对多数据融合的支持

范学鹏

August 18, 2022

1 需求分析

在已有的系统中，一个算法的数据源只能是一个，然而，在某些场景下，一个算法需要多个数据源才能得到相应的结果，因此需要将算法的数据源扩展到多个。

2 多数据融合

一个算法的输入可以是 0 个或多个数据源，这些数据源被描述为一个数组

$$[D_1, \cdots, D_i, \cdots, D_n]$$

，其中每一个元素为一个三元组：

$$(H_i, P_i^S, T_i),$$

其中，H_i 为数据 i 对应的哈希，P_i^S 为加密数据 i 所使用的枢公钥，T_i 为数据源的表示，用于可读性或算法内使用。

注意，我们此处使用*数据源*，区别于输入参数和模型参数。

本文描述如何在 Fidelius 中生成多个数据源的正确性证明，以及如何在智能合约中验证相应的证明的正确性。多数据融合还包括计算框架（HPDA）部分如何支持多个数据源，不在本文的范围内。

多数据融合的前提是数据托管，多数据融合时，系统不支持其中某一个数据是原始数据（即未加密的），因此，所有数据都需要以加密的形式被计算节点访问。为了保证加密数据的安全性，每一个数据都是以托管的形式存储在计算节点，因此，访问时，需要获得相应的授权，相应的细节见《数据托管》。

3 相关概念

为方便下文描述，本节首先罗列相关名词、概念以及符号。

- 数据：在本文，数据是指一个或一组存储在计算机中的信息，可以是 CSV、Excel 文件等，数据库中的内容，也可以是音频、视频、图像等多媒体文件。有时候，我们也称一个包含了多组数据的集合为数据集，在本文数据、数据集并没有严格的区分。
- 原始数据：指数据本身。
- 加密数据：指使用非对称加密技术，对原始数据进行加密后的结果。
- 数据哈希：是指一个数据的唯一标识，在知道原始数据的情况下，可以通过哈希算法（例如，sha3-256），计算出相应的数据哈希。
- 算法：在本文，算法是指一个程序，用于对数据进行分析。一个算法的输入包括一个或多个加密数据，加密的参数，以及授权（本文不涉及授权相关的内容，因此省略）；一个算法的输出为*算法输出*，其中包括了加密的计算结果及其他内容。
- 算法哈希（hash）：在本文，是指一个算法经过编译器编译后生成的二进制文件进行哈希操作后得到的值，是一个算法的唯一性标识。
- 请求：在本文，一个请求是指使用指定算法、指定参数对指定数据进行计算的需求。

- 数据提供方：提供数据。数据提供方的核心需求是原始数据不能在未经许可的情况下泄漏。
- 数据使用方：数据使用方提出请求，并得到算法输出。数据使用方的核心需求是
  - 请求中的参数信息不能泄露;
  - 算法输出中的结果不能泄露;
  - 算法输出中的计算结果是正确的，此处的正确是指结果是基于指定的算法、指定的数据、以及指定的参数生成的。
- Fidelius 计算节点：安装、部署了 Fidelius 隐私计算环境及工具的 TEE 节点。
- `keymgr` 是一个 Fidelius 中的命令行工具，用于创建、删除基于椭圆曲线的非对称密钥，以及查看相应的公钥。在 `keymgr` 中，私钥生成过程是在 TEE 内，且生成的私钥使用设备相关但不可见的密钥（如 TPM 中存储的密钥）加密后存储在本地；相应的公钥则可以公开查看。
- 典钥：是一对基于椭圆曲线生成的公私钥（典公钥 P^D，典私钥 S^D）。使用 `keymgr` 在 TEE 内生成，且私钥使用设备相关但不可见的密钥加密后存储到本地，因此，典钥是设备相关的，可以看做是一个 Fidelius 计算节点的标识。
- `yterminus` 是一个 Fidelius 中的命令行工具，用于在非可信环境下完成密码相关的操作。
- 枢钥：是一对基于椭圆曲线生成得公私钥（枢公钥 P^S，枢私钥 S^S）。使用 `yterminus` 在普通环境下生成，也可以使用相关的 javascript 库在网页生成，私钥直接明文存储。因此，枢钥是设备无关的。
- 可信第三方：诚实的执行相应的协议的参与方，做为第三方，在不与其中任一方合谋的前提下，不能得到原始数据或计算结果。

- 智能合约：特指部署在区块链上的智能合约，用于完成必要的交互、验证操作。注意，Fidelius 本身不包括智能合约，此处的智能合约仅表示一个可信第三方，也可以使用中心化的服务器代替。根据上下文，本文亦会使用区块链，除非特别说明，同样指部署在区块链上的智能合约。类似的，上链指调用智能合约。

- 枢私钥转发：指将枢私钥通过典公钥转发给指定的算法，记为

  $$\digamma(S^S, P^D, H_{\text{algo}}),$$

  即将枢私钥 S^S 通过典公钥 P^D 转发给哈希为 H_{algo} 的算法。此时，在 H_{algo} 对应的算法内，调用 `request_private_key_for_public_key`(P^S) 则返回 (S^S, P^D)，而在 $H'(H' \neq H_{\text{algo}})$ 对应的算法中以同样的参数调用同样的函数则返回 (nil, nil)，并抛出异常。函数

  `request_private_key_for_public_key`()

  通过加密的信道从 `keymgr` 获取枢公钥对应的枢私钥，以及加密枢私钥所使用的典公钥。

其他涉及到的概念还包括非对称加密、签名等，本文不再赘述。

4 系统原理

假设数据提供方有 n 个，分别为 $p_1, \cdots, p_n$，数据使用方为 a，算力提供方为 c，数据提供方 i 持有枢钥，$P^S_{p_i}$，$S^S_{p_i}$，数据使用方持有枢钥，P^S_a，S^S_a，算力节点持有典钥 P^D_c, S^D_c。数据方 i 持有的数据为 M_i。

4.1 前提内容

本文仅描述涉及到的修改，因此并不包含整个流程的细节，仅对必要的部分进行必要的描述。

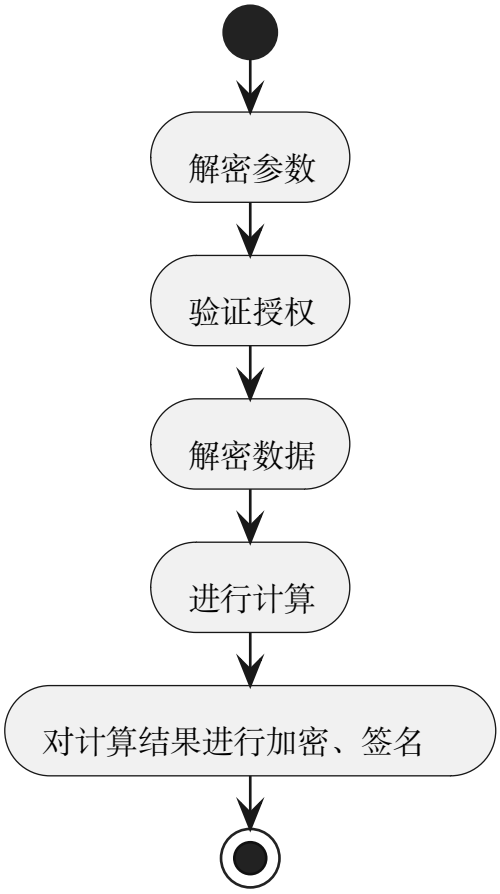

Figure 1: 在 TEE 中的计算过程

如图 1所示为 Fidelius 在 TEE 中的计算过程。其中最后一个步骤可以描述为一个函数

$$(r, \text{sig}) \leftarrow \text{GENERATE\_OUTPUT}(R, P_a^S, S_a^S, \iota, H_d, H_{\text{algo}}, \omega).$$

其中 R 为计算结果明文，P_a^S 为数据使用方枢公钥，S_a^S 为数据使用方枢私钥，ι 为加密后的参数，H_d 为数据哈希，H_{algo} 为算法哈希, ω 为消耗的计算资源。返回结果 (r, sig) 中，r 为加密的结果，sig 为对于结果的签名，用于证明计算结果的正确性。当结果为链下交付时，返回值还包括其他内容，由于不影响下文的描述，因此，此处省略。

相应的，智能合约中的验证逻辑为验证相应的签名，当签名验证通过时，则计算结果正确，否则计算结果错误。

4.2 多数据融合

如前所述，数据源描述为一个数组

$$D = [D_1, \cdots, D_i, \cdots, D_n]$$

数据 D_i 为数据提供方 p_i 持有，为

$$(H_i, P_{p_i}^S, T_i),$$

为方便描述，也记为

$$(H_i, P_i^S, T_i), \text{且} \forall i, j, 1 \leq i \leq n, 1 \leq j \leq n, i \neq j : H_i \neq H_j。$$

在 Fidelius 中，我们引入一个新的过程，如算法 1所示，将多个数据的哈希按照升序排序后，拼接为一个新的字符串，再次进行哈希操作后，得到一个单一的 “数据哈希”，问题转化为单一数据源的问题，从而可以复用原本的 GENERATE_OUTPUT。

在智能合约中，数据使用方在提交请求时，需要传入多个数据 hash 作为参数，智能合约按照同样的排序方法对多个数据 hash 进行排序、拼接、哈希后，得到单一数据哈希。

多数据融合同时支持结果的链上交付及链下交付。

4.3 安全性

算法 1并没有设计数据源实际的使用顺序，因此，可能存在数据源一样，但是使用顺序不同的情况。在此种情况下，输入参数应该包含如何使用数据源的信息，从而保证输入参数是不同的，因此，最终对于结果的前面也是不同的。

Algorithm 1 多数据融合时的算法输出

procedure GENERATE_OUTPUT_MULTI(R, P_a^S, S_a^S, ι, D, H_{algo}, ω) ▷ R 为计算结果明文，P_a^S 为数据使用方枢公钥，S_a^S 为数据使用方枢私钥，ι 为加密后的参数，D 为数据源，H_{algo} 为算法哈希, ω 为消耗的计算资源

$\delta \leftarrow \texttt{sort}(D)$ ▷ 按照 H_i 对 D 进行（升序）排序

$k \leftarrow \text{nil}$ ▷ k 为空字符串

for $i = 0$ to n **do**

$k \leftarrow k + \delta_i.H_i$ ▷ 此处为字符串拼接

end for

$H_D \leftarrow \text{Hash}(k)$

return GENERATE_OUTPUT(R, P_a^S, ι, H_D, H_{algo}, ω)

end procedure

5 Fidelius 中的实现

在 Fidelius 中，文件

`ypc/core/include/ypc/core_t/analyzer/`

`internal/data_streams/multi_data_stream.h`

中包含了对于多数据源的判定，但是多数据源的处理仍然是不完整的。

智能合约 本功能的实现需要引入智能合约，且对于 `onchain`，`offchain` 需要分别处理。

受限于智能合约的 Gas Limit, 一个算法的数据源个数有一定限制，通常认为不超过 256 个。

6 系统功能

支持本功能需要更新用户的交互流程，在用户选择数据源时，同时选择多个数据; 发布算法时，需要表明算法是否支持多个数据源。

7 关于链上结果及链下结果

本文涉及的方案同时适用于链上结果及链下结果。