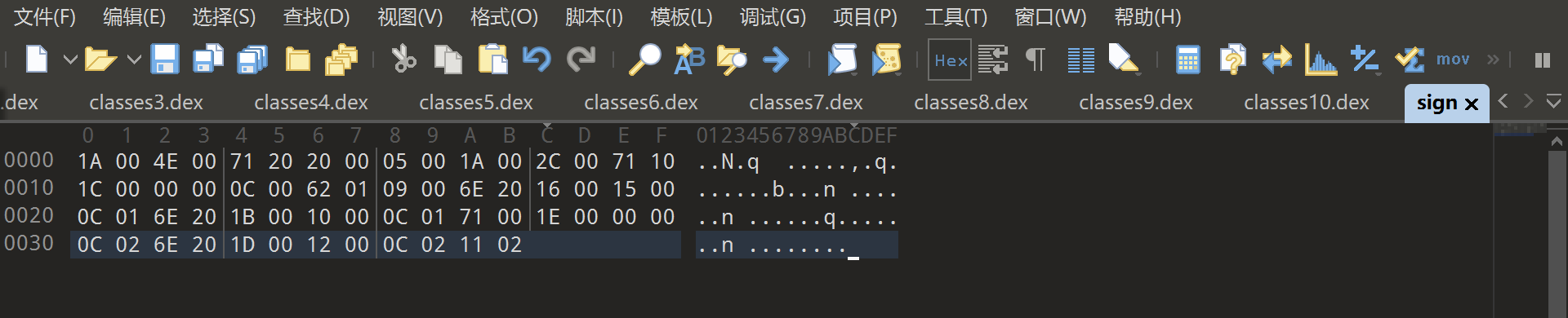
在 010Editor 中搜索找到 sign 方法的字节码并复制



新建 Hex 文件



把 sign 方法字节码粘贴到新建的文件保存文件为 sign



**AES加解密**

编写一个 kotlin 语言 AES 加解密算法工具类

package com.cyrus.vmp

import java.io.ByteArrayOutputStream

import java.io.File

import java.io.FileInputStream

import java.io.FileOutputStream

import javax.crypto.Cipher

import javax.crypto.KeyGenerator

import javax.crypto.SecretKey

import javax.crypto.spec.SecretKeySpec

object AESUtils {

private const val ALGORITHM = "AES"

private const val TRANSFORMATION = "AES/ECB/PKCS5Padding" // AES 加密模式

// 生成一个 128 位的 AES 密钥

fun generateSecretKey(): SecretKey {

val keyGenerator = KeyGenerator.getInstance(ALGORITHM)

keyGenerator.init(128) // AES 128 位

return keyGenerator.generateKey()

}

// 使用给定的密钥加密数据

fun encrypt(data: ByteArray, key: SecretKey): ByteArray {

val cipher = Cipher.getInstance(TRANSFORMATION)

cipher.init(Cipher.ENCRYPT\_MODE, key)

return cipher.doFinal(data)

}

// 使用给定的密钥解密数据

fun decrypt(data: ByteArray, key: SecretKey): ByteArray {

val cipher = Cipher.getInstance(TRANSFORMATION)

cipher.init(Cipher.DECRYPT\_MODE, key)

return cipher.doFinal(data)

}

// 将文件内容加密并导出到新文件

fun encryptFile(inputFile: File, outputFile: File, keyFile: File) {

// 读取文件内容

val fileData = readFile(inputFile)

// 生成密钥

val secretKey = generateSecretKey()

// 加密文件内容

val encryptedData = encrypt(fileData, secretKey)

// 保存加密后的数据到新文件（.vmp 文件）

writeFile(outputFile, encryptedData)

// 保存密钥到文件

saveKeyToFile(secretKey, keyFile)

}

// 解密文件内容并导出到新文件

fun decryptFile(inputFile: File, outputFile: File, keyFile: File) {

// 从文件加载密钥

val secretKey = loadKeyFromFile(keyFile)

// 读取加密后的文件内容

val encryptedData = readFile(inputFile)

// 解密文件内容

val decryptedData = decrypt(encryptedData, secretKey)

// 保存解密后的数据到文件

writeFile(outputFile, decryptedData)

}

// 读取文件内容并返回字节数组

fun readFile(file: File): ByteArray {

val fis = FileInputStream(file)

val baos = ByteArrayOutputStream()

val buffer = ByteArray(1024)

var bytesRead: Int

while (fis.read(buffer).also { bytesRead = it } != -1) {

baos.write(buffer, 0, bytesRead)

}

fis.close()

return baos.toByteArray()

}

// 将字节数组写入到文件

fun writeFile(file: File, data: ByteArray) {

val fos = FileOutputStream(file)

fos.write(data)

fos.close()

}

// 保存密钥到文件

private fun saveKeyToFile(key: SecretKey, keyFile: File) {

val fos = FileOutputStream(keyFile)

fos.write(key.encoded)

fos.close()

}

// 从文件加载密钥

fun loadKeyFromFile(keyFile: File): SecretKey {

val keyBytes = ByteArray(keyFile.length().toInt())

val fis = FileInputStream(keyFile)

fis.read(keyBytes)

fis.close()

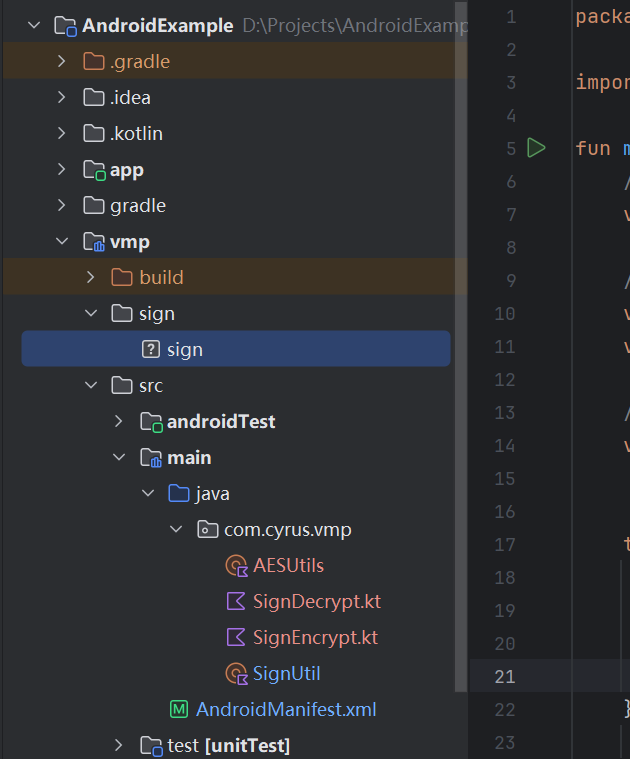
return SecretKeySpec(keyBytes, ALGORITHM)

}

}

**指令流加密**

把 sign 文件放到工程中如下路径



调用 AESUtils 类中方法对 sign 进行加密并输出加密文件和密钥

package com.cyrus.vmp

import java.io.File

fun main() {

// 获取工程根目录路径

val projectRoot = System.getProperty("user.dir")

// 设置相对路径

val encryptedFile = File(projectRoot, "vmp/sign/sign.vmp") // 相对路径

val keyFile = File(projectRoot, "vmp/sign/sign.key") // 相对路径

// 输入文件路径

val inputFile = File(projectRoot, "vmp/sign/sign") // 需要加密的文件

try {

// 使用 AES 加密文件

AESUtils.encryptFile(inputFile, encryptedFile, keyFile)

println("File encryption completed, saved as: ${encryptedFile.absolutePath}")

println("Key saved as: ${keyFile.absolutePath}")

} catch (e: Exception) {

e.printStackTrace()

}

}

**指令流解密**

package com.cyrus.vmp

import com.cyrus.vmp.AESUtils.loadKeyFromFile

import com.cyrus.vmp.AESUtils.readFile

import com.cyrus.vmp.AESUtils.writeFile

import java.io.File

fun main() {

// 获取工程根目录路径

val projectRoot = System.getProperty("user.dir")

// 输入加密文件路径

val encryptedFile = File(projectRoot, "vmp/sign/sign.vmp")

// 密钥文件路径

val keyFile = File(projectRoot, "vmp/sign/sign.key")

// 输出解密文件路径

val decryptedFile = File(projectRoot, "vmp/sign/sign\_")

try {

// 从文件加载密钥

val secretKey = loadKeyFromFile(keyFile)

// 解密文件

val encryptedData = readFile(encryptedFile)

val decryptedData: ByteArray = AESUtils.decrypt(encryptedData, secretKey)

// 保存解密后的文件

writeFile(decryptedFile, decryptedData)

println("File decryption completed, saved as: ${decryptedFile.absolutePath}")

} catch (e: Exception) {

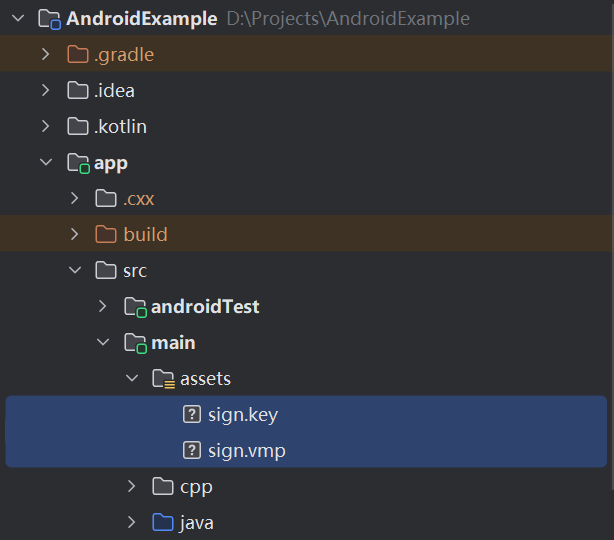
e.printStackTrace()

}

}

**Android 中运行时解密并执行指令流**

将 .vmp 和 .key 文件放在 Android 应用的 assets 目录下



编写工具类，用于读取 assets 文件并解密

package com.cyrus.example.vmp

import android.content.Context

import java.io.InputStream

import javax.crypto.Cipher

import javax.crypto.SecretKey

import javax.crypto.spec.SecretKeySpec

object AESUtils {

private const val ALGORITHM = "AES"

private const val TRANSFORMATION = "AES/ECB/PKCS5Padding"

// 从 assets 中读取文件并解密

fun decryptFileFromAssets(context: Context, vmpFileName: String, keyFileName: String): ByteArray? {

// 读取密钥文件

val key = loadKeyFromAssets(context, keyFileName)

// 读取加密的 vmp 文件

val encryptedData = readFileFromAssets(context, vmpFileName)

// 解密

return decrypt(encryptedData, key)

}

// 读取文件内容为字节数组

private fun readFileFromAssets(context: Context, fileName: String): ByteArray {

val inputStream: InputStream = context.assets.open(fileName)

return inputStream.readBytes()

}

// 从 assets 中加载密钥文件

private fun loadKeyFromAssets(context: Context, keyFileName: String): SecretKey {

val keyBytes = readFileFromAssets(context, keyFileName)

return SecretKeySpec(keyBytes, ALGORITHM)

}

// 解密

private fun decrypt(data: ByteArray, key: SecretKey): ByteArray {

val cipher = Cipher.getInstance(TRANSFORMATION)

cipher.init(Cipher.DECRYPT\_MODE, key)

return cipher.doFinal(data)

}

}

调用解密方法并读取指令流

private fun readInstructionFromAssets(): ByteArray? {

// 文件名：在 assets 中放置的加密文件和密钥文件

val vmpFileName = "sign.vmp"

val keyFileName = "sign.key"

// 解密文件

val decryptedData = AESUtils.decryptFileFromAssets(this, vmpFileName, keyFileName)

return decryptedData

}

得到解密后的指令流后调用 VMP 执行指令流对 input 参数加密

val input = "example"

// 解密并执行指令流

val bytecode = readInstructionFromAssets()

// 通过 VMP 解析器执行指令流

if (bytecode != null) {

val result = SimpleVMP.execute(bytecode, input)

// 显示 Toast

Toast.makeText(this, result, Toast.LENGTH\_SHORT).show()

}

**测试**

执行结果如下



和原来的 sign 算法对比是结果是一样的。



**源码**

完整源码：<https://github.com/CYRUS-STUDIO/AndroidExample>