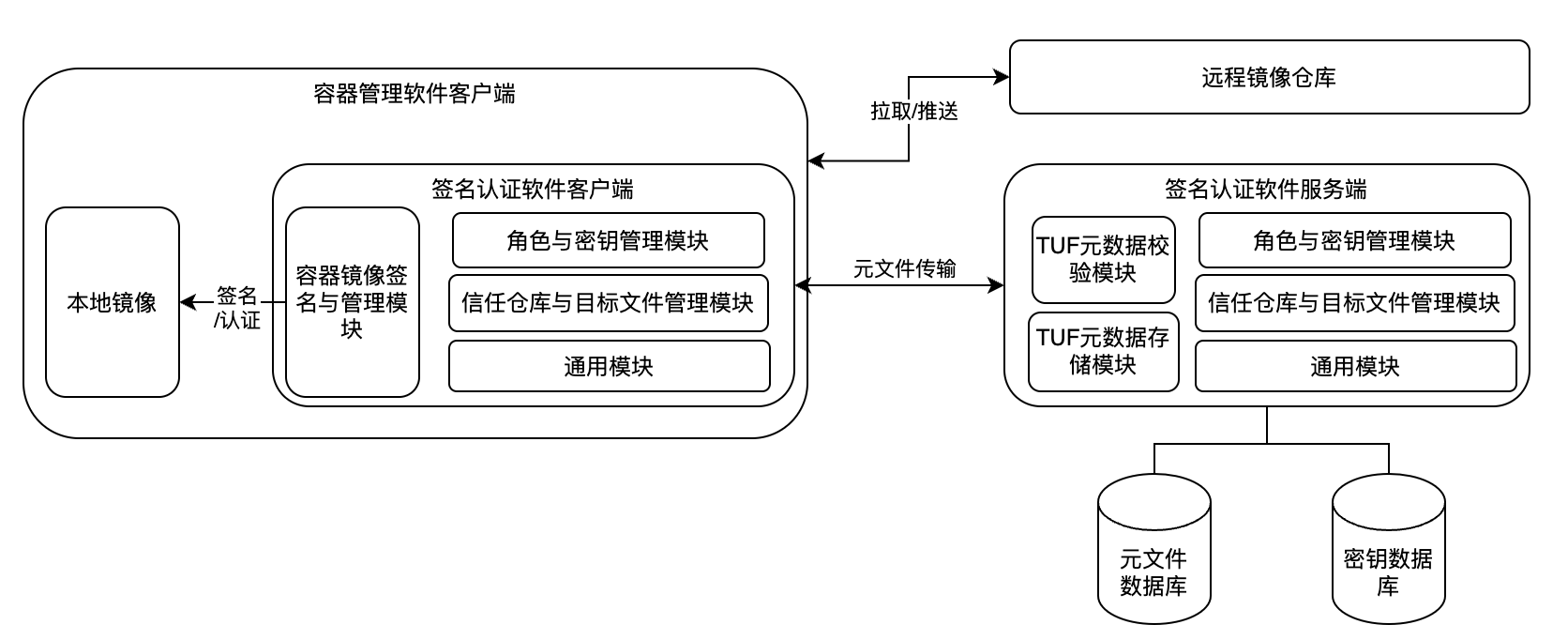
## 架构设计



签名认证软件的总体架构设计如图所示，整体分为客户端与服务端两大部分，包含命令交互工具模块、容器镜像签名与管理模块、TUF元数据校验与存储模块、信任仓库与目标文件管理模块、角色与密钥管理模块、通用模块等若干子系统，各模块共同协作实现镜像的可信签名与认证功能，保障镜像在发布与拉取过程中的安全性与完整性。

签名认证软件的目标用户包括镜像生产者（如镜像开发者，组织团队等）与镜像消费者（如使用镜像的用户）。用户可通过命令交互工具模块与签名系统交互，支持以命令行方式对系统进行操作控制。

容器镜像签名与管理模块是客户端核心功能模块，主要完成镜像签名、签名发布、镜像密钥吊销、自动化签名与验证、信任验证开启与信任信息管理等操作。该模块主要通过调用其他基础模块的能力来完成特定业务操作。

用户通过该模块可对本地构建的镜像进行可信签名，并将签名信息随镜像一同发布到镜像仓库。系统支持对镜像签名状态与信任策略进行统一管理，并支持自动化的签名验证逻辑，提高使用体验。

TUF元数据校验模块负责对签名元数据的格式、完整性、签名以及版本进行逐层验证，保障元信息的合法性与可信性。配套的TUF元数据存储模块用于对签名所需的元文件与密钥进行存储管理，为签名与验证操作提供元数据支撑。

信任仓库与目标文件管理模块实现信任仓库的初始化、同步、内容变更与签名内容列举等功能，是支撑镜像签名可信性的基础。该模块维护着系统的信任链结构与元数据信息，在镜像签名或验证过程中会与服务器侧保持同步。

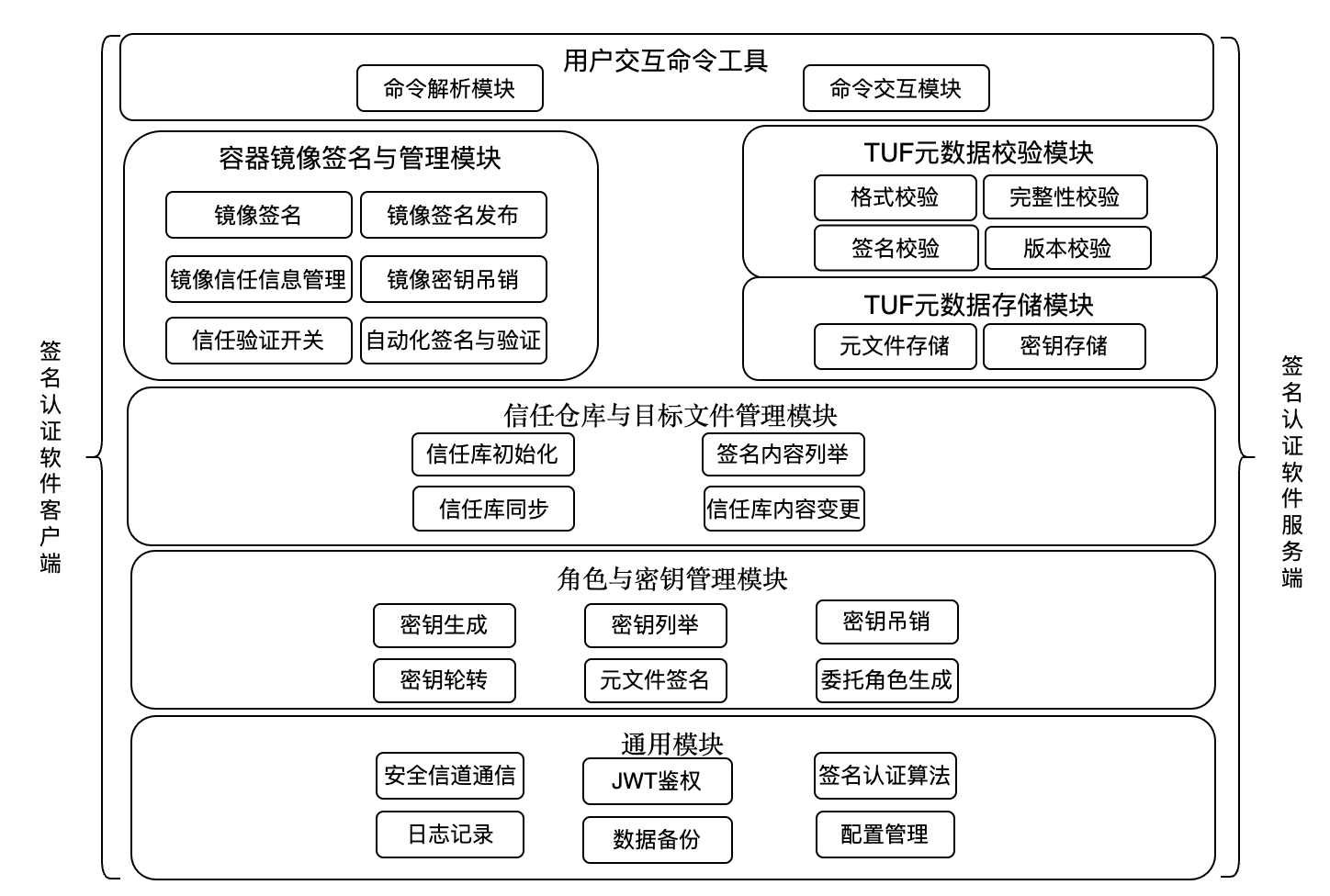
角色与密钥管理模块负责签名所需密钥的全生命周期管理，包括密钥生成、列举、轮转与吊销等基本操作，并支持元文件签名与委托角色的生成，从而实现基于TUF委托机制的精细化权限控制。

通用模块提供基础性的系统支撑功能，包括签名认证算法、JWT鉴权、配置管理、安全信道通信、日志记录与数据备份等，保障整个签名认证系统的安全性、可靠性与可维护性。其中日志记录功能支持用户操作记录的写入、查询与归档，便于安全审计。

系统在实现形态上，客户端工具作为独立可执行程序提供命令行交互接口，主要用于发起签名、验证与信任仓库管理等操作；服务端则以长期运行服务进程的形式提供元数据管理与存储能力。客户端在执行具体操作时，调用本地API与远程服务协同工作，共同完成镜像签名认证的全过程。

在镜像签名场景中，客户端会根据设定的信任策略调用角色与密钥管理模块完成签名操作，生成的签名元数据通过信任仓库模块进行存储和发布，最终用户在拉取镜像时由客户端自动完成签名验证，确保镜像来源可信、内容完整。通过上述各模块的协同配合，签名认证软件为容器镜像提供了完整的可信生命周期管理能力。

## 软件部件



客户端和服务端都包含用户交互工具，信任仓库与密钥管理模块，角色与密钥管理模块，通用模块。同时，客户端用容器镜像签名与管理模块对镜像进行签名验证，服务端通过TUF元数据校验和存储模块，对元数据进行校验和存储。

（1）用户交互命令工具模块：负责接收并识别用户输入的命令种类，然后将命令分发到对应的功能模块来执行。

（2）容器镜像签名与管理模块：该模块是镜像签名认证软件的核心组成部分，专注于容器镜像的签名、发布、信任管理和密钥吊销等关键功能，提供镜像签名认证服务。

（3）TUF元数据校验模块：该模块负责对从信任仓库下载的 TUF 元数据文件进行验证，以确保其有效性和可信度，防止各种安全攻击。

（4）TUF数据存储模块：负责管理镜像签名认证软件在本地存储的 TUF 相关元数据和密钥信息。

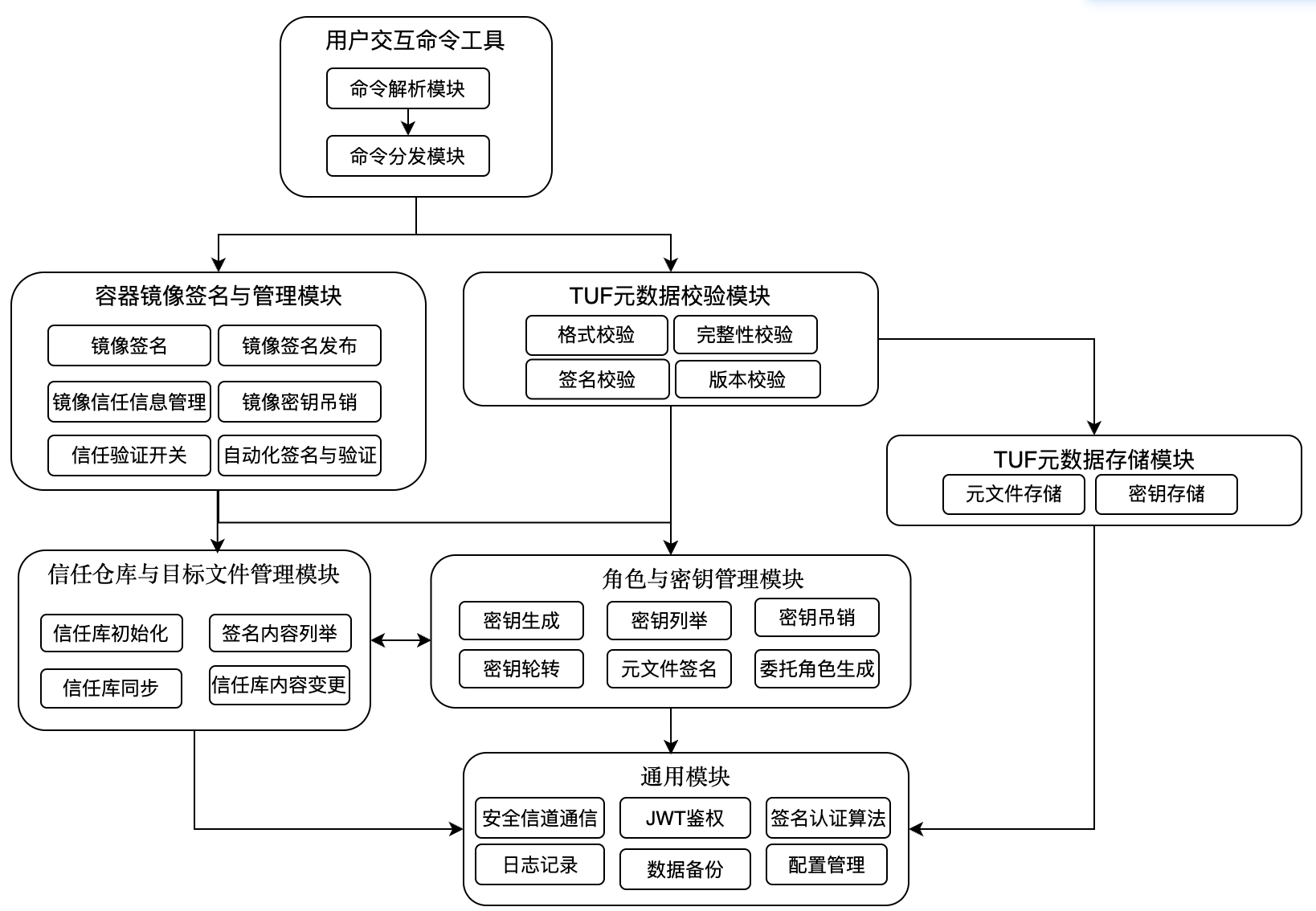
（5）信任仓库与目标文件管理模块：负责对信任仓库中的内容进行管理，并提供对目标文件（如容器镜像）对应的元数据文件的相关操作。

（6）角色与密钥管理模块：管理在签名认证过程中涉及的各种角色和密钥，包括密钥的生成、存储、轮换和吊销等。

（7）通用模块：该模块提供一些通用的、被其他模块所依赖的基础功能，以支持整个软件系统的运行和安全。主要包含安全信道通信，JWT鉴权，签名认证算法，日志记录，日志记录，配置管理等子模块。

|  |  |  |
| --- | --- | --- |
| **软件单元** | | **说明** |
| 用户交互命令工具 | 命令解析模块 | 对用户的输入命令进行解析匹配 |
| 命令分发模块 | 将命令分发到对应模块执行 |
| 容器镜像签名与管理模块 | 镜像签名 | 提供为容器镜像创建数字签名的功能. 使用目标密钥（Targets Key）对镜像内容摘要进行签名，以确保镜像的完整性和来源可信 |
| 镜像签名发布 | 将镜像的签名元数据发布到信任仓库，使其能够被客户端用于信任验证 |
| 镜像信任信息管理 | 提供查看镜像信任信息的功能. 用户可以查询目标文件的签名元数据的详细信息 |
| 镜像密钥吊销 | 支持撤销对特定镜像的信任，发起吊销请求并生成签名元数据更新，并向服务器端同步信任状态 |
| 信任验证开关 | 提供开启或关闭镜像信任验证功能的设定 |
| 自动化签名与验证 | 支持在推送镜像时自动进行签名，构建信任链，完成签名生成与上传. 同时，在拉取镜像时自动进行信任验证，校验信任链和签名，确保拉取的镜像可信 |
| TUF元数据校验模块 | 格式校验 | 验证 TUF 元数据文件的格式是否符合 TUF 规范 |
| 完整性校验 | 比对元数据文件中记录的哈希值（如 SHA256）与实际下载文件的哈希值，来验证元数据文件在传输过程中是否被篡改 |
| 签名校验 | 使用存储在 root.json 文件中的可信公钥，验证其他元数据文件的数字签名是否有效 |
| 版本校验 | 检查元数据文件的版本号是否符合更新要求，对于 root.json 的更新，还需要进行逐级版本检查 |
| TUF元数据存储模块 | 元文件存储 | 存储服务端的 TUF 元数据文件 |
| 密钥存储 | 安全地存储和管理用于签名和验证的密钥对，主要针对服务端的时间戳密钥和快照密钥 |
| 信任仓库与目标文件管理模块 | 信任库初始化 | 初始化本地信任库，并生成 TUF（The Update Framework）相关元数据文件 |
| 签名内容列举 | 列举信任库内容，查看信任仓库中存储的目标文件信息 |
| 信任库同步 | 与远程信任仓库同步元数据，以保持本地信任信息的最新状态 |
| 信任库内容变更 | 提供对信任库中目标文件的增、删、更、查功能 |
| 角色与密钥管理模块 | 密钥生成 | 生成新的公私钥对的功能，包括根密钥（Root Key）目标密钥（Targets Key），以及服务端使用的时间戳密钥和快照密钥 |
| 密钥列举 | 列举本地存储的所有密钥 |
| 密钥吊销 | 撤销对特定镜像的信任，发起吊销请求并生成签名元数据更新 |
| 密钥轮转 | 提供密钥轮换功能，支持策略变更、临期/逾期告警以及定期自动轮换等机制 |
| 元文件签名 | 对快照元数据生成并签名 |
| 委托角色生成 | 提供委托角色（密钥）添加、回收、列举功能 |
| 通用模块 | 安全信道通信 | 服务端与客户端交互安全通信通道功能 |
| JWT鉴权 | 进行客户端用户鉴权 |
| 签名认证算法 | 提供对多种算法支持和扩展性的功能，例如支持 ECDSA、RSA-PSS 算法 |
| 日志记录 | 提供日志记录的功能 |
| 数据备份 | 提供关键数据的备份恢复功能 |
| 配置管理 | 负责从配置文件或环境变量中读取和解析工作参数 |

各个软件单元调用关系如下：



当用户上传 (推送) 启用内容信任的容器镜像时，如果信任验证开关被启用，会进行镜像签名认证，客户端需要对信任（仓）库进行初始化，自动生成根密钥（Root Key）目标密钥（Target Key）。私钥需要密钥加密存储，然后使用目标密钥（Targets Key）对镜像标签进行签名。镜像发布者可以自行决定签署哪些标签，签名元数据与镜像的 TAG 部分相关联，最后发布已签名的元数据，包含根元文件，目标元文件。

服务端收到客户端发送的元数据后，对元数据进行校验，然后生成快照密钥和时间戳密钥，保存到密钥数据库存储，之后根据根元文件和目标元文件，生成快照元文件和时间戳元文件，用对应密钥签名后保存到元文件数据库。

当用户拉取启用内容信任的容器镜像时，如果信任验证开关被启用，会进行镜像签名认证，校验元文件的信任链，如果没有问题，则从镜像仓库拉取对应的目标文件，最后检查元文件中的哈希与目标文件的哈希是否一致。

## 技术方案

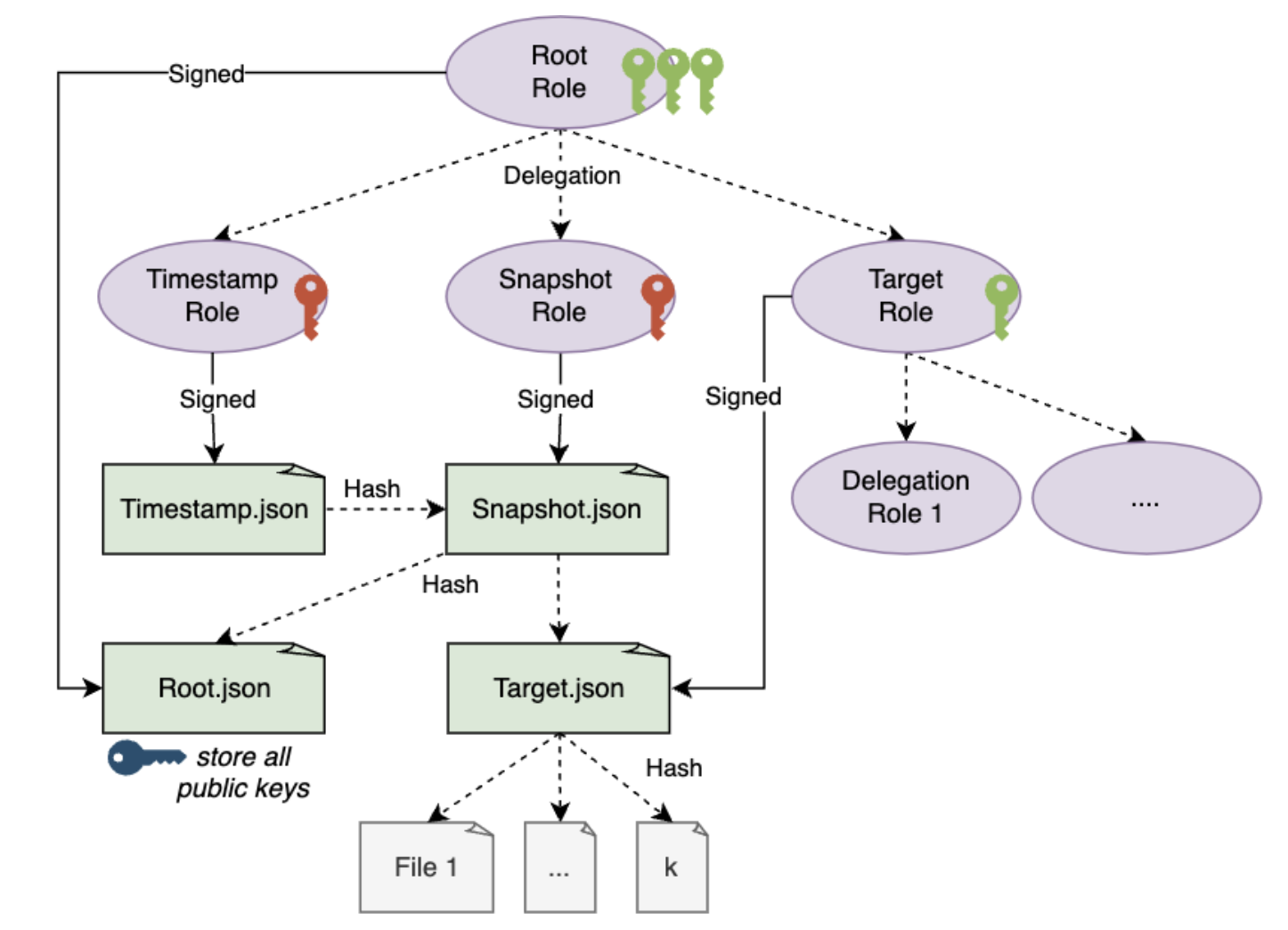
### TUF元数据校验模块

#### TUF角色与元文件定义

本系统中的TUF角色与元文件定义遵循[TUF规范v1.0.33](https://theupdateframework.github.io/specification/v1.0.33/)，各个角色职责如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **角色名称** | **职责概要** | **签名对象** | **密钥位置** | **由谁管理** |
| Root | 管理和分发所有其它角色的公钥。支持密钥轮换、撤销 | Root.json | 离线 | 项目负责人 |
| Targets | 指定哪些镜像（或文件）被信任，可发布。支持 Delegation | Targets.json | 离线 | 项目负责人 |
| Snapshot | 确保 targets（及其 delegation）元数据的一致性与版本管理，防止回滚和混合匹配攻击 | Target.json及其Delegation | 可存服务端 | 自动化构建系统或服务器 |
| Timestamp | 指出最新的 snapshot 版本，减轻冻结攻击 | Snapshot.json | 服务端 | 自动化构建系统或服务器 |

不同角色与元文件的关系定义如下：



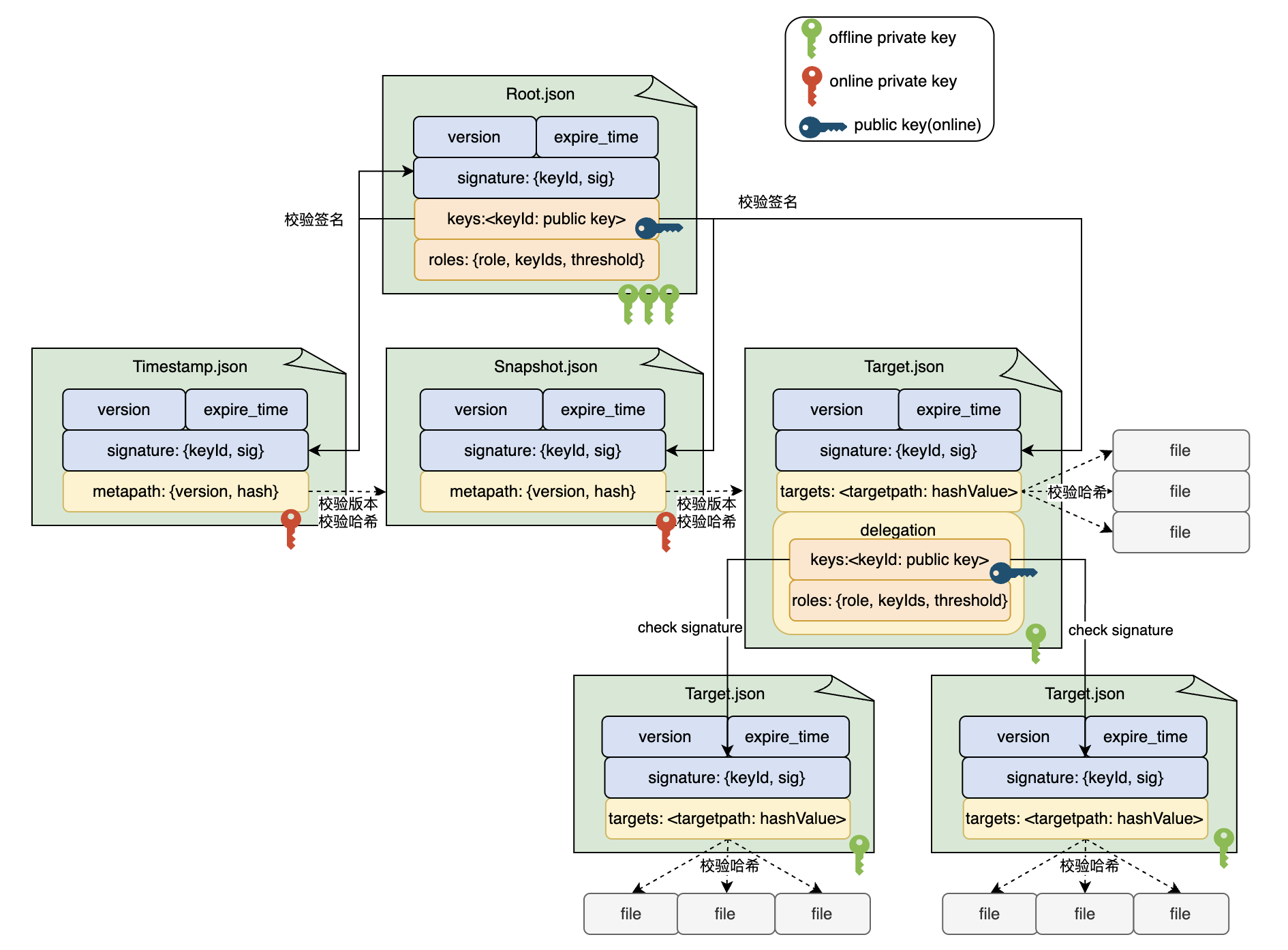
#### TUF元数据校验

1. 格式校验：TUF 规范允许实现者选择任何数据格式元数据文件，但要求所有规范中定义的字段都必须包含且能够被客户端无歧义地解析，推荐使用支持规范化（canonicalization）的数据格式，如 JSON 的子集。本模块采用的具体元数据格式 JSON进行格式校验，确保文件结构符合预期，并且包含所有必要的字段，如 \_type, spec\_version, version, expires, meta, signatures, signed 等。
2. 完整性校验：为了防止中间人攻击者的混合搭配攻击，需要对下载的元数据文件的完整性进行校验。

* Snapshot 元数据 文件中会记录 Targets 元数据 及其 delegation 的哈希值，在更新 Snapshot 元数据时，客户端需要校验其哈希值是否与 Timestamp 元数据中记录的哈希值一致。
* Timestamp 元数据 文件会记录 Snapshot 元数据 的哈希值。在更新 Targets 元数据时，需要校验其哈希值是否与 Snapshot 元数据中记录的哈希值一致。
* 这种哈希校验机制可以确保元数据文件在传输过程中没有被篡改。

1. 签名校验：TUF 的核心安全机制之一是使用数字签名来验证元数据文件的真实性。 每个元数据文件（Root, Timestamp, Snapshot, Targets）都由一个或多个与相应角色关联的密钥进行签名。 Root 元数据 文件中包含了所有其他角色的公钥。 在更新任何元数据文件时，客户端必须使用可信的 Root 元数据中记录的公钥来验证其签名，每个参与签名的密钥 ID (KEYID) 在计算阈值时只能贡献一个签名。 如果签名验证失败，客户端必须丢弃该元数据文件并中止更新
2. 版本校验：TUF 使用版本号来防止回滚攻击。 每个元数据文件都有一个版本号。 在更新元数据文件时，客户端必须检查新文件的版本号是否大于或等于旧文件的版本号

下图说明了校验流程使用了元文件的 哪些字段。



### 角色与密钥管理模块方案

#### 客户端密钥生成与管理

密钥生成模块接收角色类型与算法参数（如ECDSA，RSA等），生成不同角色（如 Root, Targets, Snapshot, Timestamp）的公私钥，符合 TUF 规范的密钥对。

对于客户端而言，主要生成的是根密钥和目标密钥，生成后的私钥通过加密方式本地存储，用户需通过交互或环境变量设置密钥解密密码。

对于服务端而言，主要生成的是时间戳密钥和快照密钥，生成后的私钥存储在密钥库中加密保存，用以后续服务器进行时间戳和快照元文件签名。

支持密钥列举接口，返回密钥仓库中所有已注册密钥的角色、KeyID、创建时间等元信息。

#### 元文件签名

该模块负责使用与每个角色关联的私钥对相应的元数据文件进行签名。

Targets 角色使用其私钥对 targets.json 进行签名，Snapshot 角色使用其私钥对 snapshot.json 进行签名，Timestamp 角色使用其私钥对 timestamp.json 进行签名，Root 角色使用其私钥对 root.json 进行签名。

对于客户端，需要负责Root和Target角色的签名。

对于服务端，需要负责Timestamp角色的签名。

Snapshot角色在哪里签名，取决于系统配置，可以在客户端生产Snapshot密钥并签名，也可以选择把Snapshot密钥托管到服务端，让Snapshot在服务端签名。

#### 元文件更新

元文件格式采用[TUF规范v1.0.33](https://theupdateframework.github.io/specification/v1.0.33/)中定义的格式，在不再赘述，不同角色的元文件更新流程如下。

**更新Root.json：**

Root.json更新需要逐级建立信任链，也就是从版本1，逐级更新到版本N，中间不能跳跃。

1. 令N表示当前可信根元数据文件的版本号。

2. 尝试下载Root.json的N+1版本，最多X字节

3. 检查是否存在恶意软件攻击：

1. 需要超过Threshold个版本N的Root.json签名验证通过（垂直验证）

2. 需要超过Threshold个版本N+1的Root.json签名验证通过（水平验证）

4. 检查是否存在回滚攻击：

1. 新Root.json版本号 = 旧Root.json版本号 + 1

5. 检查是否存在冻结攻击：中间Root.json过期没有关系

6. 持久化Root.json

7. 重复2～9步

8. 检查是否存在冻结攻击：最终的Root.json文件过期时间戳未到，且高于固定更新时间

9. 持久化最新Root.json

**更新Timestamp.json**

1. 下载Timestamp.json，最多X字节

2. 检查是否存在恶意软件攻击：用Root.json记录的公钥验证Timestamp.json签名，需要超过Threshold个签名验证通过

3. 检查是否存在回滚攻击：

1. 新Timestamp.json版本号 > 旧Timestamp.json版本号

2. 新Timestamp.json中记录的Snapshot.json版本号 >= 旧Timestamp.json中记录的Snapshot.json版本号

4. 检查是否存在冻结攻击：新TimeStamp.json文件过期时间戳未到，且高于固定更新时间

5. 持久化Timestamp.json

**更新Snapshot.json**

1. 下载Snapshot.json，最多X字节

2. 防止中间人攻击者的混合搭配攻击：校验新Snapshot.json的哈希值是否与Timestamp.json记录的哈希值一致

3. 检查是否存在恶意软件攻击：用Root.json记录的公钥验证Snapshot.json签名，需要超过Threshold个签名验证通过

4. 检查是否存在回滚攻击：

1. 新Snapshot.json版本号 = 最新的Timestamp.json中记录的Snapshot.json版本号

2. 新Snapshot.json中记录的Target.json版本号 >= 旧Snapshot.json中记录的Target.json版本号

5. 检查是否存在冻结攻击：新Snapshot.json文件过期时间戳未到，且高于固定更新时间

6. 持久化Snapshot.json

**更新Target.json**

1. 下载Target.json，最多X字节

2. 防止中间人攻击者的混合搭配攻击：校验新Target.json的哈希值是否与Snapshot.json记录的哈希值一致

3. 检查是否存在恶意软件攻击：用Root.json记录的公钥验证Target.json签名，需要超过Threshold个签名验证通过

4. 检查是否存在回滚攻击：

1. 新Target.json版本号 = 最新的Snapshot.json中记录的Target.json版本号

5. 检查是否存在冻结攻击：新Target.json文件过期时间戳未到，且高于固定更新时间

6. 持久化Target.json

#### 密钥吊销与轮换

当密钥泄露或不再使用时，需要将其吊销，密钥吊销后，客户端将不再信任使用被吊销密钥签名的任何元数据。

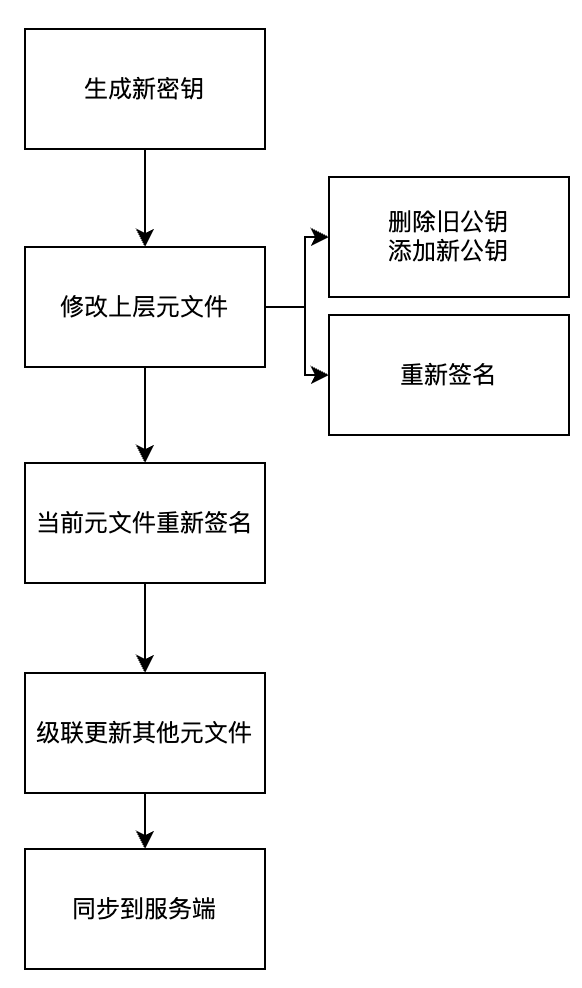
提供灵活的密钥轮换机制，支持按策略、周期、角色分类轮换密钥，确保长期安全性。

密钥轮换场景：

* 策略轮换：监听信任策略更新事件（如角色变更），触发相关角色密钥轮换；
* 告警轮换：当密钥临近过期或已过期时，接口返回轮换告警，客户端可触发轮换流程；
* 定期轮换：接口支持通过配置参数设定轮换周期，到期后自动轮换；
* 分层轮换：按角色进行分层轮换，允许针对Root更严格轮换策略；

吊销密钥的操作是通过更新上层角色（对于顶层角色来说是Root，对于委托角色来说是上层的委托角色）元数据文件来实现的，主要步骤如下。

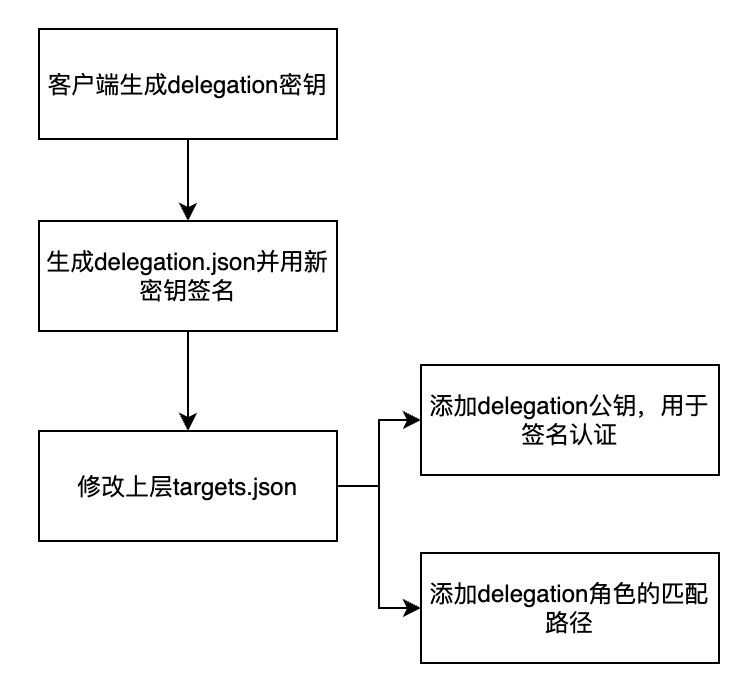
1. 使用密钥生成功能生成新密钥并保存
2. 将需要吊销或轮转的公钥从上层元数据文件中移除，添加新密钥的公钥
3. 使用上层角色的私钥重新签名发布新的上层元数据文件
4. 使用新密钥的私钥重新签名当前元数据文件
5. 级联更新其他上层元数据（比如上层targets.json更新后，对应的snapshot.json和target.json也需要更新）
6. 同步到服务端



#### 委托角色生成

TUF 支持 角色委托 (Role Delegation)，允许一个角色（通常是 Targets 角色）将对特定目标文件的信任委托给另一个角色。

主要方式是通过在 targets.json 文件中指定 delegation 信息和被委托角色的可信密钥。

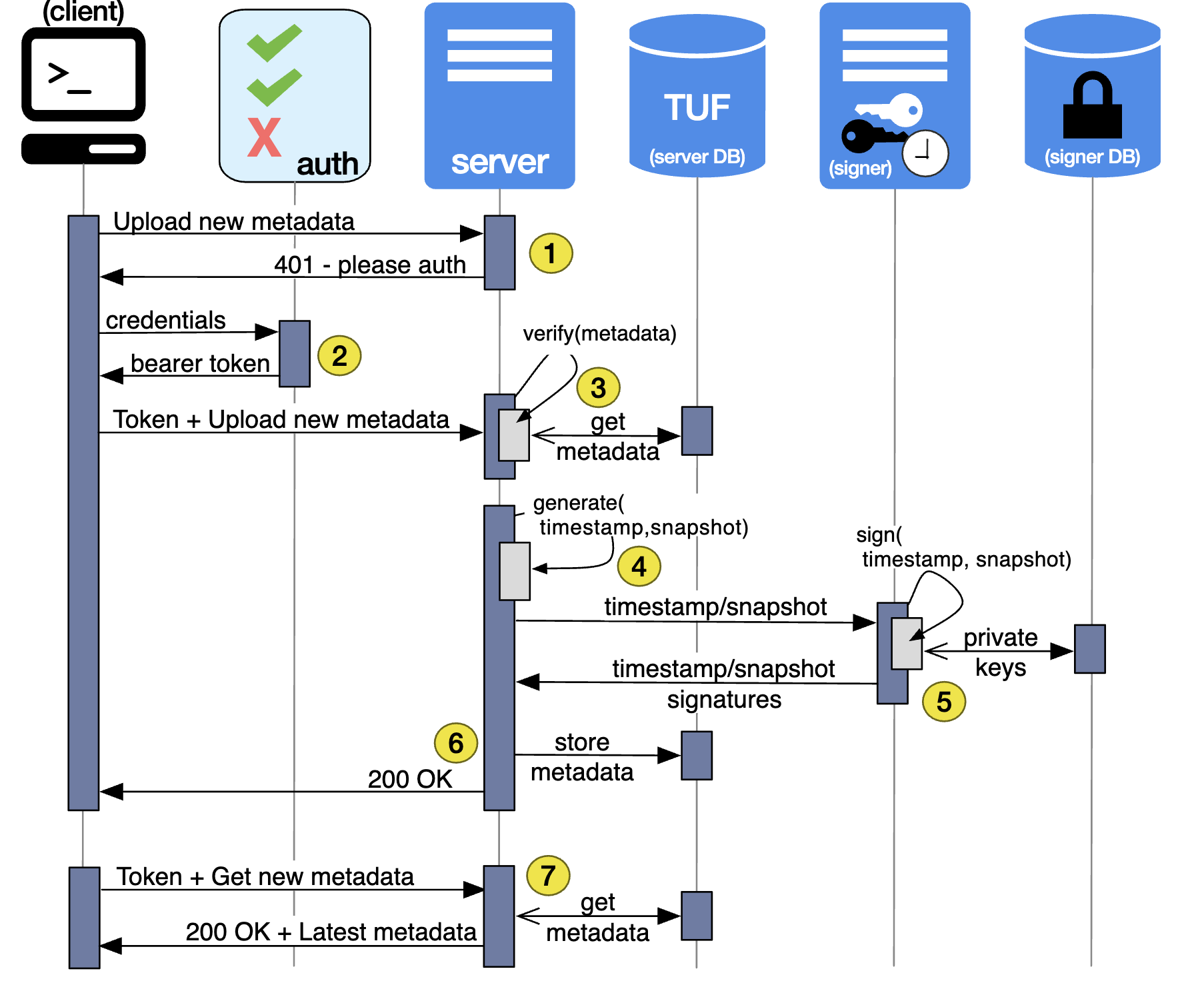


### 信任仓库与目标文件管理模块

#### 信任仓库初始化

信任库的初始化涉及到生成初始的 TUF 元数据文件和密钥，客户端生成Root，Target角色的密钥与元文件，服务端生成Snapshot，Timestamp角色的密钥与元文件，客户端与服务端进行同步，具体流程如下：

1. 客户端与服务端之间用JWT 令牌进行身份验证，确保客户端有权初始化信任库
2. 客户端使用密钥生成功能，生成Root和Target密钥，加密保存在本地存储中
3. 客户端使用Root和Target密钥，生成Root.json，Target.json，保存在本地文件缓存，并发送给服务端
4. 服务端使用密钥生成功能，生成Snapshot和Timestamp角色的密钥，加密保存在密钥数据库中
5. 服务端使用Snapshot和Timestamp密钥，结合收到的Root.json和Target.json，生成Snapshot.json，Timestamp.json
6. 服务端收到的Root.json，Target.json，Snapshot.json，Timestamp.json，都保存到元文件数据中。
7. 服务器不会向客户端主动推送元文件
8. 当客户端拉取元文件时，用本地的Root.json对收到元文件进行校验，校验无误后，保存最新状态的元文件到本地文件缓存

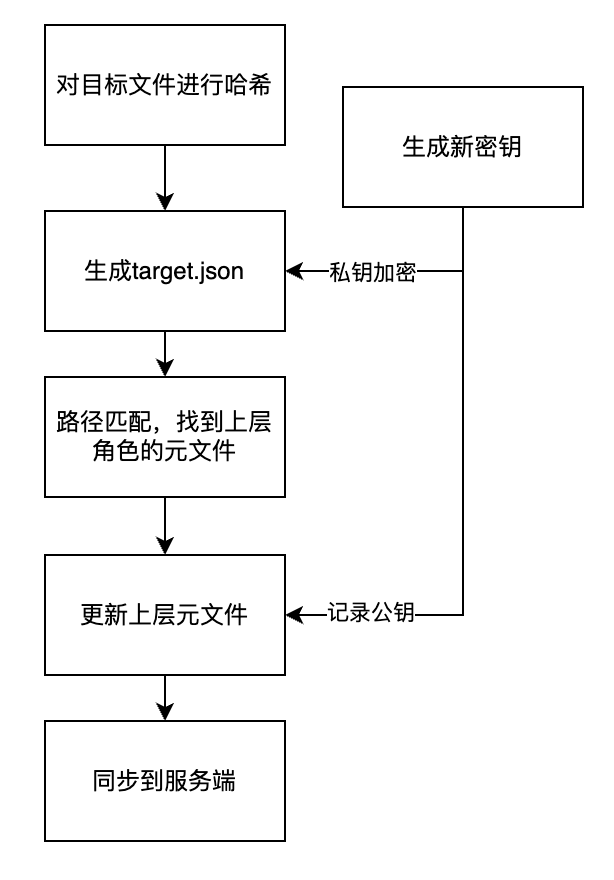


#### 信任仓库内容变更

支持对信任库中目标文件的增、删、更、查操作。客户端更新 targets.json 元数据文件，并重新签名上层对target.json，如果涉及到更新顶层target.json文件，那么还会更新root.json。服务器端接收客户端的这些变更请求，进行验证和处理，并更新snapshot.json和timestamp.json文件。

新增目标文件：

1. 使用密钥生成功能，生成新目标密钥
2. 对目标文件进行哈希，得到哈希值
3. 生成目标文件的target.json，记录目标文件哈希值，并用新密钥加密
4. 进行路径匹配，找到改目标文件所处路径的上层角色，在上层角色的json文件中添加新公钥，必要时需要更新root.json
5. 将更新后的元文件发送到服务端
6. 服务端根据新target.json，更新的snapshot.json和timestamp.json



修改目标文件：

1. 对目标文件机械能哈希，得到哈希值
2. 更改目标文件target.json中的哈希值，重新签名
3. 将更新后的元文件发送到服务端
4. 服务端根据新target.json，更新的snapshot.json和timestamp.json

删除目标文件：

1. 删除目标文件的target.json
2. 更新上层元文件，删除上层元文件中，该target.json对应的公钥
3. 将删除请求和更新后的元文件发送到服务端
4. 服务端删除目标文件的target.json，更新上层元文件，更新snapshot.json和timestamp.json

查找目标文件：

1. 拉取服务端最新的元文件数据
2. 进行路径匹配，找到目标文件对应的target.json

## 内部接口设计

### 角色与密钥管理模块接口

|  |  |  |
| --- | --- | --- |
| **功能** | **接口名称** | **接口功能** |
| 密钥管理 | KeyGenerate() | 生成新密钥，用以元文件签名，可指定密钥对应的角色 |
| KeyList() | 列出当前存储的密钥，包含信息：密钥id，角色，存储位置等 |
| KeyRemove() | 移除某个密钥，删除对应的密钥文件 |
| KeyPasswd() | 改变某个密钥的加密密码 |
| KeyRotate() | 轮转某个密钥 |
| KeyExport() | 导出密钥 |
| KeyImport() | 倒入密钥 |
| 角色管理 | DelegationAdd() | 添加角色委托 |
| DelegationList() | 列举所有委托角色，包含信息：角色名称，匹配的路径等 |
| DelegationRemove() | 移除角色委托 |

### 信任仓库与目标文件管理模块接口

|  |  |  |
| --- | --- | --- |
| **功能** | **接口名称** | **接口功能** |
| 信任仓库管理 | RepositoryInit() | 初始化信任仓库。如果本地没有密钥，自动生成根密钥，目标文件密钥。生成root.json，target.json并上传到服务器 |
| RepositoryDelete() | 移除某个信任仓库，删除该仓库的密钥和元文件 |
| 仓库同步管理 | RepositoryPublish() | 将本地仓库的修改同步到服务器中对应的远端仓库 |
| RepositoryStatus() | 查看当前本地仓库的修改 |
| RepositoryReset() | 撤销本地仓库的修改 |
| 目标文件管理 | RepositoryAdd() | 向某个信任仓库中添加目标文件 |
| RepositoryRemove() | 从某个信任仓库中移除某个目标文件 |
| RepositoryList() | 拉取远程仓库的最新状态，列出仓库中目标文件信息 |