链下数据交付的支持

范学鹏

September 8, 2022

1 需求分析

在已有的系统中，Fidelius 计算节点完成计算任务并对计算结果进行加密后，由可信第三方对计算结果进行合法检查后，将检查通过的加密数据转交给数据使用方。在某些系统实现中，该可信第三方通过部署在区块链上的智能合约实现，因此，这一可信第三方就有了一个额外的限制：所能接收的数据大小限制。这一限制一方面是因为区块大小的限制，另一方面是 GAS 的限制。虽然这一限制是动态的，但我们通常认为这一限制是 10KB，也就是说，可信第三方无法验证超过这一限制的结果。本文描述了为了解决这一限制，Fidelius 的改变以及相应的智能合约的实现。

注意,在不使用智能合约或区块链作为可信第三方时,无需引入该技术。

2 链下数据交付

区别于链上数据交付，链下数据交付是指借助区块链之外的数据传输能力完成数据传输，同时借助区块链保证传输的数据是正确且完整的。在本文的场景中，数据特指计算结果，因此也称链下结果交付，正确指计算结果是基于指定的算法、数据以及参数生成的。

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

假设数据提供方为 p, 数据使用方为 a，算力提供方为 c，数据提供方持有枢钥，P_p^S，S_p^S，数据使用方持有枢钥，P_a^S，S_a^S，算力节点持有典钥 P_c^D，S_c^D。数据方持有的数据为 M。

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。

如图 1所示为 Fidelius 的简要流程。本文重点描述由于计算结果无法上链的问题，即步骤 7、8、9。

4.2 算法输出

我们首先重新定义算法输出，注意，本部分描述的过程均在 TEE 内执行。算法 1描述了一种新的加密计算结果的方式：使用了在 TEE 内临时生成的一对密钥进行加密操作，将相应的私钥作为上链、验证的数据。

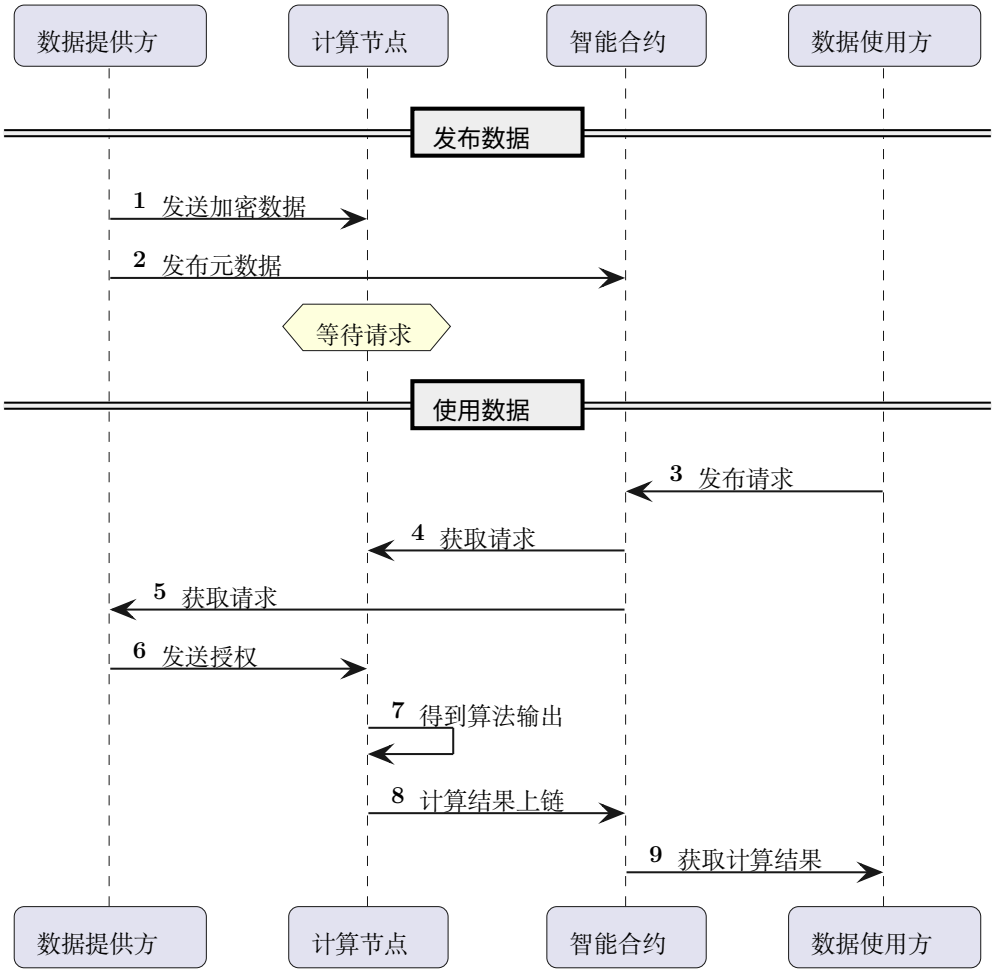

Figure 1: Fidelius 的简要流程

Algorithm 1 链下数据交付时的算法输出

procedure GENERATE_OUTPUT(R, P_a^S, S_a^S, ι, H_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，H_d 为数据哈希，H_{algo} 为算法哈希, ω 为消耗的计算资源

$S_t^S \leftarrow$ `generate_private_key`()

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

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

$s \leftarrow \text{Enc}_{P_a^S}(S_t^S)$

$H_r \leftarrow \text{Hash}(r)$

$\text{csig} \leftarrow \text{Sign}_{S_a^S}(\iota + H_d + H_{\text{algo}} + \omega)$

$\text{sig} \leftarrow \text{Sign}_{S_a^S}(s + H_r + \iota + H_d + H_{\text{algo}} + \omega)$

return r, s, ω, csig, sig

end procedure

4.3 交付过程

如图 2所示为链下结果交付的流程。在步骤 3 中，需要引入链下的数据传输方式，如 HTTP,FTP,IPFS 等。图 2中调用合约时，省略了标识请求的参数（数据 ID、请求 ID）。

在步骤 6 中，智能合约使用 P_a^S 验证签名。注意 H_r 是由数据使用方提供的，链上记录了加密参数 ι, 数据哈希 H_d, 以及算法哈希 H_{algo}，且计算节点在步骤 2 已经提交了消耗的资源 ω，因此在步骤 6 中计算节点提供了 s 及 sig 后，容易验证 s 是否是有效的。

4.4 请求的状态

智能合约记录了每个请求的状态，并提供了一组可供数据合作双方调用的方法，这些方法的描述如下，

1. `requestOffChain`： 供数据使用方调用，用于创建一个请求。参数包括分析算法的参数（使用枢公钥加密后），枢公钥，使用典公钥加密后

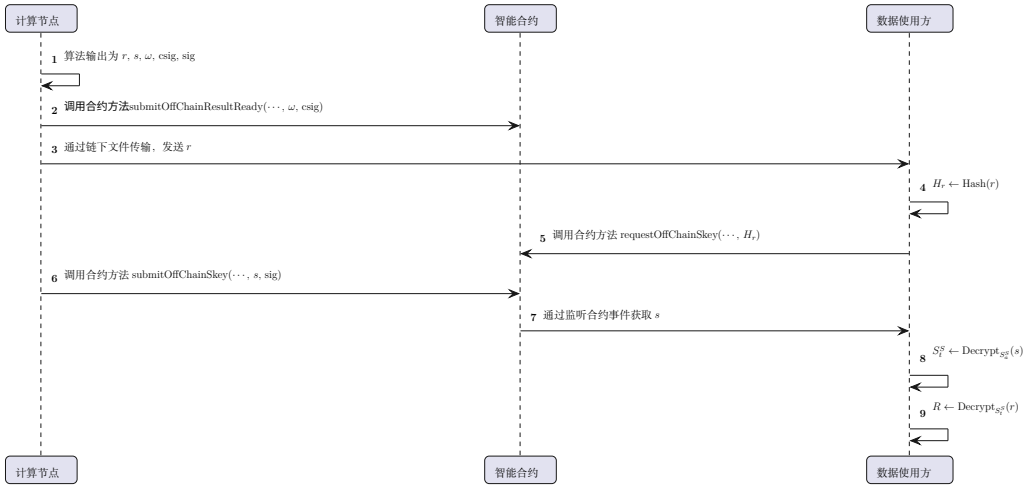

Figure 2: 链下结果交付的流程。对于合约方法的解释见下文。

的枢私钥，以及枢私钥转发所需的信息。根据场景，该合约会扣除必要的费用。

2. `submitOffChainResultReady`: 供计算节点调用，用于提醒数据使用方计算已经完成。参数包括了计算所消耗的费用及签名，根据场景，参数还可以附带下载加密结果的链接。为了保证调用方的身份，该方法会对签名进行验证。

3. `requestOffChainSkey`:供数据使用方调用，用于请求加密结果对应的私钥。参数包括加密结果的 Hash。

4. `submitOffChainSkey`:供计算节点调用，用于提交加密结果对应的私钥。参数包括使用加密后的私钥，以及签名。为了保证私钥的正确性，该方法会对签名进行验证。若验证通过，智能合约进行资金的分配和划转。

5. `revokeRequest`:供数据使用方调用，用于撤销请求。该方法仅在请求超时后方可调用。

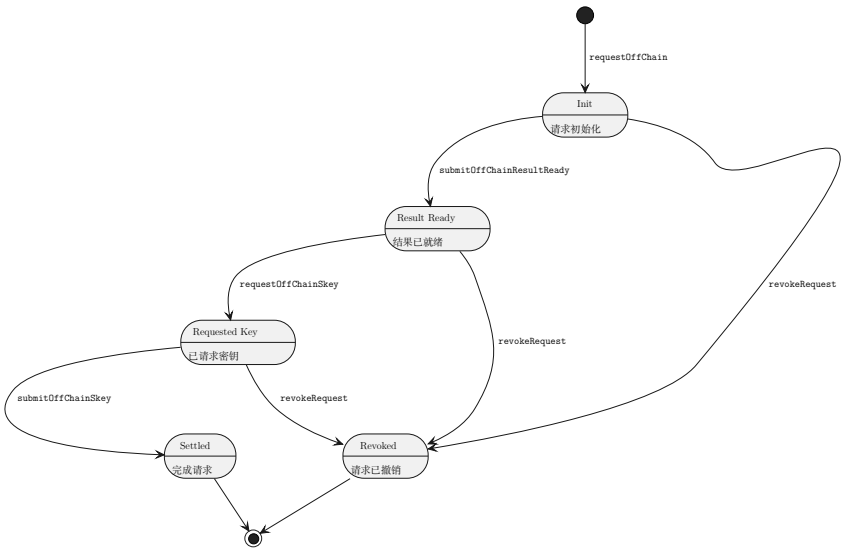

Figure 3: 智能合约中请求的状态随着智能合约方法调用的变化。

图 2中的智能合约记录了每个请求的状态，这些状态在智能合约被调用时发生改变。图 3表明了在不同调用下的状态转移情况。图 3忽略了调用合约时的权限要求。

4.5 安全性

图 2描述的交付过程保证了结果是可验证的、加密的、不可篡改的。

- 可验证：签名 sig 在 TEE 内由 S_a^S 签名，不可伪造，且可以通过智能合约进行验证;
- 加密：计算结果使用了仅数据使用方可见的密钥进行加密，因此计算节点不可见;
- 不可篡改：若计算结果被篡改，则数据使用方不能生成正确的 H_r，因此后续流程（步骤 6）不能执行成功。

5 Fidelius 中的实现

在 Fidelius 中，相关的处理逻辑在文件

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

`internal/results/offchain_result.h`

中。

智能合约 本功能的实现需要引入智能合约。智能合约的实现位于目录

`contracts/market/offchain/`.

需要注意的是，算法本身决定了结果是链上交付还是链下交付，而不是根据结果大小动态决定的。因此，链上交付或链下交付这一特性可以看作算法的一个属性。

6 系统功能

支持本功能需要更新用户的交互流程，显示交易的状态，以及数据使用方的解密方式。

此外，如图 2所示，数据使用方需要多次交互，这无疑会增加数据使用方的使用成本。在有平台（例如典枢）存在时，步骤 3 可以将 r 发送到平台，同时，步骤 6 中的内容也可以发送到平台，由平台代为操作步骤 4、5、7。不难发现，若平台会诚实的发送 r 给用户，系统的安全性并不会减弱。更新后的交付方式如图 4所示。

7 未来工作

无。

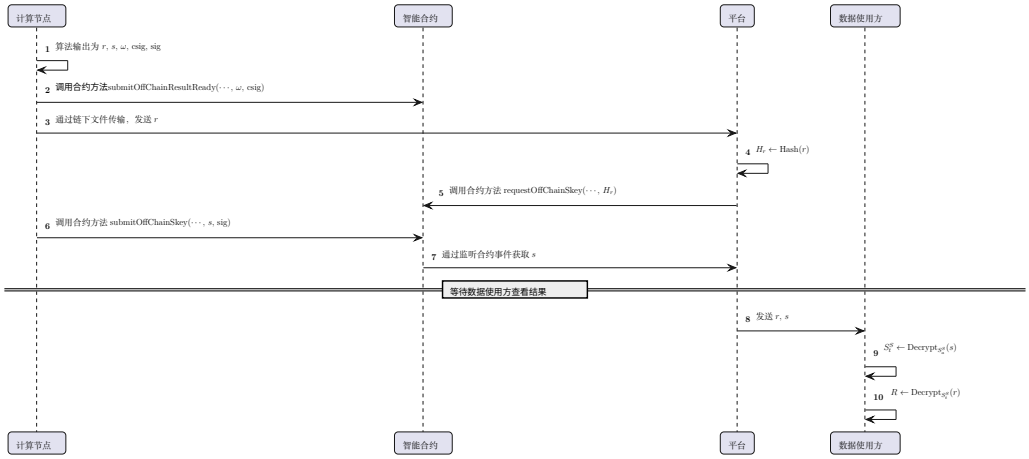

Figure 4: 存在平台时，链下数据交付的流程