YeeZ

交互式验证报告

范学鹏
https://yeez.tech

2024 年 11 月

目录

1 背景介绍

先验后买是典枢上的一个重要特性，该特性的实现通过在购买数据前查看“验证报告（Verification Report）”的方式实现。目前，验证报告有两种，一种是公开的，所有人都可以在数据详情页面查看，另一种是非公开的，只有申请的一方才可以看到（典枢平台也无权查看）。在实现上，这两种验证报告的区别在于公开的验证报告的申请者是典枢平台，而非公开的验证报告的申请者是用户。

目前，验证报告的实现逻辑如图1所示。Fidelius 节点执行验证算法后，将输出的验证报告数据或以明文发送给浏览器，或以密文发送给典枢客户端。

随着业务的发展，这种实现方式的局限性日渐突出，主要有两个：

- 无法动态扩展验证报告类型：目前前端（包括浏览器和客户端）提供的渲染组件，只支持 `text`,`image`, `bar` 三种类型，扩展新的类型则需要发布新的版本，这对于在网页上浏览的验证报告而言是可以接受的，但考虑到非公开验证报告是在典枢客户端上查看的，频繁更新典枢客户端是不可接受的；
- 验证报告的数据大小受限：由于验证报告的数据直接发送到了前端，因此，验证报告的大小不能太大，否则会有明显的性能瓶颈。

2 逻辑架构

为了解决前述问题，提出了*交互式验证报告*，架构设计如图 2所示，核心思想包括：

- 将用户与验证报告的交互作为单独的*动态加载*执行模块，以避免通过发版的方式支持新的交互类型；
- 验证报告数据不再完整的发送给前端，而是根据前端的交互逐步获取部分相应的验证报告数据。

为了方便描述，我们将验证报告数据统称为 VRData，VRData 由 Fidelius 在 TEE 环境中生成，且使用请求方的公钥加密，因此仅请求方可见。VRData 的类型使用验证程序的 Hash 标识，（下文记为 H），也就是说，我们认为一个验证程序的输出格式是确定的。

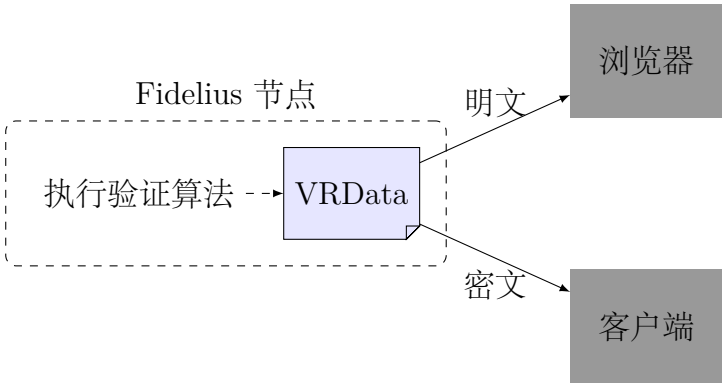

图 1: 目前对验证报告的处理：将验证报告的数据直接发送到客户端或浏览器。

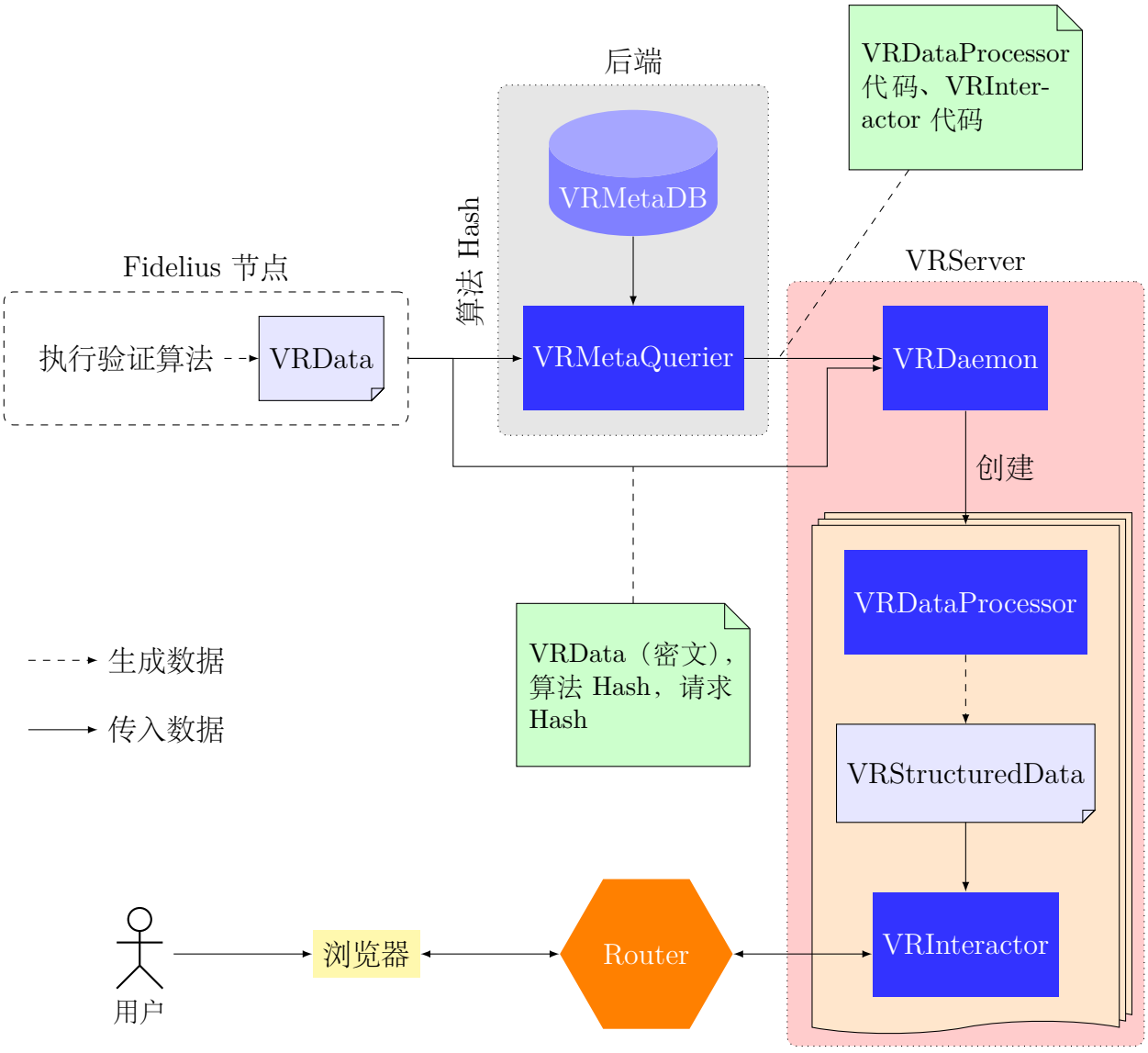

图 2: 交互式验证报告实现逻辑架构图

VRMetaDB 用于存储验证程序及相应的 VRData 的处理方法，包括

$$(H, D, S_{\text{VRDataProcessor}}, S_{\text{VRInteractor}}, \text{Sample})$$

其中 H 是验证程序的 Hash 标识，D 是验证程序的描述，$S_{\text{VRDataProcessor}}$ 是 VRDataProcessor 的可执行代码，$S_{\text{VRInteractor}}$ 是 VRInteractor 的可执行代码，Sample是该验证程序所生成的样例数据，该样例数据用于自动化测试，也用于在发布数据选择验证程序时进行展示。注意，当使用数据库时，其中不必存储代码或数据本身，也可以仅存储代码或数据文件的路径，以减少数据库大小。VRMetaDB 中的信息由验证程序和 VRInteractor、VRDataProcessor 的开发者共同维护，系统应提供相应的操作方式。

VRMetaQuerier 是后端的一个模块，接收 VRData 对应的验证程序 Hash（也就是 VRData 对应的 H），查询 VRMetaDB，输出 $S_{\text{VRDataProcessor}}, S_{\text{VRInteractor}}$。

VRServer 是提供验证报告交互的服务端，注意，VRServer 可能运行在客户端，也可能运行在后端。

VRDaemon 是 VRServer 的主模块，用于接收 $S_{\text{VRDataProcessor}}, S_{\text{VRInteractor}}$ 并启动相应的服务[1]、接收 VRData 并进行相应的处理。如果 VRDaemon 运行在后端，则意味着相应的验证报告是公开的，请求方是典枢平台，所以，VRDaemon 需要向后端请求相应的私钥，用于解密 VRData；如果 VRDaemon 运行在客户端，则意味着相应的验证报告是私有的，请求方是客户自己，因此，VRDaemon 需要在本地获取用户的私钥[2]，用于解密 VRData。

VRDataProcessor 用于处理验证报告数据，输入是 VRData（明文），输出为处理后的结构化数据（VRStructuredData）。一个 VRData 对应着一个 VRDataProcessor。VRDataProcessor 对于数据的处理是多样的，例如可以将一个图片生成多个不同分辨率的图片。VRDataProcessor 的输出可以存放在不同的存储中，可以是文件、数据库、CDN、也可以是内存。这些不同的存储称为 VRDataProcessor 的**上下文**。自然的，根据运行环境（客户端、后端）的不同，上下文的配置也是不同的，VRDataProcessor 应该能够适应不同配置的上下文。

VRInteractor 用于响应用户，是一个 Web 服务，其中包含了显示数据的页面，也包括从 VRStructuredData 中获取数据的请求。

VRDataProcessor 和 VRInteractor 是独立的。VRDataProcessor 仅在收到 VRData 后执行一次，VRInteractor 则常驻，以响应用户的操作。两者设计为独立的，这主要是考虑到 VRDataProcessor 对于数据的处理可能是耗时的（或者需要对数据进行预处理），不能即时响应，而 VRInteractor 则仅响应操作，不处理数据，提高用户

[1]可以是进程、也可以是线程、还可以是仅仅在当前线程注册相关服务

[2]一般来说需要用户输入授权密码

体验。

3 一个例子

本节给出一个实操性的例子，以说明架构中的流程和考量。

> 假设数据集包含了 10 个文件夹，每个文件夹下都包含了 1000 张高清图片（每个图片的大小为 10MB），一共包含了 10000 张图片。验证程序为每个图片生成一个压缩后图片（报告图）（大小为 400KB）。

从验证程序的描述和数据集的大小不难知道，验证报告的数据集大小大约为 10000 * 400KB = 4GB。这显然不能直接传输给浏览器或客户端。因此，目前已有的方案（图1所示）不能满足这一需求。

在本文所描述的方案中，相应的 VRDataProcessor 会为每个图片进一步生成大小仅为 4K 的缩略图，在后端场景下，将缩略图和报告图上传到 CDN，在客户端场景下，将缩略图和报告图存储在本地。VRInteractor 则展示一个运行在 web 环境下的文件浏览器。

我们先考虑该验证报告为公开的场景下，用户打开浏览器，其中显示了 VRInteractor 展示的文件浏览器，此时后端与浏览器之间不传输任何图片数据。用户打开某个文件夹之后，浏览器通过懒加载的方式，向 VRInteractor 请求部分图片的缩略图，此时的数据传输量很小。当用户查看某个图片的报告图时，仅向 VRInteractor 请求该图片的报告图，而不是所有图片的报告图，此时的数据传输量仅为该图片大小（400KB）。由此不难发现，虽然整个验证报告的大小高达 4GB，但是通过 VRDataProcessor 对属于的预处理，以及 VRInteractor 的仅响应请求的数据，使得浏览器与后端之间的传输量大大减少，提高了用户体验。

我们接下来考虑验证报告为非公开的场景，用户打开客户端。此时很可能发生的事情是客户端本地没有该验证程序对应的 VRDataProcessor 和 VRInteractor 的执行代码，因此，首先需要从 VRMetaQuerier 获取响应的执行代码，并通过 VRDaemo 启动。接下来的流程同上，此处不在赘述。可以看到，该架构很好的解决了客户端更新的问题。

4 实现考虑

本节论述可能的实现方式。首先，需要明确各个组件所运行的环境，VRMetaQuerier 的运行环境是后端，这是毫无争议的。VRServer 的组件则如表1所示。注意，客

	VRDaemon	VRDataProcessor	VRInteractor
后端	✓	✓	
浏览器			✓
客户端	✓	✓	✓

表 1: VRServer 中各个组件的运行环境

户端需要同时兼容 Windows, MacOS 以及 Linux。

综上考虑，应该在 VRDaemon 和 VRDataProcessor 应该使用同样的后端技术实现[3]，VRInteractor 则应该使用前端技术实现（如 Vue）。

接下来，考虑可用的后端技术。不同于仅部署在服务器侧，此处需要在不同的客户端中实现，因此需要满足可移植性，也就是在 Windows，MacOS，Linux 中可以运行。

考虑到团队的技术栈，不考虑 webassembly 技术，仅考虑 Node.js 和 Java。两者都符合技术要求，区别仅在于如果使用 Java，则需要将可执行代码发布为 jar 包，并且在典枢客户端中集成 JRE，以便用户安装使用，这意味着需要改变安装方式。而使用 Node.js 的话，则与目前典枢客户端的 electron 技术栈一致，可以无缝完成安装升级，此外，基于 Node.js 的相关加、解密相关工具（相比于 Java）更加成熟稳定，。使用 Node.js 的弊端在于开发资源分配不均匀，导致前端开发的压力更大。

综合考虑之下，VRServer 的实现应该基于 Node.js 技术实现，同时能够支持后端和客户端。

[3]应该仅有一份代码，而不是为两个版本分布维护两份代码