TaskGraph: 在 Fidelius 中构建复杂数据服务

范学鹏

September 19, 2022

1 需求分析

在已有的系统中，一次数据服务（用户得到最终的模型或结论）仅包含一个计算任务，该计算任务读入原始数据，并产生最终的模型或结论。然而，在复杂的场景下，直接从原始数据产生最终的模型或结论是复杂的。例如，在机器学习中，需要对原始数据进行多次的清洗; 在卫星遥感数据中，需要将原始数据中的信息进一步处理为标准化范围的值，例如夜间灯光数据中对云层的处理、对闪电的处理。可以认为，从一个或多个数据中获得模型或结论，需要多个不同领域、不同经验的专家对数据进行多次处理，而将不同的处理方法集成到一个计算任务中，是不现实的。为此，我们提出 TaskGraph，解决大规模数据合作中不同的数据处理阶段需要多个不同领域、不同经验的专家相互协作的问题。

2 任务图（Task Graph）

在计算机科学中，任务图（Task Graph）的定义如下：

> The task graph represents the application: Nodes denote computational tasks, and edges model precedence constraints between

tasks. [1]

在本文，我们对任务图的定义类似。在 Fidelius 中，一个数据服务可以描述为一个任务图，任务图中的一个结点指一个计算任务，即一个 Fidelius 计算节点中、通过算法 enclave 完成的一次计算，任务图的边表示数据服务的中间数据，出边表示一个计算任务的输出，入边表示一个计算任务的输入。一个数据服务的输入为原始数据，输出为模型或结论。

一个数据服务可以有多个输入数据，这些输入数据可能来自不同的源或数据提供方，每个数据使用不同的枢公钥加密，且每个数据提供方对于数据使用有单独的定价。因此，我们将一个数据 D_i 描述为一个三元组

$$D_i = (d_i, P_i^S, C_i),$$

其中 d_i 为使用 P_i^S 加密后的数据，C_i 为数据的价格。注意中间数据也可以使用同样的三元组表示。

我们将一个数据服务对应的任务图 G 描述为一个集合，集合中的每一个元素表示一个输入 I，计算任务 A，输出 O 的三元组：

$$G = \{\cdots, (I, A, O), \cdots\},$$

输入 I 既可以是数据任务的输入数据，也可以是中间数据; 输出 O 既可以是中间数据，也可以是数据服务的输出。数据服务的输出只有一个，记为 o。

计算任务 A 包括相应的算法 enclave, E, 也包括运行的环境，由于运行的环境通常由典公钥 P^D 标识，因此我们将计算任务表示为一个三元组

$$A = (E, P^D, C),$$

其中 C 为价格，我们忽略了计算任务的其他内容，如输入参数、模型等。

图 2 表示了一个任务图的示例。该图也表示为

$$\begin{aligned} G = \{&(D_1, A_1, I_4^1), (D_2, A_2, I_4^2), (D_3, A_3, I_5^3), \\ &(I_4^1, A_4, I_5^4), (I_t^3, A_5, I_6^5), (I_6^5, A_6, o)\} \end{aligned} \tag{1}$$

[1] Encyclopedia of Parallel Computing pp 2013 – 2025, Task Graph Scheduling

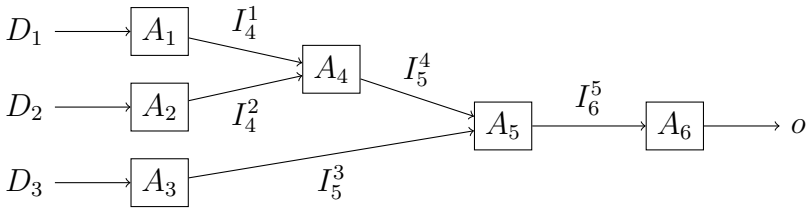

Figure 1: 任务图示例。其中，D_1, D_2, D_3 为任务图的输入，o 为任务图的输出，$A_1, A_2, A_3, A_4, A_5, A_6$ 为计算任务。

3 场景及约束条件

虽然一个数据服务可以抽象为一个任务图，但仍然有必要描述该任务图中的各个参与方，以明确任务图中对于隐私的需求。

- 数据提供方：提供任务图中的原始数据，以自己持有的枢公钥加密原始数据;
- 算法提供方：开发者，算法提供方也可以提供相应的模型;
- 数据使用方：得到最终的任务图输出;
- 算力提供方：安装 Fidelius 隐私计算节点，持有典公钥;
- 中间数据发起方：根据平台上的算法、数据，输入参数后，生成新的中间数据，并发布新的中间数据。

数据提供方及算法提供方根据自身的情况，不断将新的数据及算法发布到平台上。中间数据发起方则根据自己的经验或需要，组合已有的数据和算法，加上自己认为合理的参数，请求算力提供方完成中间数据的计算，完成后，将中间数据发布。因此，在任一时刻，平台上的中间数据会形成一个有向无环图（等价于没有数据任务输出的任务图）。同样的，在任一时刻，数据使用方选择某一个中间数据，获取自己需要的模型或结论。

举个例子，假设在平台上已经存在了某个原始数据集 D_1，且存在一个筛选算法 A_1，能够将 D_1 中的数据根据输入对某一列的值进行筛选，但是该

算法的开发者并不知道什么样的输入值是合理的。此时某个中间数据发起方（根据自己的经验）认为在算法 A_1 中输入确定的值对数据 D_1 处理后能够更方便后续的处理，因此认为应该生成一个中间数据，D_2。于是，该中间数据发起方发起了一个计算任务，生成了中间数据，并将该中间数据上链、发布。需要注意的是，直到中间数据 D_2 发布，该中间数据发起方可能都不知道这个中间数据会被如何使用。在中间数据 D_2 发布后的某个时刻，数据使用方可以基于 D_2，直接获取相应的模型或结论。

注意，在整个过程中，数据使用方是选择一个已有的中间数据获取计算结果，因此，对于数据使用方而言，一次任务涉及的计算任务依然只有一个。数据使用方也可以通过多次操作（可以提供更简便的操作方式）的方式，编排整个任务图，以完成更复杂的数据服务。

3.1 约束条件

首先，数据提供方对于原始数据的隐私保护有很强的要求，因此任务图的输入数据是使用各自的枢公钥加密的。同样的，由于中间数据很大程度上也能泄漏隐私，因此中间数据同样是需要隐私保护的，且中间数据的隐私要求等价于原始数据。

其次，中间数据是原始数据或中间数据经过算法、输入参数加工后生成的，因此，数据提供方（直接或间接的）、算法提供方以及中间数据发起方共同享有中间数据的主权，因此，任何一方都不能独自查看（解密）中间数据 [2]。

最后，算法包含了开发者的经验和知识（尤其是当算法中包含了训练出的模型时），因此算法的开发者需要整个计算过程保证算法（包括模型）不被泄漏。

[2]仅当多方共同合谋时，才能查看中间数据

4 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

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

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。

5 原理及算法

首先，用于产生中间数据的算法 enclave, 不同于其他算法 enclave, 是在算法 enclave 开发时就确定的，我们称这种用于产生中间数据的算法为转换器或 *transformer*。根据转换器的输入数据的不同，我们将转换器分为两类，输入转换器和中间转换器。以图 2 为例，A_1, A_2, A_3 为输入转换器，A_4, A_5 为输出转换器，A_6 则不是转换器。

5.1 输入转换器

我们首先讨论输入转换器，输入转换器用于将数据服务的输入数据（原始数据）转换为中间数据。和一般的算法 enclave 类似，输入转换器对原始数据进行计算处理后，得到结果。不同的是，输入转换器对结果的处理方法。为了下文讨论方便，我们假设输入转换器的输入数据为

$$D_i = (d_i, P_i^S, C_i),$$

P_i^S 对应的枢私钥为 S_i^S，为数据提供方持有; 算法提供方（开发者）d 的枢钥对为 P_d^S, S_d^S; 中间数据发起方 m 的枢钥对为 P_m^S, S_m^S[3]。

Algorithm 1 输入转换器的结果处理

procedure GENERATE_OUTPUT(R, P_i^S, P_d^S, P_m^S, ι, H_d, H_{algo}, ω) ▷ R 为计算结果明文，P_i^S 为数据提供方方枢公钥，P_d^S 为算法开发者的枢公钥, P_m^S 为中间数据发起方的枢公钥，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

$S_i^S, \text{dian\_pkey} \leftarrow \texttt{request\_private\_key\_for\_public\_key}(P_i^S)$ ▷ 通过枢私钥转发获得枢私钥

$S_d^S, \text{dian\_pkey} \leftarrow \texttt{request\_private\_key\_for\_public\_key}(P_d^S)$ ▷ 通过枢私钥转发获得枢私钥

$S_m^S, \text{dian\_pkey} \leftarrow \texttt{request\_private\_key\_for\_public\_key}(P_m^S)$ ▷ 通过枢私钥转发获得枢私钥

$S_t^S \leftarrow S_i^S + S_d^S + S_m^S$ ▷ 此处 + 为定义域?群运算

$P_t^S \leftarrow \texttt{generate\_public\_key}(S_t^S)$

$r \leftarrow \text{Enc}_{P_t^S}(R)$

$H_R \leftarrow \text{Hash}(R)$

$\text{sig} \leftarrow \text{Sign}_{S_t^S}(\iota + H_d + H_{\text{algo}} + H_R + \omega)$ ▷ 此处 + 为字符串拼接

return r, H_R, P_t^S, ω, sig

end procedure

算法 1 描述了对于输入转换器的结果的处理, 处理之后的 r 即为中间数据。注意到 S_t^S 是由多个枢私钥生成的，我们称之为中间数据的枢钥。P_t^S 也可以通过定义域上群运算直接使用相应的公钥生成，即

$$P_t^S \leftarrow P_i^S \times P_d^S \times P_m^S,$$

此处，$\times$ 为定义域上的群运算。

[3] P_m^S 也用来加密输入参数

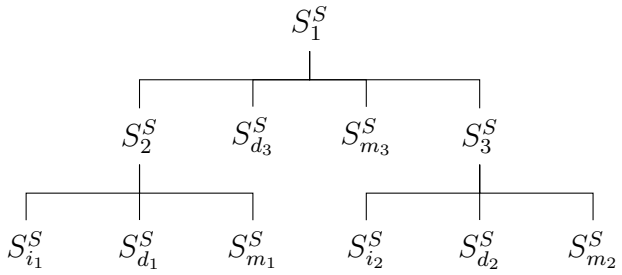

Figure 2: 中间数据的枢私钥生成树。通过在私钥的定义域上的加法群运算，将多个枢私钥 “求和”，生成一个新的枢私钥。相应的枢公钥的计算可以按照同样的过程在公钥的定义域上的加法群运算完成。

中间数据仍然需要发布上链，类似于原始数据的上链过程，只是在签名验证时，是通过多方的枢公钥生成相应的公钥并进行签名的验证。此处不再赘述。

5.2 中间转换器

不同于输入转换器，中间转换器的输入数据为中间数据，因此，我们有必要首先讨论中间数据的枢钥。

中间数据的枢私钥可以描述为一个树（枢私钥的生成树），叶子节点为各个枢私钥，根节点为所有叶子节点的 “和”，即中间数据的枢私钥。如图 2 所示为一个典型的中间数据的枢私钥的生成树。对于中间数据 τ 的枢私钥 S_τ^S，我们称其生成树为 $T(S_\tau^S)$，对应的枢公钥 P_τ^S 的生成树为 $T(P_\tau^S)$。不难发现，$T(S_\tau^S)$ 和 $T(P_\tau^S)$ 是同构的。通常使用 `JSON` 描述中间数据的枢钥生成树，且仅包含其中的枢公钥。需要生成中间数据的枢私钥时，通过枢私钥转发获取叶子节点的枢私钥，然后按照等价的树结构，生成最终的枢私钥。

通过枢私钥生成树获取中间数据的枢私钥，中间转换器才能解密中间数据。

算法 2 描述了对于中间转换器的结果处理。算法仅描述了只有一个中间数据的情况，是容易扩展到多个中间数据的。

Algorithm 2 中间转换器的结果处理

procedure GENERATE_OUTPUT(R, $T(P_i^S)$, P_d^S, P_m^S, ι, H_d, H_{algo}, ω) ▷ R 为计算结果明文，$T(P_i^S)$ 为中间数据的枢公钥生成树，P_d^S 为算法开发者的枢公钥，P_m^S 为中间数据发起方的枢公钥，ι 为加密后的参数，H_d 为输入中间数据哈希，H_{algo} 为算法哈希，ω 为消耗的计算资源

$T(S_i^S) \leftarrow$ generate_shu_private_tree($T(P_i^S)$) ▷ 通过枢私钥转发构造枢私钥生成树

S_d^S, dian_pkey $\leftarrow$ `request_private_key_for_public_key`(P_d^S) ▷ 通过枢私钥转发获得枢私钥

S_m^S, dian_pkey $\leftarrow$ `request_private_key_for_public_key`(P_m^S) ▷ 通过枢私钥转发获得枢私钥

$S_t^S \leftarrow \text{sum}(T(S_i^S)) + S_d^S + S_m^S$ ▷ 此处 + 为定义域?群运算

$P_t^S \leftarrow$ `generate_public_key`(S_t^S)

$r \leftarrow \text{Enc}_{P_t^S}(R)$

$H_R \leftarrow \text{Hash}(R)$

$\text{sig} \leftarrow \text{Sign}_{S_t^S}(\iota + H_d + H_{\text{algo}} + H_R + \omega)$ ▷ 此处 + 为字符串拼接

return r, H_R, P_t^S, ω, sig

end procedure

5.3 其他

- 对于基于中间数据产生最终结果的算法 enclave（如图 2 中的 A_6），虽然其定义并不是转换器，不需要按照算法 1 或算法 2 编写，但是，由于其输入数据同样为中间数据，需要使用中间数据的枢私钥生成树才能解密中间数据。因此，仍然需要构建枢私钥生成树。

- 为了构造枢私钥生成树，需要每个叶子节点的枢私钥做枢私钥转发，这会带来一定的交互成本。由于枢私钥转发需要执行环境的典公钥，因此，这个步骤并不能提前进行。

6 费用

本节我们讨论中间数据的定价问题。

考虑到算法、原始数据都已经在平台（区块链）存在的情况下，数据使用方可以自行编排一整个计算任务，也可以使用已经生成的中间数据，此时，中间数据的定价就成为影响数据使用方决策的重要因素。简单来说，直接使用中间数据可以节省计算费用。

中间数据发起方在获取中间数据的时候，仅需要支付计算费用，而不需要支付算法或输入数据的价格。生成的中间数据的价格包括输入数据的价格、算法价格以及中间数据发起方给定的额外价格。数据使用方在使用一个中间数据的时候，支付的费用包括了所有用于产生这个中间数据的算法费用、所有数据源的费用、中间数据的额外价格以及算力费用 [4]。因此，中间数据发起方给定的额外价格应该低于产生中间数据的计算费用 [5]。

[4]此处略去了平台的手续费

[5]此处把计算时间等因素也折算为了计算费用

7 实现

在智能合约中，实现相应的群运算是需要修改区块链本身的，这可以通过预编译合约实现。

在展现中间数据的时候，可以同时展示生成该中间数据的过程（以有向无环图的形式展示）。