Gatekeeper

Gatekeeper 子系统会在可信执行环境 (TEE) 中执行设备解锁图案/密码身份验证。Gatekeeper 会使用由硬件支持的密钥通过 HMAC 注册和验证密码。此外，Gatekeeper 会限制连续失败的验证尝试次数，并且必须根据指定的超时和指定的连续失败尝试次数拒绝服务请求。

当用户验证其密码时，Gatekeeper 会使用 TEE 派生的共享密钥对身份验证认证签名，以发送至由硬件支持的 Keystore (/security/keystore/index.html)。也就是说，Gatekeeper 认证可让 Keystore 知道可以发布与身份验证绑定的密钥（例如，应用创建的密钥）供应用使用了。

架构

Gatekeeper 包括以下 3 个主要组件：

- **`gatekeeperd`**（Gatekeeper 守护进程）。一种 C++ Binder 服务，其中包含独立于平台的逻辑，并且与 **`GateKeeperService`** Java 接口相对应。
- **Gatekeeper 硬件抽象层 (HAL)**。**`hardware/libhardware/include/hardware/gatekeeper.h`** 中的 HAL 接口，是一个实现模块。
- **Gatekeeper (TEE)**。**`gatekeeperd`** 的 TEE 副本。基于 TEE 的 Gatekeeper 实现。

Gatekeeper 需要实现 Gatekeeper HAL (#hal_implementation)（具体来说就是实现 **`hardware/libhardware/include/hardware/gatekeeper.h`** 中的函数）和 TEE 特有的 Gatekeeper 组件 (#trusty_and_other_implementations)（部分基于 **`system/gatekeeper/include/gatekeeper/gatekeeper.h`** 标头文件，该文件中包含用于创建和/访问密钥以及用于计算签名的纯虚函数）。

`LockSettingsService` 会通过 Binder 发出一个请求，该请求会到达 Android 操作系统中的 **`gatekeeperd`** 守护进程。**`gatekeeperd`** 守护进程会发出一个请求，该请求会到达此守护进程在 TEE 中的副本 (Gatekeeper)。

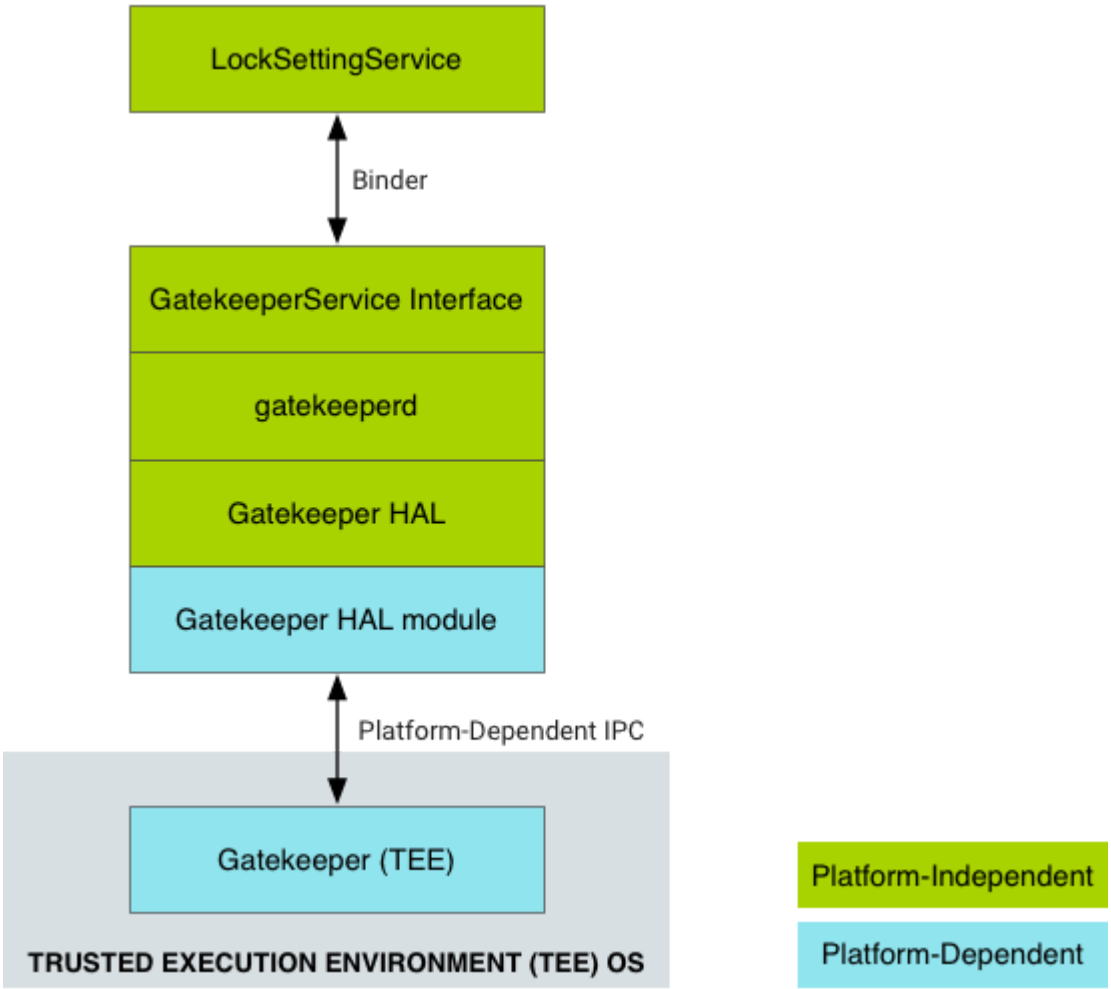

图 1. GateKeeper 进行身份验证的概要数据流程

gatekeeperd 守护进程会向 Android 框架 API 授予访问 HAL 的权限，并且会参与向 Keystore 报告设备身份验证 (/security/authentication/index.html)的活动。 **gatekeeperd** 守护进程会在自己的进程中运行，与系统服务器隔离开来。

HAL 实现

gatekeeperd 守护进程会利用 HAL 同 **gatekeeperd** 守护进程的 TEE 副本进行交互，以进行密码身份验证。HAL 实现必须能够签署（注册）和验证 Blob。所有实现都需要遵循每次密码验证成功时生成的身份验证令牌 (AuthToken) 的标准格式。要详细了解 AuthToken 的内容和语义，请参阅 AuthToken 格式 (/security/authentication/index.html#authtoken_format)。

要实现 **hardware/libhardware/include/hardware/gatekeeper.h** 标头文件，则必须实现 **enroll** 和 **verify** 函数。

- **enroll** 函数会获取一个密码 Blob，为其签名，并以句柄的形式返回签名。返回的 Blob（通过调用 **enroll**）必须有 **system/gatekeeper/include/gatekeeper/password_handle.h** 中显示的结构。

- `verify` 函数必须将通过收到的密码生成的签名与注册的密码句柄进行比较，并确认两者是否一致。

用于注册和验证的密钥不得更改，并且应该可以在每次设备启动时重新派生。

Trusty 和其他实现

Trusty (/security/trusty/index.html)操作系统是 Google 的开放源代码信任的操作系统，适用于 TEE 环境并且包含一个经过批准的 GateKeeper 实现。不过，只要 TEE 有权访问一个由硬件支持的密钥以及一个安全的单调时钟（**在暂停状态下运行**），您就可以使用**任何 TEE 操作系统**来实现 Gatekeeper。

Trusty 会使用内部 IPC 系统直接在 Keymaster 和 Trusty Gatekeeper（Gatekeeper 的 Trusty 实现）之间传达共享的密钥。这个共享的密钥用于签署已发送至 Keystore 的 AuthToken，以便提供密码验证认证。Trusty Gatekeeper 会在每次使用该密钥时向 Keymaster 请求该密钥，而不会保留或缓存该密钥的值。实现能够随意以不会降低安全性的任何方式共享该密钥。

HMAC 密钥用于注册和验证密码，是派生的密钥，单独保存在 GateKeeper 中。

Android 提供了一种通用的 C++ 版本的 GateKeeper 实现，只需添加设备专用例程即可完成。要使用设备专用代码为您的 TEE 实现 TEE Gatekeeper，请参阅 `system/gatekeeper/include/gatekeeper/gatekeeper.h` 中的函数和注释： 对于 TEE GateKeeper，合规实现的主要责任有：

- 遵循 Gatekeeper HAL。
- 返回的 AuthToken 的格式必须符合 AuthToken 规范（在身份验证 (/security/authentication/index.html)中进行了介绍）。
- TEE Gatekeeper 必须能够通过以下方法与 Keymaster 共享 HMAC 密钥：按需通过 TEE IPC 请求密钥，或始终维护密钥值的有效缓存。

用户安全 ID (SID)

用户 SID 是用户的 TEE 代码（与 Android 用户 ID 之间没有明显的关联）。每当用户注册新密码时，如果未提供之前的密码，系统就会使用加密伪随机数生成器 (PRNG) 生成一个用户 SID。这称为“不可信”重新注册，在正常情况下，Android 框架不允许进行这种操作。如果用户提供了之前的有效密码，便会发生“可信”重新注册；在这种情况下，用户 SID 会迁移到新密码句柄，从而保留绑定到它的密钥。

注册密码时，用户 SID 会随密码句柄中的密码一起接受 HMAC 处理。

用户 SID 会写入到 **verify** 函数返回的 AuthToken 中，并且会同所有与身份验证绑定的 Keystore 密钥相关联（要详细了解 AuthToken 格式和 Keystore，请参阅身份验证（/security/authentication/index.html））。由于对 **enroll** 函数的不可信调用会更改用户 SID，因此此类调用会使绑定到相应密码的密钥无法再使用。攻击者在控制 Android 操作系统后可以更改设备密码，但在此过程中，他们需要破坏掉受 Root 保护的敏感密钥。

请求次数限制

GateKeeper 必须能够安全地限制对用户凭据进行暴力破解的尝试次数。如 **hardware/libhardware/include/hardware/gatekeeper.h** 中所示，HAL 能够返回一个超时（以毫秒数计）。超时旨在通知客户端在超时过去之前不要再次调用 GateKeeper；如果有待处理的超时，GateKeeper 不应处理相关请求。

Gatekeeper 必须先编写一个失败计数器，然后再验证用户密码。如果密码验证成功，则应清除失败计数器。这可以在发出 **verify** 调用后防止攻击者发起以下攻击：通过停用嵌入式 MMC (eMMC) 来阻止请求次数限制。此外，**enroll** 函数还会验证用户密码（如果提供了），并且必须以同样的方式对其加以限制。

如果设备支持，强烈建议将失败计数器写入到安全存储空间。如果设备不支持文件级加密，或如果安全存储空间的速度过慢，实现可以直接使用 Replay Protected Memory Block (RPMB)。