


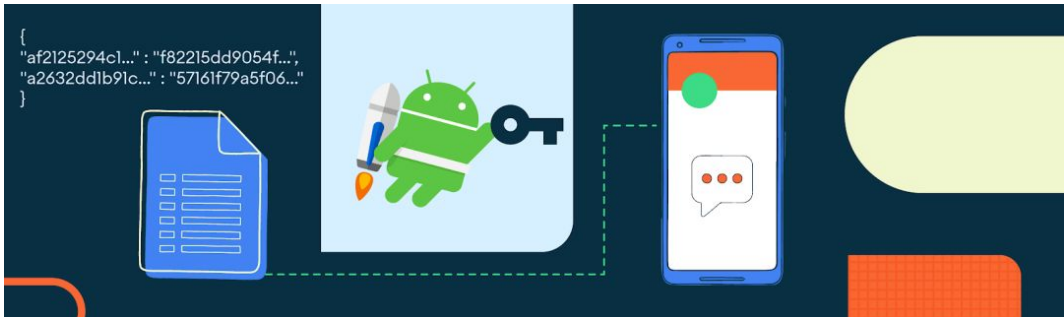
使用 Jetpack Security 在 Android 上进行数据加密



谷歌开发者 
已认证的官方帐号

已关注

5 人赞同了该文章



作者 / Jon Markoff, Staff Developer Advocate, Android Security

您是否尝试过对应用中的数据加密？作为开发者，您想要保护数据安全，并确保数据掌握在其合理使用者的手中。但是，大多数 Android 开发者没有专门的安全团队来帮助他们正确地加密应用数据。就算通过网络来搜索如何加密数据，您得到的答案也可能已经过时好几年了，找到的示例也难以保证准确性。

Jetpack Security (JetSec)加密库为 Files 和 SharedPreferences 对象的加密操作提供了抽象支持。该库使用了安全且运用广泛的密码学原语 (cryptographic primitives)，强化了 AndroidKeyStore 的使用。使用 EncryptedFile 和 EncryptedSharedPreferences 可以让您在本地保护可能包含敏感数据、API 密钥、OAuth 令牌和其他类型机密信息的文件。

从 5.0 开始，Android 会默认对用户数据分区的内容进行加密，那您为什么还需要加密应用中的数据呢？这是因为在某些场合中，您可能需要额外的保护。如果您的应用使用共享存储(shared storage)，则应该对数据进行加密。如果您的应用处理敏感信息，包括但不限于个人身份可识别信息 (Personally Identifiable Information, PII)、健康记录、财务信息或企业数据，那么您的应用应该对其主目录中的数据进行加密。如果可能，我们建议您将此类信息与生物验证操作绑定，以提供额外的保护。

Jetpack Security 基于Tink，而 Tink 是 Google 的一个开源并支持跨平台的安全项目。如果您需要常规加密、混合加密或类似的安全措施，那么 Tink 可能适用于您的项目。Jetpack Security 的数据结构与 Tink 完全兼容。

密钥生成

在开始加密数据之前，首先要了解您的加密密钥是如何被保护的。Jetpack Security 使用一个主密钥 (master key) 对所有的子密钥 (subkey) 进行加密，子密钥则被用于每个加密操作。JetSec 在 MasterKeys 类中提供了建议的默认主密钥。这个类使用基础的 AES256-GCM 密钥，该密钥在 AndroidKeyStore 中生成并存储。AndroidKeyStore 是一个在 TEE 或 StrongBox 中存储加密密钥的容器，这使得其内容很难被提取。子密钥则存储在可配置的 SharedPreferences 对象中。

我们在 Jetpack Security 中主要使用 AES256_GCM_SPEC 规范，在一般的用例中很推荐使用该规范。AES256-GCM 是对称的，并且在现代设备上运算的速度通常很快。

```
val keyAlias = MasterKeys.getOrCreate(MasterKeys.AES256_GCM_SPEC)
```

对于配置更多样或处理非常敏感数据的应用，我们建议您构建自己的KeyGenParameterSpec，选

- **userAuthenticationRequired()** 和 **userAuthenticationValiditySeconds()** 可以用来创建限时密钥。限时密钥需要通过 BiometricPrompt 获得授权，才能对对称密钥进行加密和解密。
- **unlockedDeviceRequired()** 可以设置一个标志，用于确保在设备未解锁时不会发生密钥访问。该开关值在 Android 9 及更高版本上可用。
- 使用 **setIsStrongBoxBacked()**，即可在更强大的独立芯片上运行加密操作。这会对性能带来轻微的影响，但更加安全。此功能在运行 Android 9 或更高版本的某些设备上可用。

注意: 如果您的应用需要在后台加密数据，则不应使用限时密钥或要求设备处于解锁状态，因为如果没有用户在场，您的操作将无法完成。

```
// Custom Advanced Master Key
val advancedSpec = KeyGenParameterSpec.Builder(
    "master_key",
    KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
).apply {
    setBlockModes(KeyProperties.BLOCK_MODE_GCM)
    setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
    setKeySize(256)
    setUserAuthenticationRequired(true)
    setUserAuthenticationValidityDurationSeconds(15) // must be larger than 0
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
        setUnlockedDeviceRequired(true)
        setIsStrongBoxBacked(true)
    }
}.build()

val advancedKeyAlias = MasterKeys.getOrCreate(advancedSpec)
```

解锁限时密钥

如果您的密钥是使用以下选项创建的，则必须使用 BiometricPrompt 对设备进行授权:

- **userAuthenticationRequired** 值为 true
- **userAuthenticationValiditySeconds** > 0

在用户进行验证后，将基于有效秒数字段中给出时长解锁密钥。AndroidKeystore 没有用于查询密钥设置的 API，因此您的应用必须自己记录这些设置。您应该在展示授权界面的 Activity 的 **onCreate()** 方法中构建 BiometricPrompt 实例，以引导用户进行授权操作。

用来解锁限时密钥的 BiometricPrompt 代码:

```
// Activity.onCreate

val promptInfo = PromptInfo.Builder()
    .setTitle("Unlock?")
    .setDescription("Would you like to unlock this key?")
    .setDeviceCredentialAllowed(true)
    .build()

val biometricPrompt = BiometricPrompt(
    this, // Activity
    ContextCompat.getMainExecutor(this),
    authenticationCallback
)
```



```

        result: AuthenticationResult
    ) {
        super.onAuthenticationSucceeded(result)
        // Unlocked -- do work here.
    }
    override fun onAuthenticationError(
        errorCode: Int, errString: CharSequence
    ) {
        super.onAuthenticationError(errorCode, errString)
        // Handle error.
    }
}

```

To use:

```
biometricPrompt.authenticate(promptInfo)
```

加密文件

Jetpack Security 包含一个 `EncryptedFile` 类，它解决了加密文件数据的问题。与 `File` 相似，`EncryptedFile` 提供一个 `FileInputStream` 对象用于读取，一个 `FileOutputStream` 对象用于写入。我们使用遵循 OAE2 定义的 Streaming AHEAD 对文件进行加密。数据被分为多个区块，并使用 AES256-GCM 进行加密，使得外界无法对其进行重组。

```

val secretFile = File(filesDir, "super_secret")
val encryptedFile = EncryptedFile.Builder(
    secretFile,
    applicationContext,
    advancedKeyAlias,
    FileEncryptionScheme.AES256_GCM_HKDF_4KB)
    .setKeysetAlias("file_key") // optional
    .setKeysetPrefName("secret_shared_prefs") // optional
    .build()

encryptedFile.openFileOutput().use { outputStream ->
    // Write data to your encrypted file
}

encryptedFile.openFileInput().use { inputStream ->
    // Read data from your encrypted file
}

```

加密 SharedPreferences

如果您的应用需要保存键值对 (例如 API 密钥)，JetSec 提供了 `EncryptedSharedPreferences` 类，该类使用的是您所熟知的 `SharedPreferences` 接口。

键和值均会被加密。键使用能提供确定性密文的 [AES256-SIV-CMAC](#) 进行加密；值则使用 AES256-GCM 进行加密，并绑定到加密的键。该方案允许对机要数据进行安全加密，同时仍然便于查询。

```

EncryptedSharedPreferences.create(
    "my_secret_prefs",
    advancedKeyAlias,
    applicationContext,
    PrefKeyEncryptionScheme.AES256_SIV,
    PrefValueEncryptionScheme.AES256_GCM
).edit {

```

FileLocker是我们准备的一个示例应用，您可以在 Android Security GitHub 示例页面上找到它。这个应用很好地展示了应该如何使用 Jetpack Security 进行文件加密。

祝大家加密愉快！

[点击这里](#)进一步了解安全处理数据最佳实践

android

发布于 05-13

[Android](#) [数据加密](#) [Android 开发](#)

推荐阅读



Android开发之简单登录界面

没想好昵称呵



【Android 修炼手册】常用技术篇 -- 聊聊 Android 的打包

ZYLAB

发表于Andro...

使用 repo 组织 Android 工程

关于 Repo 的文章，我之前写过一篇：Repo VS Submodule，不过只是讲了 Repo 和 Submodule 的区别，以及 Repo 使用过程中的一些注意事项，然而在具体的项目里的所谓的最佳实践却并没有涉及...

Ailurus

微信抢红包插件与Android辅助功能

逢年过节大家都少不了发微信红包，通过微信红包来表达祝福。同时，微信还有拼手气群红包。各种群好友群，亲戚群，工作群逢年过节常常会有红包可抢。抢红包的口诀是：“网速要好，手速要快”...

古城

还没有评论

写下你的评论...

