



# 本科生毕业论文（设计）

题目： 基于 TEE 和区块链的  
隐私保护计算平台

姓 名 王明业

学 号 17343107

院 系 计算机学院

专 业 软件工程

指导教师 郑子彬 教授

2021 年 4 月 10 日

# 基于 TEE 和区块链的隐私保护计算平台

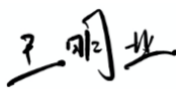
## A Privacy-Preserving Computing Platform based on TEE and Blockchain

姓 名	王明业
学 号	17343107
院 系	计算机学院
专 业	软件工程
指 导 教 师	郑子彬 教授

2021 年 4 月 10 日

## 学术诚信声明

本人郑重声明：所呈交的毕业论文（设计），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文（设计）不包含任何其他个人或集体已经发表或撰写过的作品成果。对本论文（设计）的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本论文（设计）的知识产权归属于培养单位。本人完全意识到本声明的法律结果由本人承担。

作者签名： 

日 期： 2021 年 4 月 10 日

## 摘要

TEE (Trusted Execution Environment) 即可信执行环境, 它保证加载到其中的代码和数据在保密性和完整性方面受到保护<sup>[1]</sup>。区块链本质是一个数据结构, 储存于区块链中的信息具有不可伪造、可追溯和公开透明的特征<sup>[2][3]</sup>。本文给出一种计算平台的设计, 该平台使用当下流行的加密方法如 RSA<sup>[4]</sup>、AES<sup>[5]</sup>, 并结合 TEE 和区块链技术, 可以全程保护用户的隐私。

**关键词:** TEE、可信执行环境、区块链、隐私保护、服务计算、计算平台、代码运行平台

## **ABSTRACT**

TEE (Trusted Execution Environment) guarantees that the confidentiality and the integrity of the code and data loaded into it are protected. The essence of the blockchain is a data structure, and the information stored in the blockchain has the characteristics of unforgeability, traceability and transparency. This paper presents a design of a computing platform. The platform which uses popular encryption methods such as RSA and AES, and integrates with TEE and blockchain technology, can protect the privacy of users throughout the process.

**Keywords:** TEE (Trusted Execution Environment), Blockchain, Privacy Preserving, Service Computing, Computing Platform, Codes Running Platform

# 目录

一、 绪论.....	4
二、 TEE、区块链和密码学技术在本平台的使用 .....	5
(一) TEE 工具.....	5
(二) 区块链工具 .....	6
(三) 密码学技术.....	7
1. 非对称加密算法 RSA.....	7
2. 对称加密算法 AES .....	8
3. 散列算法 SHA-256 .....	8
三、 整体架构与流程.....	9
(一) 整体架构.....	9
(二) 流程详解 .....	9
1. 用户端与密钥管理器进行密钥传递的一系列过程.....	10
2. 用户端发送代码文件给文件接收器的一系列过程.....	12
3. 文件接收器发送代码文件给代码运行器的一系列过程.....	14
4. 代码运行器将结果和过程信息发送给区块链记录的一系列过程.....	15
5. 用户端向区块链记录请求信息的一系列过程.....	16
(三) 安全性分析 .....	18
四、 模块（类）的详解.....	19
(一) 时序图 .....	19
(二) 类的详解 .....	22
1. Task 与 Client .....	22
2. KeyManager.....	25
3. FileReceiver .....	26
4. CodeRunner .....	27
5. BlockchainRecorder .....	27
五、 服务端接口设计 .....	29
(一) 密钥获取 .....	30
(二) 任务上传 .....	30
(三) 区块获取 .....	31
(四) 全链获取 .....	31
六、 样例实现.....	32
(一) 语言与第三方库 .....	32
(二) 部署与运行 .....	32
七、 参考文献.....	33
附录 A. 35	
八、 致谢.....	36

## 一、 绪论

当下的一些计算任务，如深度学习，需要很高的算力<sup>[6]</sup>，个人电脑的配置无法满足在可接受时长内完成这些计算任务并得到结果，很多用户会使用大公司的云计算平台运行这些计算任务。比较流行的一种云计算服务平台做法是：在物理机器上运行虚拟机，用户与虚拟机实例进行交互，用户对虚拟机有完整的访问和操作权限<sup>[7]</sup>，使用体验等同于以管理员身份在使用一个操作系统。例如，虚拟机实例的操作系统是 Ubuntu，那么用户的使用体验等同于以管理员身份在使用一个完整的 Ubuntu 操作系统。并且，公司监听虚拟机的使用以记账收费。

这样的云计算服务平台做法可能会有以下问题：

在物理机器上运行多个虚拟机，虚拟机的运行会占用资源。考虑到机器最主要的功能是运行计算任务，那么相对于计算任务占用资源，运行虚拟机占用资源可以看成是一种浪费。

小企业或者个人，如果有空闲计算资源想供他人使用，由于技术限制，他们可能无法像大公司一样在物理机器上运行多个相互隔离的虚拟机实例供用户使用，并进行监听与记账收费。并且，小企业或个人的空闲计算资源可能不够充裕，无法支持运行多个虚拟机并同时运行计算任务。

对于不熟悉系统操作的人员，面对一个虚拟机实例，需要花费时间精力去学习虚拟机系统的使用。而他们想要的服务，只是上传代码文件到服务器运行，然后得到结果。

本文设计的基于 TEE 和区块链的隐私保护计算平台，其基本运作流程是：平台在 TEE 中运行，用户上传计算任务（代码）到平台。平台令用户代码在 TEE 中执行，并得到运行结果以及运行过程信息。而后平台将加密的运行结果和运行信息存入区块链中，区块链可公开访问，用户访问区块链内容以获取运行结果。

本文设计的平台，只用在物理机器上运行一个带有在 TEE 中执行代码的功能的系统，然后在该系统中执行一套程序。这套程序会生成并管理密钥，接收客户端发来的代码，然后执行代码完成计算任务，最后把计算结果以及计算过程信息记录在区块链上。加密算法、散列算法等密码学技术保证了客户端和服务端数据传输的保密性；在 TEE 中执行程序保证了计算任务运行时的保密性；区块链上储存加密后的计算结果和计算过程信息，保证了计算结果的不可篡改性、保密性和匿名性，实现了全程的隐私保护。

本文设计的基于 TEE 和区块链的隐私保护计算平台只用在物理机器上运行一个操作

系统，不会有运行多个虚拟机占用过多资源的情况。并且，运行本文提出计算平台的技术门槛和开销，相比运行上述云计算服务平台要低。此外，用密码学技术、TEE 和区块链实现的全程隐私保护，可以让用户足够信任。而用户需要做的只是通过 HTTP<sup>[8]</sup>请求与计算平台进行交互，无需操作一个完整的系统。可见，本文设计的计算平台改善了前述云计算服务平台存在的缺点。

## 二、 TEE、区块链和密码学技术在本平台的使用

TEE 工具、区块链工具和一些密码学技术的使用是本文提出的隐私保护计算平台的关键所在。先了解 TEE 工具、区块链工具和平台使用到的密码学技术，有助于理解平台的架构、流程和模块功能。在此先给出 TEE 工具、区块链工具和平台使用到的密码学技术的描述，并提及其在本平台中的大致使用方式。

### （一） TEE 工具

比较流行的 TEE 开发工具是 Intel Software Guard Extensions (SGX)<sup>[9]</sup>。但 Intel SGX 相对比较底层，使用 Intel SGX 的 SDK 开发程序，过程非常繁琐：需要花费大量时间学习相关接口、编程模型以及 SGX SDK 的编译系统。

Youren Shen 等提出的 Occlum，是一个针对 Intel SGX 的内存安全、多进程的库操作系统<sup>[10]</sup>。Occlum 的最大优势是其易用性。使用 Occlum，可以不用编写任何额外的 SGX 相关编码，仅需要使用一些 shell 命令就可以使程序在 SGX 保护下运行，也即是在 TEE 中运行。此外，Occlum 还支持运行多种不同编程语言的程序，包括 C/C++、Python、Go 和 Java。



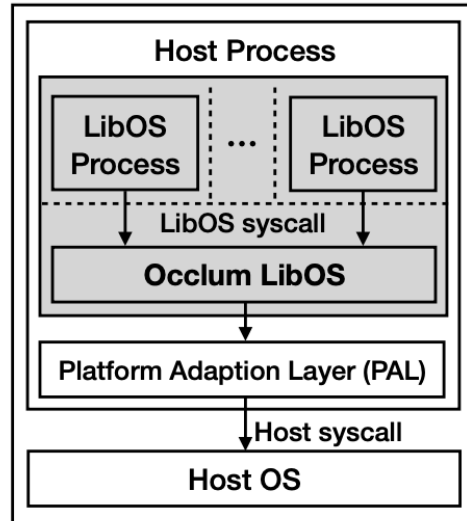


图 1 Occlum 库操作系统概述

如图 1 所示，Occlum 是建立在主操作系统（Host OS）上的库操作系统（LibOS 即 Library OS）。其具体使用方式是：在安装有 Occlum 的主系统上，使用 Occlum 的相关命令运行代码文件，即可实现代码在 TEE 中执行。事实上，Occlum 作者给出了一个内置 Occlum 的 Ubuntu 18.04 系统镜像，使用该镜像安装 Ubuntu 操作系统，即可在 Ubuntu 中使用 Occlum 命令，无需额外安装。该 Ubuntu 系统的其他使用，如文件系统、系统命令等，与一般的 Ubuntu 系统一样。

由于主系统是 Ubuntu，所以平台代码的部署、服务器维护等操作，跟使用一般 Ubuntu 机器的操作一样。要让平台和计算任务运行在 TEE 中，只需要编写 shell 脚本，使用 Occlum 命令，让相关代码运行即可。

## （二） 区块链工具

区块链本质是一种按照时间顺序将数据区块用类似链表的方式组成的数据结构<sup>[11]</sup>。一个基本的区块链，每个区块会包含<sup>[12]</sup>：索引（index）、时间戳（timestamp）、交易数据（transactions，事实上交易数据可以是任何想储存在区块上的数据）、校验（proof）和前一个区块的散列（the hash of the previous block）。

本平台会在代码中自行实现一个适用本平台需求的区块链。该区块链的区块中的交易数据实际记录的是每次计算任务的信息，包括这次任务的标示、计算结果和运行信息，具体将在后文讲解。并且，因为只有服务端运行计算任务并将相关信息写入区块链，所以该区块链只会在服务端增长，而不是多方共同作用使得区块链增长，所以省去了工作量证明

机制和校验。

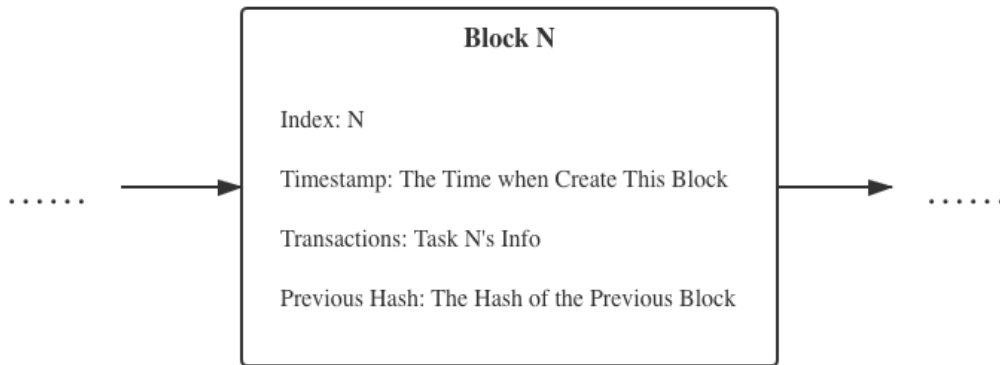


图 2 本平台所用区块链示意图（第 N 个区块）

### （三） 密码学技术

#### 1. 非对称加密算法 RSA

RSA 是目前被广泛使用的非对称加密算法。Ron Rivest, Adi Shamir 和 Leonard Adleman 于 1977 年共同提出了 RSA。RSA 的名称由三位作者姓氏首字母组合而成<sup>[13]</sup>。RSA 的具体操作实现原理以及实现细节在此不赘述，只是简单介绍一下其大致使用过程。

设 A 想要通过一个不安全的媒介接收 B 的一条私密消息，并使用 RSA 算法来保证消息传输的安全性。A 首先要按照 RSA 规定的密钥生成方式来产生一个公钥和一个私钥，其中公钥可以公开，私钥必须保持机密。B 获取 A 的公钥，然后使用 A 的公钥，对私密消息应用 RSA 加密流程，生成密文，将密文通过不安全的媒介传输给 A。A 得到密文后，可以用自己的私钥来解密，得到消息的原文。由于只有 A 拥有其公钥对应的私钥，所以即便他人在媒介上截取了密文，也无法解密得到消息原文。

此外，RSA 还可以对消息进行签名。设 A 想对传给 B 的消息签名，那么 A 先对消息计算散列值（信息摘要），然后 A 根据 RSA 算法流程，用私钥对该散列值加密（如同上述公钥对消息进行加密的过程），如此产生一个签名，并将该签名附在消息的后面，一同传输给 B。B 获得 A 传来的带有签名的消息后，可以根据 RSA 算法流程，用 A 的公钥对签名进行解密（如同上述私钥对消息进行解密的过程），得到一个值。然后 B 自己对消息计算散列值，如果此时得到的散列值与先前解密得到的值相同，那么 B 可以确定发信人持有 A 的私钥，且此消息在传输媒介上未被篡改。

RSA 算法本身要求明文长度要小于其模值，所以不能一次性加密过长的数据。并且，

RSA 加解密的速度比对称加密算法如 AES 慢得多。实际使用一般会用 RSA 加密对称加密算法如 AES 的密钥，再用 AES 加密消息。

在本平台中，RSA 算法会用来加密 AES 密钥、对传输数据签名，以及加密计算任务的散列值（实现时会保证散列算法得到的散列值长度允许 RSA 加密）。

## 2. 对称加密算法 AES

AES 即 Advanced Encryption Standard，是目前被广泛使用的非堆成加密算法，由 oan Daemen 和 Vincent Rijmen 共同提出。AES 具体操作实现原理以及实现细节同样不再赘述，在此简单介绍一下其大致使用过程。

AES 支持 128、192 和 256 位的密钥。设 A 打算使用 128 位长的密钥来对消息进行 AES 加密，那么 A 首先要生成一条长度为 128 位（16 字节）的数据作为密钥。然后按照 AES 算法的流程对消息进行加密生成密文。只要持有与 A 加密时使用的密钥相同的密钥，就可以解密密文得到消息原文。

在本平台中，AES 算法会用来加密一些较长的数据，例如 RSA 公钥、文件内容、计算任务运行结果和运行信息等。

## 3. 散列算法 SHA-256

SHA-256 属于 SHA-2 标准，由美国国家安全局研发。使用 SHA-256 算法，对任意给定的消息，能生成长度为 256 位的散列值（信息摘要）。并且，无法从消息的散列值推出原消息，也几乎不会有不同的消息产生相同的散列值。

在本平台中，SHA-256 会在 RSA 签名过程中使用到，并且，对于每个计算任务，也会按一定规则用 SHA-256 生成每个任务特有的散列值，该散列值会作为每个计算任务的唯一标识。

### 三、整体架构与流程

#### （一）整体架构

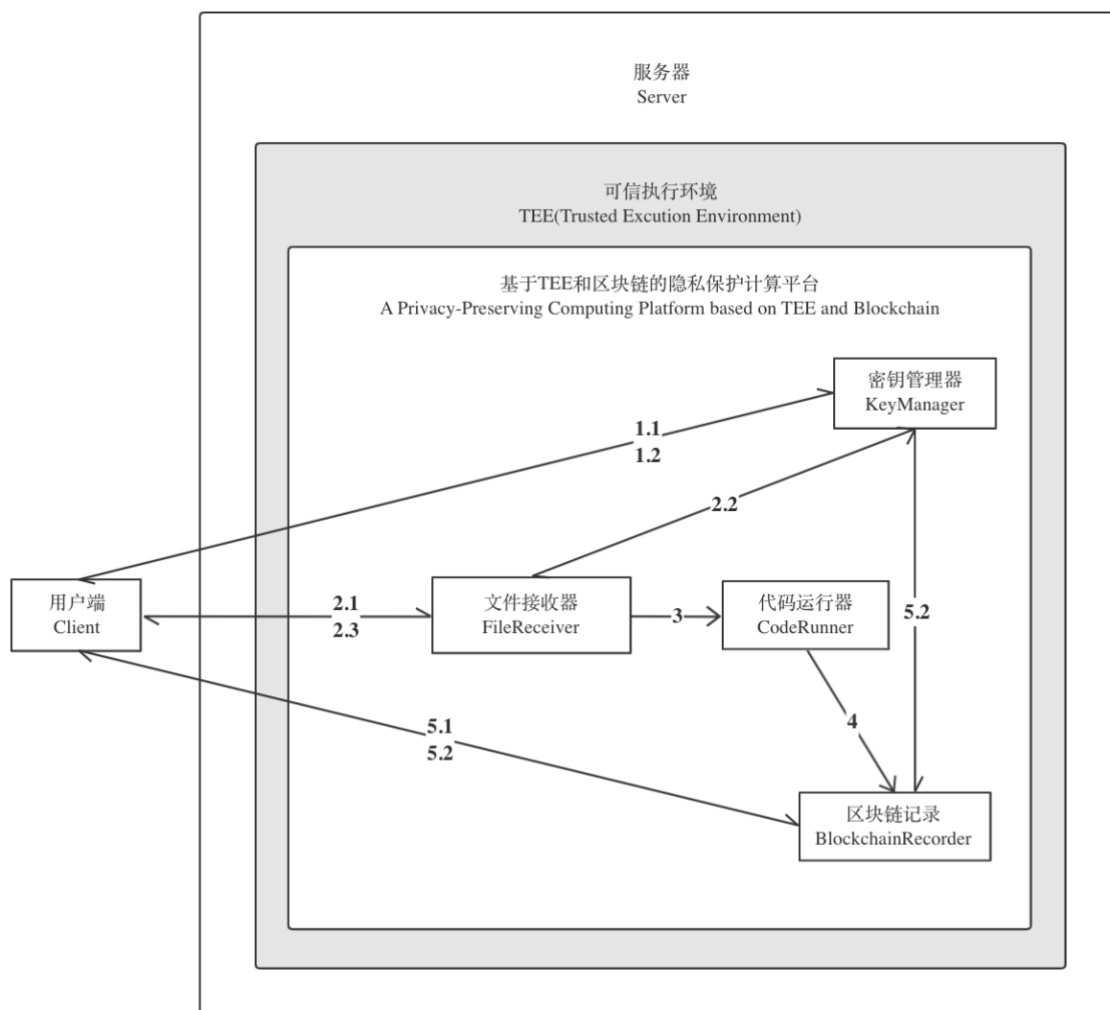


图 3 基于 TEE 与区块链的隐私保护计算平台 整体架构示意图

如图 3 基于 TEE 与区块链的隐私保护计算平台 整体架构示意图所示，本平台运行于服务器的 TEE 中。平台包括密钥管理器 KeyManager，文件接收器 FileReceiver、代码运行器 CodeRunner 和区块链记录 BlockchainRecorder。图中个模块间的连线表示模块间有交互；连线上的标号表示具体交互过程或具体流程，详情可见下一部分（二）流程详解。

#### （二）流程详解

本部分标号、指代等请参考：图 3 基于 TEE 与区块链的隐私保护计算平台 整体架构示意图。流程的标号代表流程发生的先后顺序，数字越小，代表流程越先发生。

流程中非对称加解密算法采用 RSA，对于一对 RSA 密钥  $KR_N$ ，称其中公钥为  $KR_{N_{pub}}$ ，

私钥为  $KR_{N\text{pri}}$ ;

流程中对称加解密算法采用 AES, AES 密钥称为 KA;

流程中散列（信息摘要）算法采用 SHA-256;

用户端 A 的第  $n$  次计算任务记为  $A_n$ , 如 A 的第 1 次计算任务记为  $A_1$ ;

那么, 用户端 A 执行计算任务  $A_1$  的流程如下:

(用户端 A 执行计算任务  $A_n$  ( $n > 1$ ) 时, 可省略下述流程 1, 仅进行 23456)

**1. 用户端与密钥管理器进行密钥传递的一系列过程**

1.1:

用户端 A 生成一对密钥  $KR_A$ , 将其中公钥  $KR_{A\text{pub}}$  发送给密钥管理器;

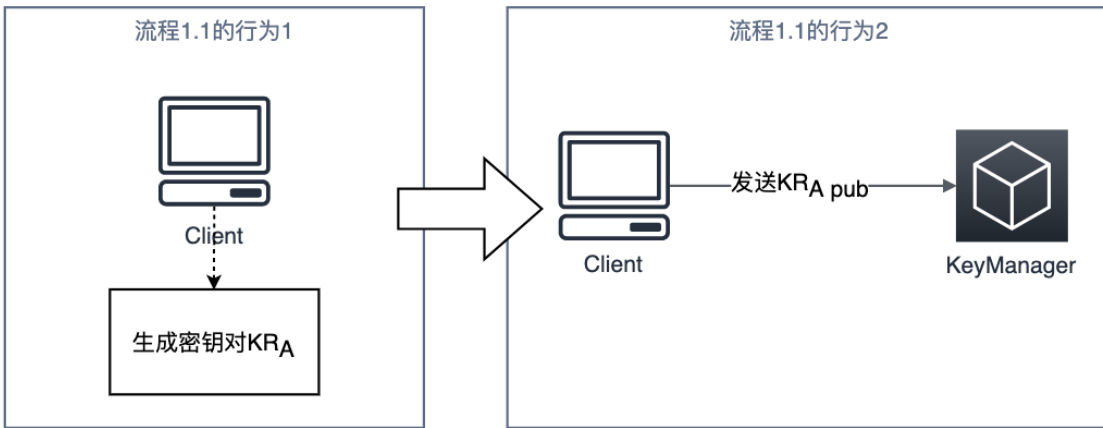


图 4 流程 1.1 示意图

1.2:

密钥管理器收到用户端 A 发来的包含公钥  $KR_{A\text{pub}}$  的请求后, 生成一对新密钥  $KR_S$  和一个新密钥 KA, 密钥管理器保存  $KR_{A\text{pub}}$  与  $KR_S$  及 KA 的对应关系。然后将新生成的密钥 KA 用  $KR_{A\text{pub}}$  加密,  $KR_{S\text{pub}}$  用 KA 加密, 并对加密后的密文用  $KR_{S\text{pri}}$  进行加签, 返回给用户端 A。

用户端 A 先用  $KR_{A\text{pri}}$  对密文解密得到 KA, 再用 KA 对密文解密得到  $KR_{S\text{pub}}$ , 而后用  $KR_{S\text{pub}}$  对密文进行验签, 验签成功说明得到的  $KR_{S\text{pub}}$  和 KA 可信。需注意  $KR_{S\text{pub}}$  虽然是公钥, 但用户端 A 不能随意公开  $KR_{S\text{pub}}$ 。具体原因见 (三) 安全性分析。

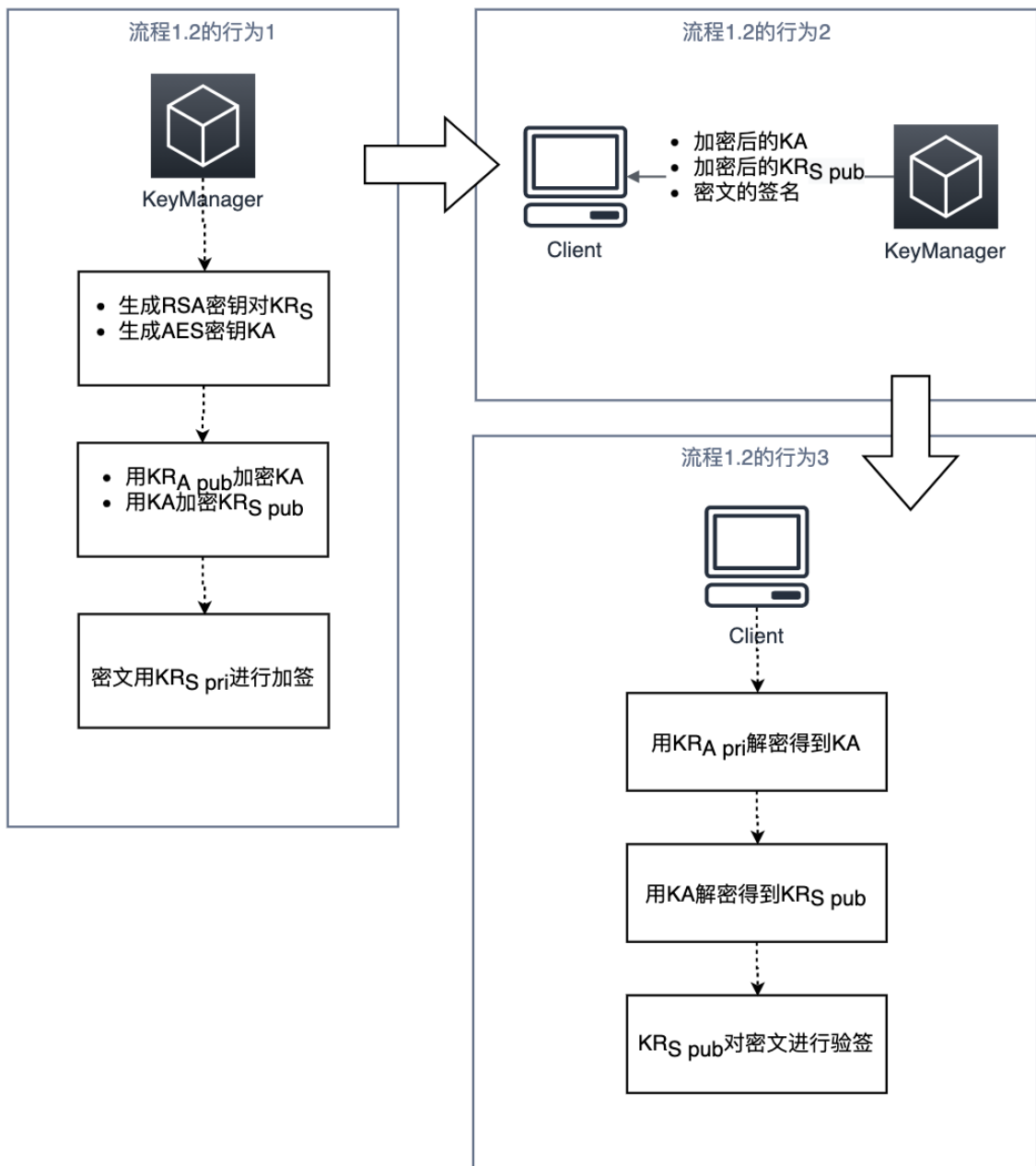


图 5 流程 1.2 示意图

流程 1 完成后，各模块拥有信息如下：

用户端 A	密钥管理器	文件接收器	代码运行器	区块链记录
$KR_A$ ,	$KR_A_{pub}$ ,			
$KR_S_{pub}$ ,	$KR_S$ ,			
KA	KA,			

表 1 流程 1 完成后各模块拥有信息

## 2. 用户端发送代码文件给文件接收器的一系列过程

### 2.1:

用户端 A 创建一个任务，该任务名称为 `task_name`，任务创建时间为 `create_time`，要执行的代码文件为 `File`，`File` 在用户端的路径为 `task_file_path`。对 `task_name`，`create_time` 和 `task_file_path` 拼接而成的字符串生成信息摘要 `HashA1`。

用  $KR_{S\ pub}$  对 `HashA1` 和 `KA` 进行加密，用 `KA` 对代码文件和数据文件 `File` 进行加密，，然后将自有的公钥  $KR_{A\ pub}$ ，用  $KR_{S\ pub}$  加密的 `HashA1 encrypted` 和 `KAencrypted`，用 `KA` 加密后的文件 `Fileencrypted`，以及  $KR_{A\ pri}$  对各加密内容的签名，一起发送给文件接收器；

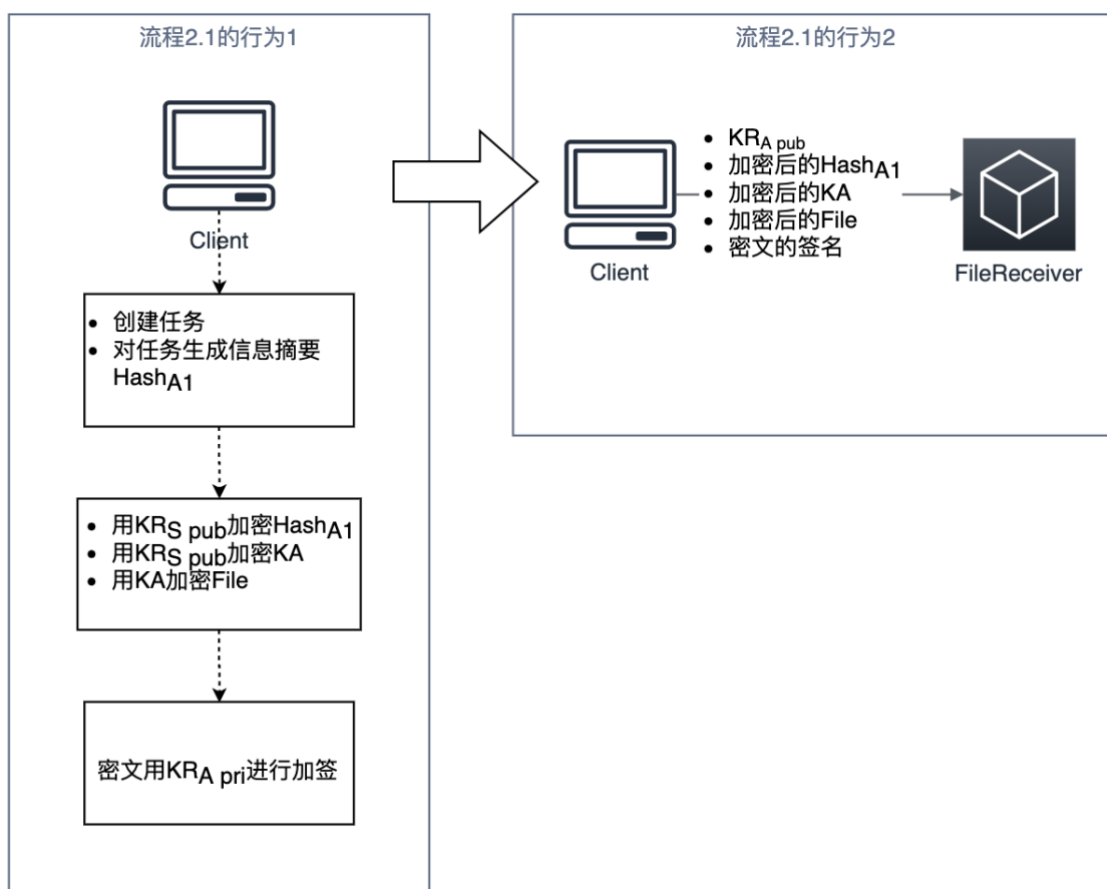


图 6 流程 2.1 示意图

### 2.2:

文件接收器先用收到的  $KR_{A\ pub}$  对各加密内容验签。

验签成功后，文件接收器用收到的  $KR_{A\ pub}$  从密钥管理器中获取对应的  $KR_S$  和 `KA`。

文件接收器用  $KR_{S\ pri}$  解密 `KAencrypted`，若解密后得到的 `KA` 与从密钥管理器获得的 `KA` 相同，那么说明本次从用户端 A 接收的信息确实由用户端 A 发出。

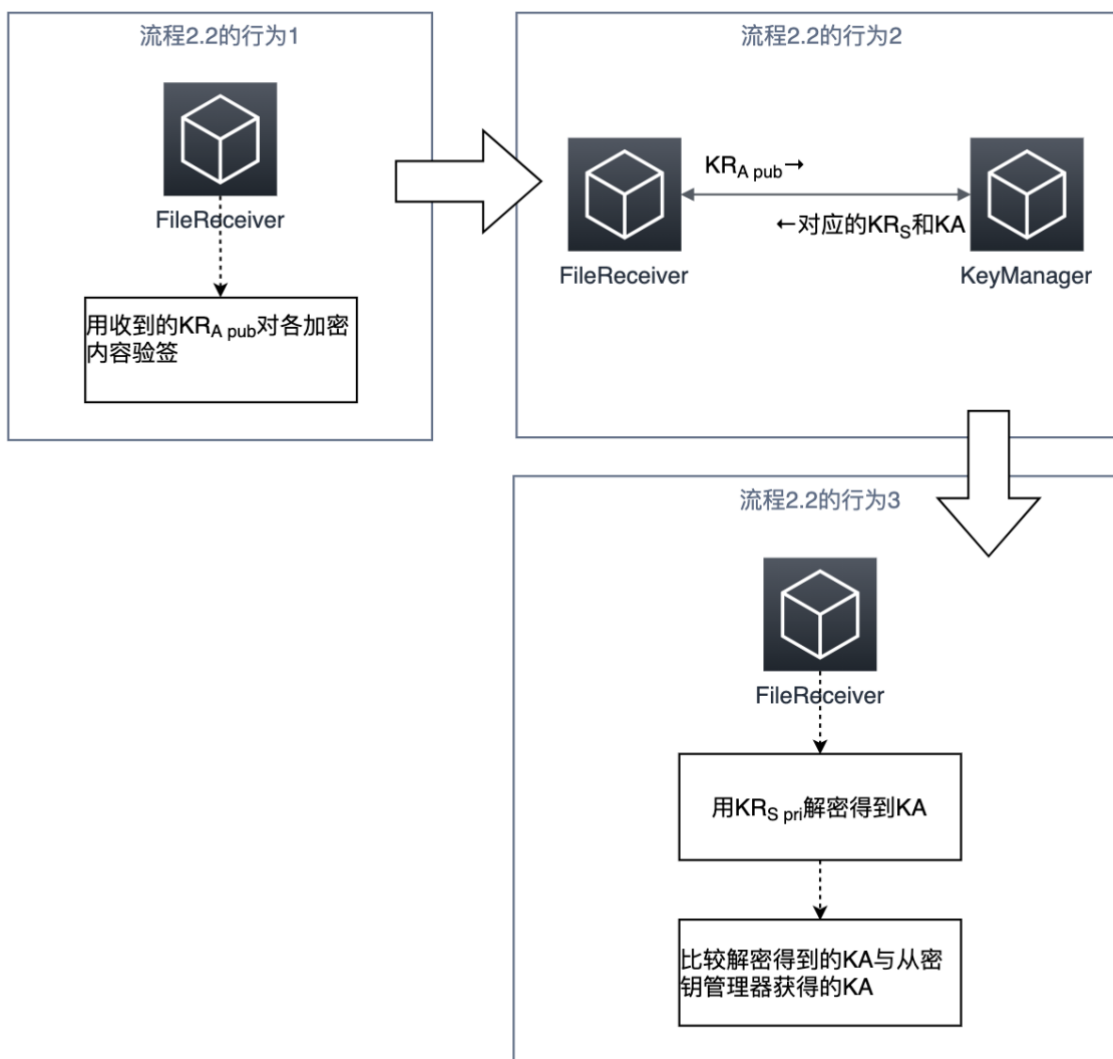


图 7 流程图 2.2 示意图

### 2.3:

文件接收器用  $KR_{S\ pri}$  解密  $Hash_{A1\ encrypted}$  得到  $Hash_{A1}$ ，用  $KA$  解密  $File_{encrypted}$  进行解密得到  $File$ 。并且文件接收器向用户端返回信息，表示成功接收该次任务。

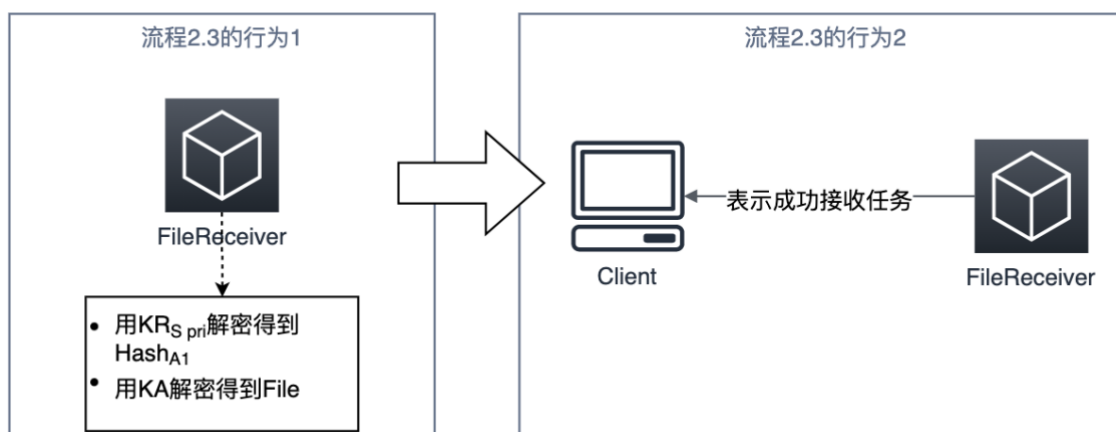




图 8 流程 2.3 示意图

流程 2 完成后，各模块拥有信息如下：

用户端 A	密钥管理器	文件接收器	代码运行器	区块链记录
$KR_A$ , $KR_{S\text{ pub}}$ , $KA$ , $File$ , $Hash_{A1}$	$KR_{A\text{ pub}}$ , $KR_S$ , $KA$ ,	$KR_{A\text{ pub}}$ , $KR_S$ , $KA$ , $File$ , $Hash_{A1}$		

表 2 流程 2 完成后各模块拥有信息

### 3. 文件接收器发送代码文件给代码运行器的一系列过程

文件接收器将  $KR_{A\text{ pub}}$ ， $KA$ ， $Hash_{A1}$  与  $File$  传递到代码运行器。

代码运行器运行  $File$ ，得到运行结果  $result_{A1}$ ，将运行结果用  $KA$  加密得到  $result_{A1\text{ encrypted}}$ ；

此外，还记录运行信息  $run\_info_{A1}$ ，并对其用  $KA$  加密，得到  $run\_info_{A1\text{ encrypted}}$ ；

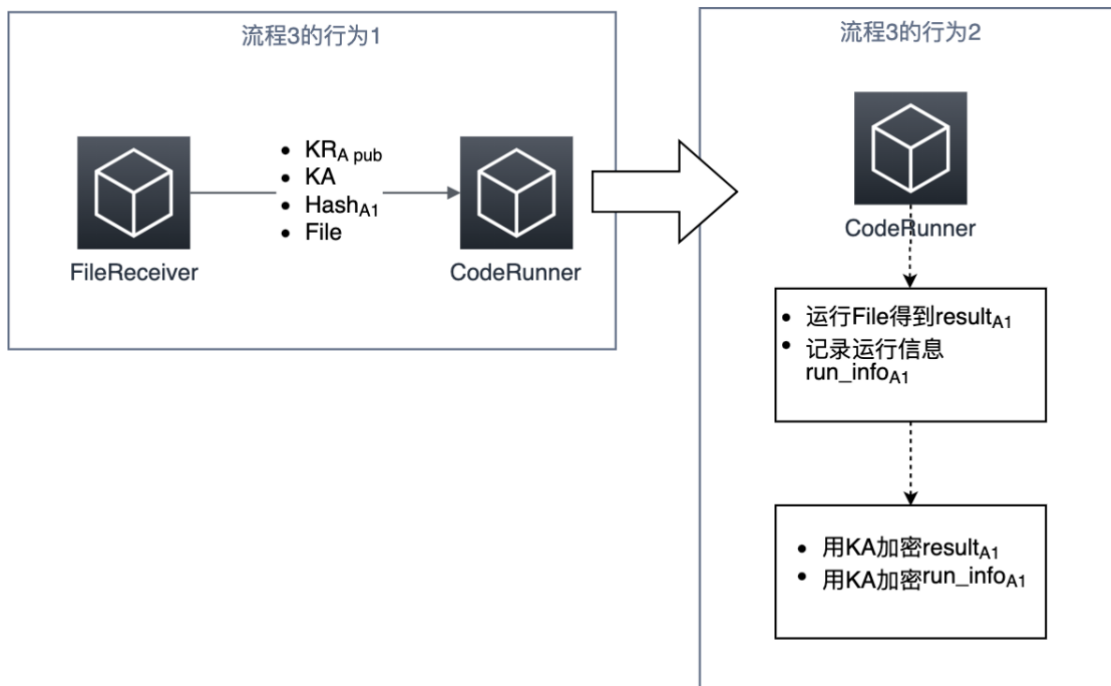


图 9 流程 3 示意图

流程 3 完成后，各模块拥有信息如下：

用户端 A	密钥管理器	文件接收器	代码运行器	区块链记录
$KR_A$ , $KR_{S\text{ pub}}$ , $KA$ , File, $Hash_{A1}$	$KR_{A\text{ pub}}$ , $KR_S$ , $KA$ ,	$KR_{A\text{ pub}}$ , $KR_S$ , $KA$ , File, $Hash_{A1}$	$KR_{A\text{ pub}}$ , $KA$ , File, $Hash_{A1}$ , $result_{A1\text{ encrypted}}$ , $run\_info_{A1\text{ encrypted}}$	

表 3 流程 3 完成后各模块拥有信息

#### 4. 代码运行器将结果和过程信息发送给区块链记录的一系列过程

代码运行器将  $KR_{A\text{ pub}}$ ， $Hash_{A1}$ ， $result_{A1\text{ encrypted}}$  和  $run\_info_{A1\text{ encrypted}}$  传递到区块链记录。

区块链记录创建新区块记录  $Hash_{A1}$ ， $result_{A1\text{ encrypted}}$  和  $run\_info_{A1\text{ encrypted}}$ ；并且，维护  $KR_{A\text{ pub}}$  和其每次任务 Hash 的对应关系，以便后续操作，并且也可基于此对应关系计费。

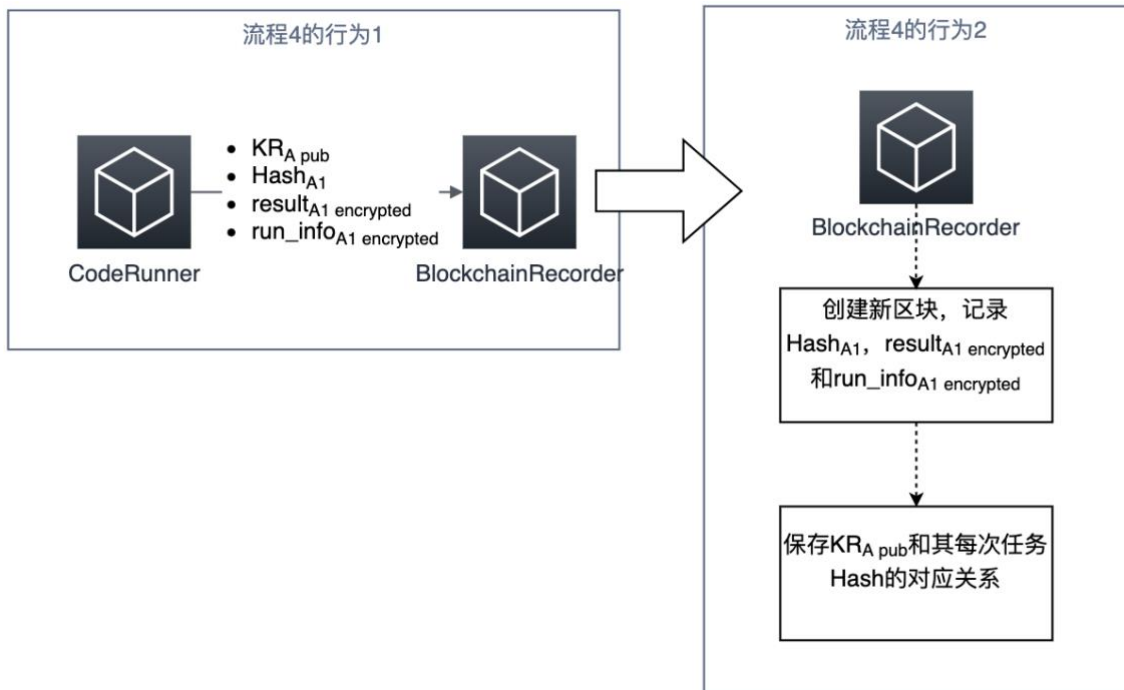


图 10 流程 4 示意图

流程 4 完成后，各模块拥有信息如下：

用户端 A	密钥管理器	文件接收器	代码运行器	区块链记录
$KR_A$ , $KR_{S\text{ pub}}$ , $KA$ , File, $Hash_{A1}$	$KR_{A\text{ pub}}$ , $KR_S$ , $KA$ ,	$KR_{A\text{ pub}}$ , $KR_S$ , $KA$ , File, $Hash_{A1}$	$KR_{A\text{ pub}}$ , $KA$ , File, $Hash_{A1}$ , $result_{A1\text{ encrypted}}$ , $run\_info_{A1}$ $run\_info_{A1\text{ encrypted}}$	$KR_{A\text{ pub}}$ , $Hash_{A1}$ , $result_{A1\text{ encrypted}}$ , $run\_info_{A1}$ $run\_info_{A1\text{ encrypted}}$

表 4 流程 4 完成后各模块拥有信息

## 5. 用户端向区块链记录请求信息的一系列过程

用户端 A 将  $Hash_{A1}$  发送给区块链记录，以查询记录着任务 A1 信息的区块。

区块链根据自身维护的  $KR_{A\text{ pub}}$  和其每次任务 Hash 的对应关系，通过  $Hash_{A1}$  得到相应的  $KR_{A\text{ pub}}$ ，用  $KR_{A\text{ pub}}$  向密钥管理器请求对应的  $KR_{S\text{ pri}}$ ，而后用  $KR_{S\text{ pri}}$  将记录着  $Hash_{A1}$  的区块加签，再将记录着  $Hash_{A1}$  的区块和签名一同返回给用户端 A。

用户端 A 可用  $KR_{S\text{ pub}}$  验签。然后用  $KA$  解密区块中的  $result_{A1\text{ encrypted}}$  和  $run\_info_{A1\text{ encrypted}}$ ，得到  $result_{A1}$  和  $run\_info_{A1}$ ；

注：用户端 A 也可以请求同步整个区块链记录到用户端 A；

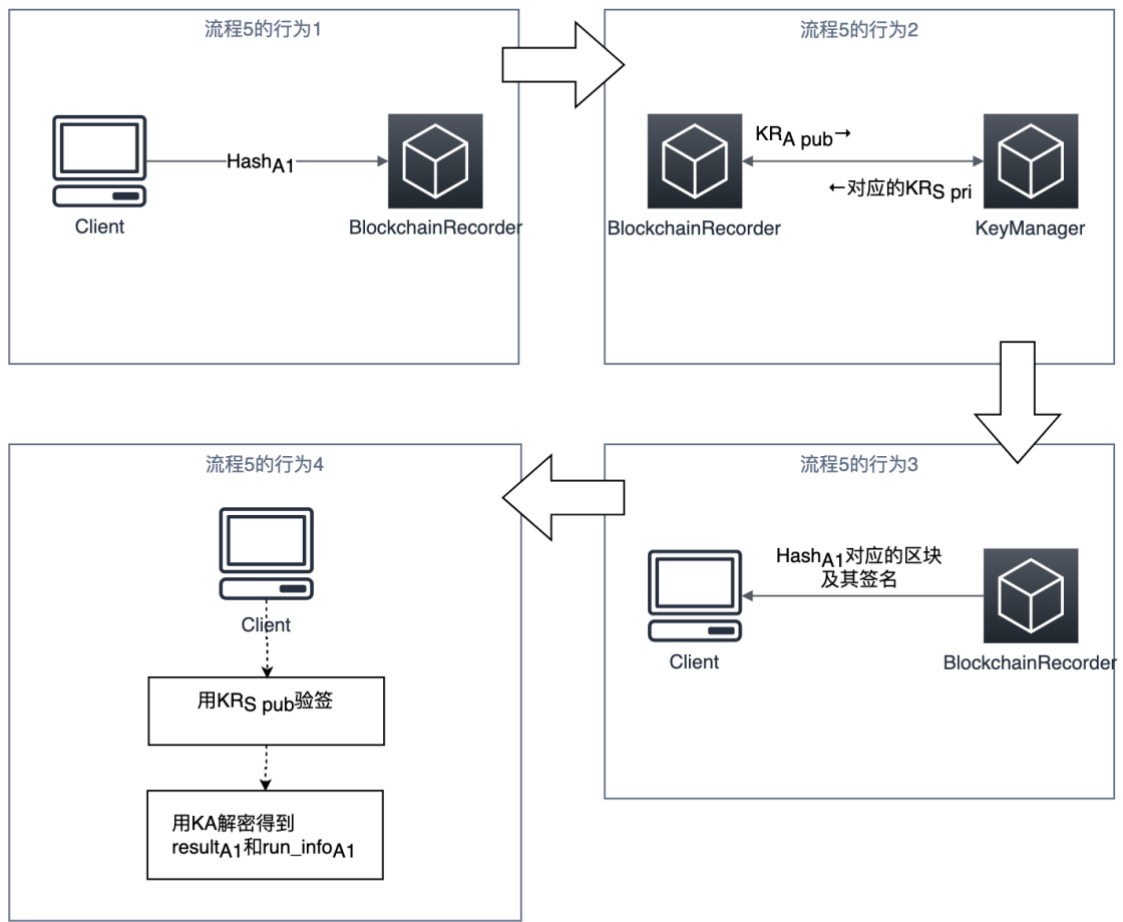


图 11 流程 5 示意图（用 Hash<sub>A1</sub> 请求）

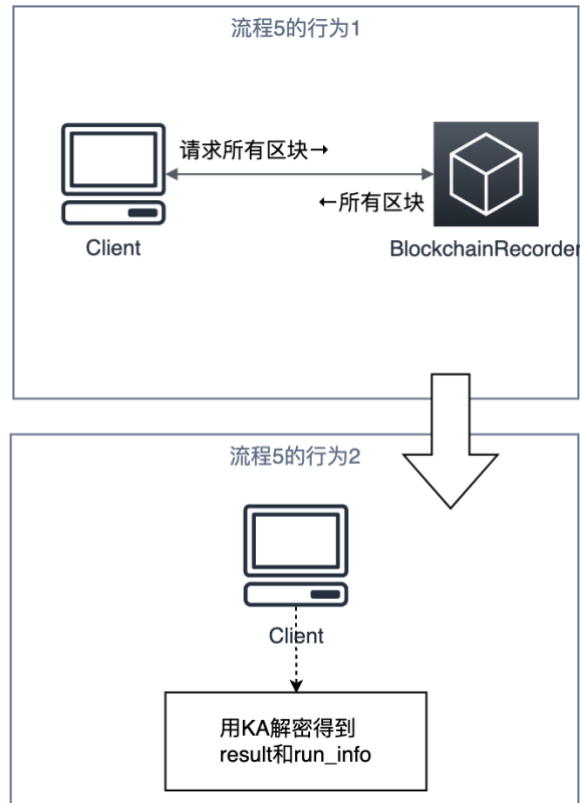


图 12 流程 5 示意图（请求整个区块链）

流程 5 完成后，各模块拥有信息如下：

用户端 A	密钥管理器	文件接收器	代码运行器	区块链记录
$KR_A$ , $KR_{S\text{ pub}}$ , $KA$ , File, $Hash_{A1}$ , $result_{A1}$ , $run\_info_{A1}$ ,	$KR_{A\text{ pub}}$ , $KR_S$ , $KA$ ,			$KR_{A\text{ pub}}$ , $Hash_{A1}$ , $result_{A1\text{ encrypted}}$ , $run\_info_{A1\text{ encrypted}}$ ,

表 5 流程 5 完成后各模块拥有信息

### （三） 安全性分析

流程 2 中有一个密钥比对的过程：用户端 A 用服务端分发给用户端 A 的独特的 RSA 公钥加密独特的 AES 密钥，服务端用自身储存的、对应用户端 A 的 RSA 私钥解密 AES

密钥，再比对解密结果是否与储存的对应用户端 A 的 AES 密钥一致。这一过程实际是一个身份验证过程，要破解这样的过程，需要破解服务端分发的独特 RSA 公钥（假设使用 RSA-1024，需破解 1024 位密钥的 RSA 算法），还要破解 AES 密钥（假设使用 AES-128，需破解 128 位的密钥，注意不是破解算法，因为 AES 密钥在此只是做一个比对）。如果服务端对于所有用户端都只使用同一套 RSA 密钥，并只维护用户端公钥-独特 AES 密钥的对应关系，那么这个身份验证过程实际只剩下一个 AES 密钥比对的过程，假设使用 AES-128，那么只需破解 128 位的密钥（同样只是破解密钥而不是破解算法，因为只是做一个比对）。也即，对每个用户端分发独特的服务端公钥，能保证流程图 2 身份验证过程的可靠性，该可靠性基于 RSA 算法的可靠性；如果没有分发独特公钥，只比对 AES 密钥，攻击者等同于在破解一个 128 位的密码。所以，服务端给每个用户端分发独特公钥，用户保密分发到的服务端公钥，这样的做法会更安全。

平台本身和用户的计算任务都是在同一服务器上的 TEE 中执行的。考虑到 TEE 执行环境可信，所以认为在 TEE 中进行的流程安全性与隐私均得到保障。

除了在 TEE 中实行的行为外，流程 1、2、5 中还包含客户端与平台的交互。平台实际运行时，该交互会通过 HTTP<sup>[8]</sup>请求实现。而 HTTP 使用明文传输内容，所以未经加密的信息可能不安全。在流程 1、2、5 中，未加密就进行传输的信息有：用户端 A 的 RSA 公钥、计算任务的散列值和区块信息。而 RSA 公钥本身就是可公开的，计算任务的散列值和区块信息本身就以明文写在区块链上，即本身就是公开的。所以这些未经加密就传输的信息不造成隐私的泄露。此外，每次传输的加签验签步骤，保证了信息的不可篡改性。

综上，平台可以实现全程的隐私保护。

## 四、 模块（类）的详解

图 3 基于 TEE 与区块链的隐私保护计算平台 整体架构示意图 中列出的各个模块（小直角矩形），即是需要实现的类。各个模块名即是各个类名。

### （一） 时序图

基于前述整体架构和流程详解，可以给出本平台运行的时序图。

需注意此处的标号与序号仅代表该时序图中事件的排列，与三、整体架构与流程中的标号和序号没有对应关系。

并且，图中省略了类保存信息、加密、解密和验签等默认自身进行的行为。  
为能清晰阅览，令时序图单独占一页篇幅，请见下页。

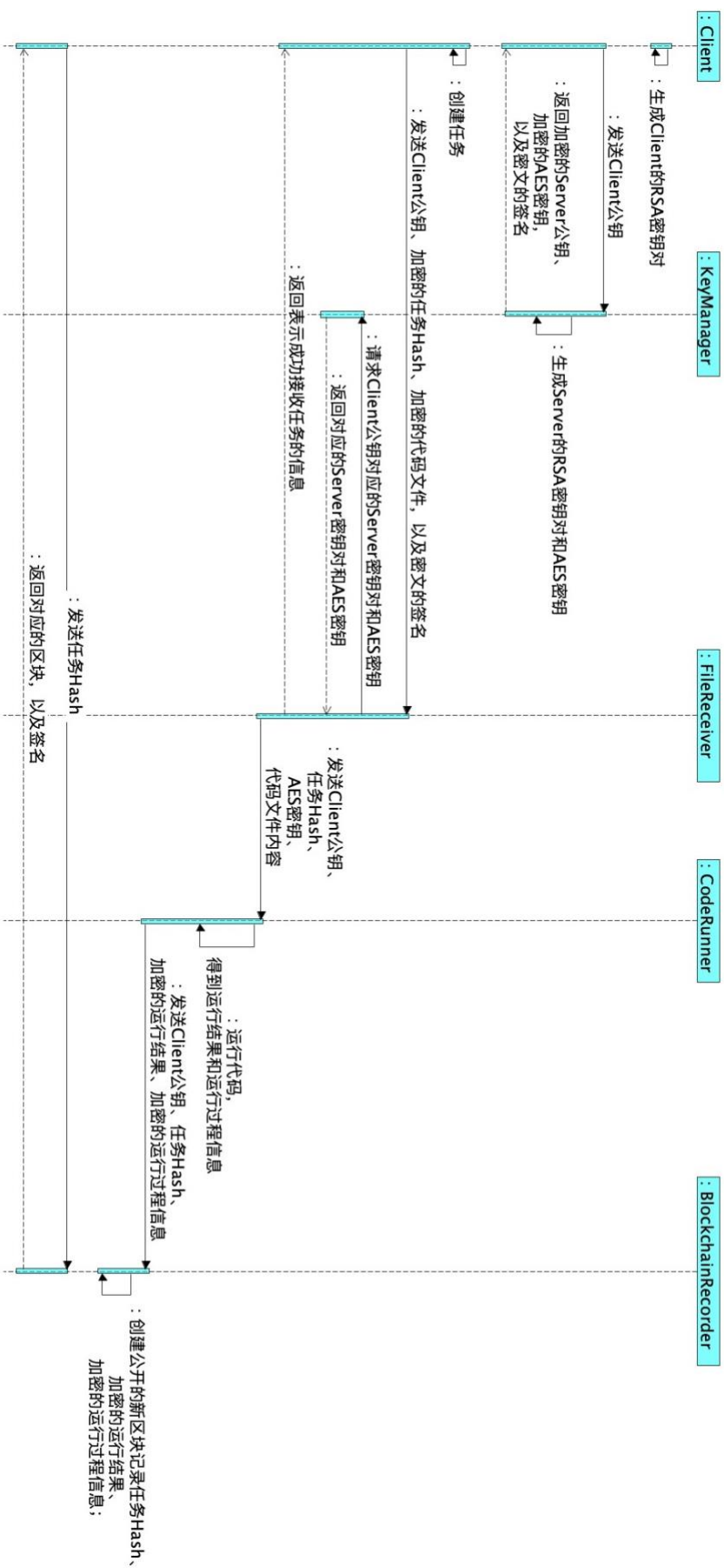


图 13 时序图



## （二） 类的详解

平台样例使用 Python 3.7 实现，根据 Python 命名规范<sup>[4]</sup>，类名使用首字母大写驼峰命名(CapWords)；函数名、一般变量名和实例名使用小写下划线命名(lower\_with\_under)。Python 中类的构造函数是 `__init__()`，而不是一个与类同名的函数。并且，在方法中使用成员变量，会用 `self`.前缀而不是 `this`.前缀。此外，类中定义的一般方法，需定义第一个传入参数为“`self`”（但使用方法时无需传 `self`）。本部分会使用这些规则。

### 1. Task 与 Client

Task 被 Client 使用。

Task 类的成员变量：

`task_name`: str, 任务名；

`create_time`: str, 创建时间；

`task_file_path`: str, 代码文件在用户端的路径；

`task_hash`: bytes, 任务散列值；

`has_result`: bool, 是否已得到执行结果；

`result`: str, 任务执行结果的内容；

`run_info`: str, 任务执行过程信息；

Task 类的方法：

`__init__(self, task_name, create_time, task_file_path, task_hash)`:

构造函数，传入参数对类中的同名成员变量赋值。`self.has_result` 置为 `False`。其余变量置为 `None`。

Client 类的成员变量：

`rsa_private_key`: bytes, Client 的 RSA 私钥；

`rsa_public_key`:: bytes, Client 的 RSA 公钥；

`aes_key`: bytes, Server 返回给 Client 的 AES 密钥；

`aes_key_valid`: bool, AES 密钥是否验签成功

`server_public_key`: bytes, Server 返回给 Client 的，Server 的 RSA 公钥；

server\_public\_key\_valid: bool, Server 的 RSA 公钥是否验签成功;

task\_info: list, 储存任务信息的列表, 其中储存的是 Task 类;

Client 类的方法:

`__init__(self):`

构造函数, 生成 RSA 密钥对, 私钥赋给 `self.rsa_private_key`, 公钥赋给 `self.rsa_public_key`。其他成员变量: bool 变量置为 False, 列表变量置为空列表, 其余变量赋值 None。

`decrypt_AES_key(self, enc_aes_key: bytes) -> None:`

传入参数: `enc_aes_key`, 即 Server 传来的加密的 AES 密钥;

返回值: None;

用 `self.rsa_private_key` 解密 `enc_aes_key`, 将解密后的 AES 密钥赋给 `self.aes_key`。

`decrypt_server_public_key(self, enc_public_key: bytes) -> None :`

传入参数: `enc_public_key`, 即 Sever 传来的加密的 RSA 公钥;

返回值: None;

用 `self.aes_key` 解密 `enc_public_key`, 将解密后的 Server 公钥赋给 `self.server_public_key`。

`validate_enc_keys(self, enc_aes_key: bytes, enc_public_key: bytes, enc_aes_signature: bytes, public_key_signature: bytes) -> bool:`

传入参数: `enc_aes_key`, 即 Server 传来的加密的 AES 密钥;

`enc_public_key`, 即 Sever 传来的加密的 RSA 公钥;

`enc_aes_signature`, 即 Server 传来的 `enc_aes_key` 的签名;

`public_key_signature`: 即 Server 传来的 `enc_public_key` 的签名;

返回值: 如果验签成功返回 True, 失败返回 False;

用 `self.server_public_key` 对密钥验签, 如果验签成功, 将 `self.aes_key_valid` 和 `self.server_public_key_valid` 置为 True, 并返回 True。

失败则无其他操作, 返回 False。

`rsa_encrypt_aes_key(self) -> bytes:`

传入参数：无；

返回值：self.server\_public\_key 加密的 self.aes\_key。

`aes_encrypt_file_bytes(self, file_path: str) -> bytes:`

传入参数：file\_path，即代码文件路径；

返回值：加密后的代码文件内容；

用 self.aes\_key 加密指定代码文件的内容。

`sign_by_private_key(self, enc_text: bytes) -> bytes:`

传入参数：enc\_text，即待签名的密文；

返回值：针对该内容的签名；

用 self.rsa\_private\_key 对 enc\_text 签名。

`generate_task(self, task_name: str, task_file_path: str) -> dict:`

传入参数：task\_name，即定义的任务名；

task\_file\_path，即任务代码文件所在路径；

返回值：一个字典，包含了将要发送到文件接收器的信息，包括 Client 的公钥、加密的 AES 密钥、加密的任务散列值和加密的代码文件内容，以及各加密信息的签名。

该方法包括的操作有：用 task\_name、当前时间戳 time 和 task\_file\_path 组成一个字符串，生成散列值作为 task\_hash；用 self.server\_public\_key 加密 task\_hash；

调用 self.rsa\_encrypt\_aes\_key() 方法得到用 self.server\_public\_key 加密的 self.aes\_key；

根据文件路径读取文件内容，用 self.aes\_key 加密文件内容；调用 self.sign\_by\_private\_key() 方法对各加密内容验签；

用 task\_name、time、task\_file\_path 和 task\_hash 创建 Task 实例，并将该 Task 实例加入 self.task\_info 列表中。

`validate_block(self, block: dict, signature: bytes) -> bool:`

传入参数：block，即待验签的区块内容；signature，即待验签的签名；

返回值：如果验签成功返回 True，失败返回 False；

`decrypt_and_save_results(self, block: dict) -> dict:`

传入参数: `block`, 即区块内容;

返回值: 解密后的运行结果和过程信息;

用 `self.aes_key` 解密区块中储存的加密的运行结果和加密的过程信息, 将得到的结果和过程信息放入 `self.task_info` 中对应的 `Task` 里。

## 2. KeyManager

`KeyManager` 类的成员变量:

`client_key_map_server_key: dict`, 保存各个用户端公钥与相应服务端公钥、服务端私钥和 AES 密钥的对应关系。

`KeyManager` 类的方法:

`__init__(self):`

构造函数, 初始化 `self.client_key_map_server_key` 为空字典。

`generate_key_for_client_key(self, client_key: bytes) -> None:`

传入参数: `client_key`, 即用户端的 RSA 公钥;

返回值: `None`;

生成一对新的 RSA 密钥对和一个新的 AES 密钥, 在 `self.client_key_map_server_key` 中保存 `client_key` 与新生成 RSA 密钥对和 AES 密钥的对应关系。

`get_keys_by_client_key(self, client_key: bytes) -> dict:`

传入参数: `client_key`, 即用户端的 RSA 公钥;

返回值: `client_key` 对应的 Server 的 RSA 密钥对 (公钥和私钥) 和 AES 密钥;

访问 `self.client_key_map_server_key` 以获得 `client_key` 对应的密钥。

`return_encrypted_keys_and_sign(self, client_key: bytes) -> dict:`

传入参数: `client_key`, 即用户端的 RSA 公钥;

返回值: 加密的 `client_key` 对应的 Server 的 RSA 公钥、加密的 `client_key` 对应的 AES

密钥，以及密文的签名；

用 `client_key` 对应的 AES 密钥加密 Server 的 RSA 公钥，用 `client_key` 加密 AES 密钥，用 `client_key` 对应的 Server 的 RSA 私钥进行签名。

### 3. FileReceiver

FileReceiver 类的成员变量：无。

FileReceiver 类的方法（构造函数为空）：

```
validate_and_decrypt_task_info(self,  
                                key_manager: KeyManager,  
                                client_public_key: bytes,  
                                enc_aes_key: bytes,  
                                enc_aes_key_signature: bytes,  
                                enc_task_hash: bytes,  
                                enc_task_hash_signature: bytes,  
                                enc_file_content: bytes,  
                                enc_file_content_signature: bytes) -> Tuple:
```

传入参数：key\_manager，即密钥管理器实例；

client\_public\_key: Client 的 RSA 公钥；

enc\_aes\_key 和 enc\_aes\_key\_signature: 加密的 AES 密钥及其签名；

enc\_task\_hash 和 enc\_task\_hash\_signature: 加密的任务散列值及其签名；

enc\_file\_content 和 enc\_file\_content\_signature: 加密的代码文件内容及其

签名；

返回值：元组的第一项：返回给 Client 的字符串，若未成功接收任务，返回错误内容文本，若成功接收任务，返回接收成功文本；元组第二项：返回给 CodeRunner 的内容，若未成功接收任务，为 None，若成功接收任务，为包含 Client 公钥、AES 密钥、任务散列值和文件内容（均为解密后的原文）的字典；

该方法包括的操作有：用 `client_key` 从 `key_manager` 中查询得到对应的 Server 的 RSA 公钥 `server_public_key`，Server 的 RSA 私钥 `server_private_key` 和 AES 密钥 `aes_key`；

用 `server_private_key` 验证各项签名，然后解密 `enc_aes_key`，将解密得到内容与 `aes_key`

进行比对。完成比对后，用 `aes_key` 解密 `enc_task_hash` 和 `enc_file_content`。最后返回相应内容。

#### 4. CodeRunner

`CodeRunner` 类的成员变量：无。

`CodeRunner` 类的方法（构造函数为空）：

```
run_code_file(self, client_public_key: bytes,  
               aes_key: bytes,  
               task_hash: bytes,  
               file_content: bytes) -> dict:
```

传入参数： `client_public_key`，即 `Client` 的 RSA 公钥；

`aes_key`，即 AES 密钥；

`task_hash`，即任务散列值；

`file_content`，即代码文件内容；

返回值：字典，其中包含 `client_public_key`、任务散列值、加密的代码运行结果和加密的运行过程信息；

该方法包括的操作有：将代码保存至临时文件，对临时文件赋予运行权限，运行代码并记录运行过程信息（样例中记录的是运行时间），删除临时文件，加密运行结果和运行过程信息。

#### 5. BlockchainRecorder

`BlockchainRecorder` 类的成员变量：

`client_public_key_maps_task_hash`: dict, 记录每个用户端 RSA 公钥对应的任务散列值；

`chain`: list, 链，其中是每个区块；

`current_transactions`: list, 当前的交易数据；

`BlockchainRecorder` 类的方法：

```
__init__(self):
```

构造函数，成员变量均置为空。并调用一次 `self.new_block`，生成创世区块，创世区块

的 `previous_hash` 指定为字符串"1"的 bytes;

```
new_block(self, previous_hash: bytes = None) -> block:
```

传入参数: `previous_hash`, 前一个区块的散列值, 默认为 `None`, 即默认是计算前一个区块的散列值作为 `previous_hash`, 无需特别指定。只有创建创世区块的时候要指定 `previous_hash`;

返回值: 该方法产生的新 `block`;

`block` 是一个字典。新的 `block` 的 `index` 是 `self.chain` 列表的长度加 1; `timestamp` 是从 1970 年 1 月 1 日 00:00:00 (UTC)开始到现在经过的秒数, 调用 `time.time()`即可获取; `transactions` 是 `self.current_transactions`; `previous_hash` 是 `self.chain` 最后一个元素 (也即是链上的最后一个 `block`) 的散列值, 或者是传入参数指定的值。创建完新 `block` 后, 将 `self.current_transactions` 置为空, 然后将新 `block` 放入 `self.chian` 的末尾。

```
new_transaction(self, task_hash: bytes,  
                 enc_result: bytes,  
                 enc_run_info: bytes) -> int:
```

传入参数: `task_hash`, 即任务的散列值;

`enc_result` 即加密后的结果;

`enc_run_info` 即加密后的运行过程信息;

返回值: 当前链上最后一个区块的 `index`;

该方法用 `task_hash`, `enc_result` 和 `enc_run_inf` 创建新交易, 并将新交易放入 `self.current_transactions`。

```
new_record(self, client_public_key: bytes,  
            task_hash: bytes,  
            enc_result: bytes,  
            enc_run_info: bytes) -> bool:
```

传入参数: `client_public_key`, 即用户端的 RSA 公钥;

`task_hash`, 即任务散列值;

`enc_result`, 即加密的运行结果;

`enc_run_info`，即加密的运行过程信息；

返回值：成功完成全部操作返回 `True`，否则完成 `False`；

首先更新 `self.client_public_key_maps_task_hash`，向其中添加 `client_public_key` 与 `task_hash` 的关系。然后调用 `self.new_transaction()`，用 `task_hash`，`enc_result` 和 `enc_run_info` 生成新交易，然后调用 `self.new_block` 生成新区块。

`find_client_key_by_task_hash(self, task_hash: bytes) -> bytes:`

传入参数：`task_hash`，即任务散列值；

返回值：拥有该 `task_hash` 的用户端 RSA 公钥，若没找到，返回 `None`；

从 `self.client_public_key_maps_task_hash` 中找到 `task_hash` 属于哪个用户端 RSA 公钥。

`find_block_by_task_hash(self, task_hash: bytes) -> block:`

传入参数：`task_hash`，即任务散列值；

返回值：交易数据中拥有该 `task_hash` 的 `block`，若没找到，返回 `None`；

遍历链以寻找对应区块。

`return_block_and_signature(self, task_hash: bytes, key_manager: KeyManager) -> dict:`

传入参数：`task_hash`，即任务散列值；`key_manager`：即 `KeyManager` 实例；

返回值：交易数据中拥有该 `task_hash` 的 `block`，以及该 `block` 的签名；

调用 `self.find_block_by_task_hash` 找到 `task_hash` 对应的 `block`，调用 `self.find_client_key_by_task_hash` 找到该 `task_hash` 对应的用户端公钥，用用户端公钥从 `key_manager` 中查询得到对应的服务端私钥，用服务端私钥对 `block` 签名。

## 五、 服务端接口设计

用户端通过 HTTP 请求与服务端交互，相关数据包含在 JSON 中。需注意，Python 的 `bytes` 字节数据不能直接用 JSON 传输，可以将字节数据用 `base64` 编码<sup>[15]</sup>为字符串，再放入 JSON 传输，接收端用 `base64` 解码以获得原数据。本部分中所说的数据类型，均指数据在 JSON 中的类型，所以，在前文提到的类型为 `bytes` 的数据，在本部分中的类型均为 `string`。这一点在下文不再作特别说明。



这里给出的服务端接口设计遵循 RESTful 接口设计规范<sup>[16]</sup>。

## （一） 密钥获取

获取独特的服务端密钥和独特的 AES 密钥。

URL: <http://127.0.0.1:8384/key>;

请求方式: POST;

参数:

参数名	必选	类型	说明
client_key	是	string	用户端 RSA 公钥

表 6 密钥获取接口参数

返回值:

参数名	类型	说明
enc_rsa_public_key	string	加密的服务端 RSA 公钥
enc_aes_key	string	加密的 AES 密钥
enc_rsa_public_key_signature	string	加密的服务端 RSA 公钥的签名
enc_aes_key_signature	string	加密的 AES 密钥的签名

表 7 密钥获取接口返回值

## （二） 任务上传

将计算任务（代码文件内容）上传到服务端。

URL: <http://127.0.0.1:8384/task>

请求方式: POST;

参数:

参数名	必选	类型	说明
client_key	是	string	用户端 RSA 公钥
enc_aes_key	是	string	加密的 AES 密钥
enc_task_hash	是	string	加密的任务散列值
enc_file_content	是	string	加密的文件内容
enc_aes_key_signature	是	string	加密的 AES 密钥的签名

enc_task_hash_signature	是	string	加密的任务散列值的签名
enc_file_content_signature	是	string	加密的文件内容的签名

表 8 任务上传接口参数

返回值：

参数名	类型	说明
task_arranged_status	string	任务安排的成功与否

表 9 任务上传接口返回值

### （三） 区块获取

根据任务散列值获取对应区块。

URL: <http://127.0.0.1:8384/block>

请求方式：GET

参数：

参数名	必选	类型	说明
task_hash	是	string	任务散列值

表 10 区块获取接口参数

返回值：

参数名	类型	说明
block	json	区块内容
signature	string	区块内容的签名

表 11 区块获取接口返回值

### （四） 全链获取

获取整个区块链的内容。

URL: <http://127.0.0.1:8384/block>

请求方式：GET

参数：无

返回值：

参数名	类型	说明
chain	json	全链内容

表 12 全链获取接口返回值

## 六、 样例实现

### （一） 语言与第三方库

样例使用 Python 3.7 实现。Python3.7 也是 Occlum 官方示例中使用的 Python 版本。

样例使用到的第三方库有：PyCryptodome 和 Flask<sup>[17]</sup>。

PyCryptodome 是一个包含低级密码学方法的 Python 库。本样例中使用了其中实现的 RSA 算法、AES 算法和 SHA256 算法，以及密钥的生成函数、随机字节的生成等功能；

Flask 是用 Python 编写的的轻量 Web 应用框架。本样例使用 Flask 来开发服务端接口。

### （二） 部署与运行

Occlum 官方给出了内置 Occlum 的 Ubuntu 18.04 系统的 Docker 镜像。本样例运行在 Occlum 官方的 Ubuntu 18.04 系统 Docker 镜像的实例中，并且该实例 8383 端口映射到 Docker 宿主机的 8384 端口。相当于样例平台通过宿主机 8384 端口对外服务。

## 七、 参考文献

- [1] Sabt M, Achemlal M, Bouabdallah A. Trusted execution environment: what it is, and what it is not[C]//2015 IEEE Trustcom/BigDataSE/ISPA. IEEE, 2015, 1: 57-64.
- [2] Yaga D, Mell P, Roby N, et al. Blockchain technology overview[J]. arXiv preprint arXiv:1906.11078, 2019.
- [3] Zheng Z, Xie S, Dai H N, et al. Blockchain challenges and opportunities: A survey[J]. International Journal of Web and Grid Services, 2018, 14(4): 352-375.
- [4] Rivest R L, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems[J]. Communications of the ACM, 1978, 21(2): 120-126.
- [5] Daemen J, Rijmen V. Reijndael: The Advanced Encryption Standard[J]. Dr. Dobb's Journal: Software Tools for the Professional Programmer, 2001, 26(3): 137-139.
- [6] 余凯, 贾磊, 陈雨强, 等. 深度学习的昨天, 今天和明天[J]. 计算机研究与发展, 2013, 50(9): 1799.
- [7] 吴吉义, 平玲娣, 潘雪增, 等. 云计算: 从概念到平台[J]. 电信科学, 2009, 25(12): 23-30.
- [8] Fielding R, Gettys J, Mogul J, et al. Hypertext transfer protocol–HTTP/1.1[J]. 1999.
- [9] McKeen F, Alexandrovich I, Anati I, et al. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave[M]//Proceedings of the Hardware and Architectural Support for Security and Privacy 2016. 2016: 1-9.
- [10] Shen Y, Tian H, Chen Y, et al. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx[C]//Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020: 955-970.
- [11] 沈鑫, 裴庆祺, 刘雪峰. 区块链技术综述[J]. 网络与信息安全学报, 2016, 2(11): 11-20.
- [12] Van Flymen D. Learn blockchains by building one[J]. The fastest way to learn how Blockchains work is to build one, 2017.
- [13] Calderbank M. The rsa cryptosystem: History, algorithm, primes[J]. Chicago: math. uchicago. edu, 2007.
- [14] Van Rossum G, Warsaw B, Coghlan N. PEP 8: style guide for Python code[J]. Python. org, 2001, 1565.

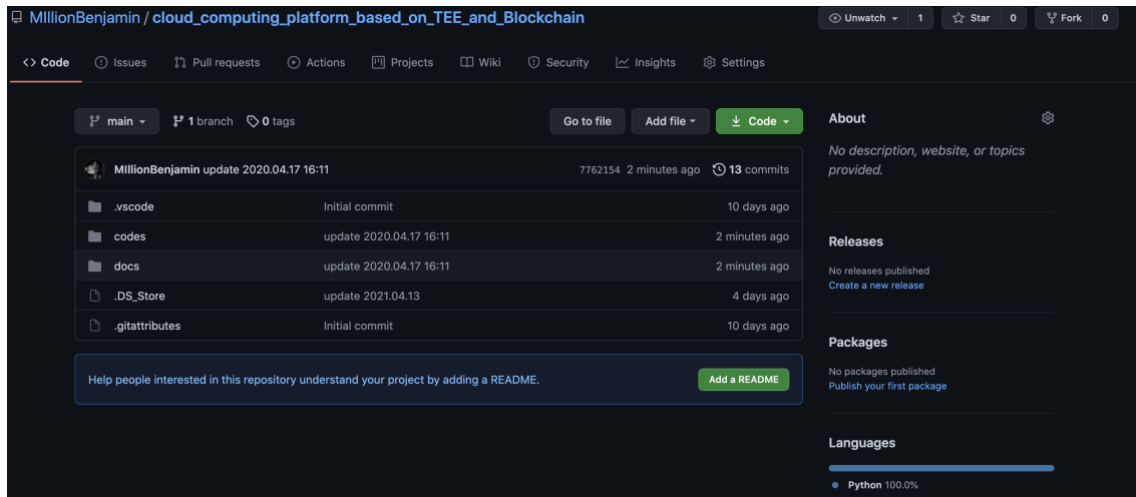
- [15] Josefsson S. The base16, base32, and base64 data encodings[R]. RFC 4648, October, 2006.
- [16] Richardson L, Ruby S. RESTful web services[M]. " O'Reilly Media, Inc.", 2008.
- [17] Grinberg M. Flask web development: developing web applications with python[M]. " O'Reilly Media, Inc.", 2018.

## 附录 A.

样例代码仓库 URL:

[https://github.com/MillionBenjamin/cloud\\_computing\\_platform\\_based\\_on\\_TEE\\_and\\_Blockchain](https://github.com/MillionBenjamin/cloud_computing_platform_based_on_TEE_and_Blockchain)

代码仓库截图:



## 八、 致谢

致谢应以简短的文字对课题研究与论文撰写过程中曾直接给予帮助的人员(例如指导教师、答疑教师及其他人员)表达自己的谢意，这不仅是一种礼貌，也是对他人劳动的尊重，是治学者应当遵循的学术规范。内容限一页。