利用区块链技术解决个人隐私保护的问题

#目前提出的SSI的思想支持用户将自己的隐私信息掌握在自己的手中，而不是信任一个或多个身份提供服务器。

摘要：

在当下，越来越多的网络用户被不断要求提供详细的身份信息以获得各种网络服务。许多的网站和APP在搜集到用户的真实信息后转手卖出了用户的个人信息，其中甚至有用户的身份证号，详细住址等信息。而用户的个人信息一旦被上传，就脱离了其个人的控制。隐私泄露的情况非常严重，甚至威胁到了用户的生命财产安全。为了使用户接管他自己身份（自我主权身份）的控制权，从而更好地保护自己的隐私信息，本文提出了一种新型的隐私保护框架，该框架结合了区块链技术、访问控制、数据加密、隐写术和分布式解密、门限密码学等多种技术，以实现对用户隐私数据的高效保护。我们的框架使用区块链作为访问控制的基础设施，并利用多方解密技术来增强数据的安全性。我们还设计了一种特殊的数据处理方法，使得请求者可以进行必要的身份验证或信息采集，而无需接触到用户的完整隐私数据。

1. 引言

随着互联网的发展，用户的隐私数据安全问题越来越引起人们的关注。在这个背景下，如何在保护用户隐私的同时，实现必要的数据交换和验证，成为了一个重要的研究问题。本文提出了一种新型的隐私保护框架，该框架的设计目标是在用户提供个人信息进行身份验证或数据采集时，保护其隐私数据的安全性和完整性。

许多用户在使用手机或浏览器时，总是要频繁地向不同的网页或APP输入自己的个人信息，有些甚至是隐私信息，如电话、住址、身份证、银行卡号。用户要获得某种服务或执行某种操作就需要提供自己的信息。然而这些信息一旦被上传至网络，就脱离了用户的控制。用户无法确定其流向和用途，当信息泄露时，用户甚至不知道是谁泄露了自己的信息。

以往的工作要防止信息泄露，主要是审查程序，如进行污点分析或跟踪。然而，污点分析存在假阳性和假阴性的问题，以及较大的性能开销导致其对用户敏感信息的保护是不够的。数据本身的流动应该有一种记账的机制，对于哪些程序或服务访问了用户敏感数据，都应该记录下来，同时防止抵赖。

区块链技术很好地提供了这样的记录机制，首先，区块链的不可变更性保证了记录的完整性，即一旦数据被写入区块链，就无法修改或删除。这意味着任何对用户敏感数据的访问操作都将被永久记录，确保数据的真实性和完整性。且每个区块都带有时间戳，这提供了一个明确的、按时间顺序的数据访问历史记录。其次，区块链上的交易需要网络中的多数节点达成共识后才能被添加。这确保了数据访问记录的真实性，因为单一的恶意节点无法独自修改记录。并且区块链数据是分布式存储的，这意味着没有单一的失败点，增加了数据的持久性和可用性。最后，区块链的链式结构使得每个区块都包含前一个区块的哈希值，形成一个连续的链。这确保了数据的完整性，因为任何对过去数据的尝试修改都会导致后续所有区块的哈希值不匹配。

然而，区块链这个技术用于个人隐私保护的场景仍然有许多要解决的问题。第一，区块链上的数据是透明的，可以被任何人查看，而用户的隐私数据则必须要通过加密等手段保护起来。第二，区块链往往采用智能合约来处理交易，且智能合约的代码是公开的。然而，在线上身份验证或身份采集的过程中，需要用到身份拥有者和身份请求者的私钥进行加解密的操作。而这些私钥是安全敏感的，不能存放在智能合约之中。第三，如果将数据存储在链下，而仅在链上存储哈希值或指针，那么外部存储也会存在安全风险。第四，如果在确认信息传递的双方之间通过安全信道直接传递隐私信息，则无法避免隐私信息被滥用，如果隐私信息被身份请求者泄露，则无法被跟踪和举证。

1. 背景和相关工作
2. 系统组成，风险模型，设计原则，设计目标
3. 我们提出的框架
   1. 概要

我们的框架使用区块链作为访问控制的基础设施，所有对用户个人信息的访问都必须通过区块链进行记录和追踪。当外部实体（如网站或应用）请求访问用户的隐私数据时，用户会向区块链提交一个包含了加密API地址的交易，供请求者调用。请求者解密API地址后，可以调用API来获取用户的隐私数据。为防止恶意攻击，API会设置短期令牌以确保访问的安全性。用户的隐私数据存储在一个加密的数据库中，API负责数据的解密、处理和再加密。处理后的数据将被分为三部分，每一部分都需要一个不同的参与方来进行解密。这样的设计旨在增强数据安全性，防止任何单个参与方能够完全解密数据。

* 1. 详细设计

第一阶段：用户注册，区块链创建

用户在本地创建一个私有的容器，在容器中创建一个数据库，将自己的隐私信息加密存储在该数据库中。创建的数据库作为卷来挂载，以方便其他本地容器访问这个数据库中的加密数据。加密使用的是用户自己的RSA公钥。而创建数据库的代码以及加密存储个人信息的代码和RSA公钥都封装在一个容器中。使用容器的目的是利用其隔离特性，保证一定的安全性，即不管容器运行在的环境是否安全，都不影响容器内程序的正常运行。

接着，用户申请创建一个自己的区块链（我们实现的区块链系统可以为每个用户保存他们自己的区块链）。这个区块链的初始区块包含了用户的公钥和签名。

第二阶段：用户请求外部服务

当用户请求某些需要验证身份的服务或执行某些特定的操作时，外部服务需要验证用户的身份才能允许下一步的操作。而有些身份验证需要用到多因素认证，过程中会搜集用户的敏感信息。而敏感信息一旦脱离了用户就失去控制，无法保证其不被泄露。利用区块链就是记录每一个对敏感信息访问的实体，以实现可追溯和防抵赖。

我们的设计中，外部服务通过双因素的方式来验证用户身份。首先，要求用户提供公钥和签名。服务提供者验证了签名后，会进一步要求用户提供更多的身份信息来确认用户的身份。这时，用户会提供自己的区块链服务器的访问地址和自己的区块链ID。服务提供者要想获得用户的隐私信息，则必须通过区块链来访问。

第三阶段：服务提供者请求隐私信息

服务提供者在两种情况下会需要用户的隐私信息，一种情况是进行信息的采集，一种情况是进行身份验证。这两种情况要区别对待，因为信息采集需要完整的信息，且出现的频率比较低，要更谨慎地授权。而身份验证则更普遍，只提供部分信息，也足以验证用户的身份，出现的频率更高，不进行控制很容易造成信息泄露。

因此，在我们的设计中，会区分身份信息的采集和身份信息的验证这两种操作。服务提供商访问区块链来创建新的交易。区块链上可以挂载的交易有两种，一种是服务提供商的请求类型交易，一种是用户的应答类型的交易。服务提供商在请求交易中要填写，自己的URL，代表身份的公钥和签名，以及请求用户隐私信息的原因和类型（身份采集或身份验证）。

第四阶段：区块链记录交易

区块链服务程序发现新的交易后，会创建新的区块，并基于PoW共识机制来挖矿，以将新的区块加入到区块链中去。多个区块链服务程序同步区块链，并在本地保存已验证的最长的区块链。

用户查看区块链已更新，就可以选择如何应答服务提供商的请求。如果接受请求，那么用户会将一个能够处理其个人信息的API的地址使用服务提供商的公钥加密，上传到区块链的下一个交易中，以供访问者调用。

区块链处理了用户的应答交易后，区块链变长。访问者查询到区块链新增了区块，并解密和调用其中的API地址，以获得用户信息。

第五阶段：用户提供加密信息

访问者解密API地址后访问特定网页，调用相应的API，这个网页设置了短期令牌，阻止攻击者的访问。

这个API是存放在另一个容器中的，它将访问在第一阶段存放在容器中的数据库，其中存放有用户加密的隐私数据，API将隐私数据解密为明文，并根据需要（对于身份验证则隐去10%的字符，而对于身份采集则不隐去字符）隐去其中的一小部分字符，生成图片，存放隐去字符后的明文，再用访问者的公钥，生成隐藏水印，添加到图片中。之后，用访问者的公钥加密图片。再将加密后的数据分为3个部分（最终的解密还需要额外三个参与方），随机生成三个参与方的顺序，并将加密顺序放在最终密文的头部，之后依次使用三方的公钥加密相应部分的数据，并计算数据被加密前的hash，将这些数据组合起来，用访问者的公钥加密。最后，发送给访问者。

这个过程是设计的核心部分。第一，使用短期令牌来访问特定网页，可以有效防止攻击者通过直接访问API地址来获取数据。短期令牌的有效期限制了攻击者的窗口期，提高了系统的整体安全性。第二，用户隐私数据存储在数据库中，并且以加密形式保存。API负责解密数据，但根据不同的需求进行字符的部分隐藏，以保护敏感信息，例如在身份验证情境下隐藏一部分字符。这种做法有助于减少敏感数据暴露的风险。第三，通过将用户数据加上隐藏水印并嵌入图片中，可以追踪数据的来源。这在需要追溯数据泄漏或滥用时非常有用，同时也增加了数据的完整性。第四，通过使用访问者的公钥多次加密数据，可以确保数据只能被访问者解密。多重加密层次增加了数据的保密性，即使有人截获了数据，也难以解密其中的内容。第五，将数据分为三部分，并使用三个不同参与方的公钥进行加密，然后按随机顺序排列，进一步增加了数据的安全性。这种做法需要攻击者同时获取三个不同参与方的私钥才能解密数据。第六，将加密顺序放在最终密文的头部，并且随机生成三个参与方的顺序，增加了攻击者破解顺序的难度。第七，计算数据被加密前的哈希值并将其包含在最终加密数据中，可以确保数据在传输过程中没有被篡改。如果数据在传输过程中发生任何改动，哈希值将会不匹配，提示数据的篡改。

第六阶段：隐私信息请求者解密信息

访问者接收到加密数据后寻找三个参与方按照顺序依次解密，最终获得明文图片，这里的明文隐藏了一部分信息，但是，如果是身份验证，则足以确认用户的身份，因为明文图片中还包含用户完整信息的hash，访问者可以比对自己已有信息进行验证，如果是信息采集，则没有隐去信息，访问者可以获得完整信息并计算hash来确认信息的完整性。

1. 实现

我们的代码主要是Python语言的。容器部分的代码是运行在Docker Desktop上的。因为主机是Windows环境，所以需要安装WSL2内核。

另外，我们重新实现了一个简单的区块链用于存储所有对用户隐私信息的访问操作。我们利用了Python中的flask框架来将对区块链的访问转为对特定网页的GET或POST请求，从而方便用户调用区块链中的方法，实现安全保护。例如，当用户要创建自己的区块链时，他可以访问<https://blockchain_servern_ip/blockchain/new>这个网址。这个网页在搜集到足够的信息后会传到后台服务器来处理信息，服务器最终调用函数blockchain.create\_genesis\_block()来创建创世区块。当创建成功之后，服务器会返回blockchain\_id代表区块链创建成功。

我们提供了3个区块链服务程序分别在不同的端口等待用户的请求。当创建区块链的请求到达时，3个服务程序会同步更新自己存储的区块链的信息，在完成更新后，将区块链ID返回给用户。

所有创建的区块链都将保存在磁盘，每次区块链服务程序启动时，将从磁盘加载已创建的区块链。

增加了存储到磁盘的操作后，create\_chain的时间略有延长从Average time to create a new blockchain: 6.12 seconds到，Average time to create a new blockchain: 6.16 seconds。

增加了transaction之后，后台自动挖掘，在下一轮get chain时已能够显示出来。

容器1中创建数据库和写数据库的时间。

2023-08-27 18:33:05 Create Database time: 0.028538227081298828 seconds

2023-08-27 18:33:05 Encrypt and store time: 0.005376100540161133 seconds

1. 性能评估
2. 安全性分析
3. 讨论
4. 总结

附录

在讨论容器中的数据库安全性时，您提到了两种主要的设计方法：1) 将数据库放在容器内，同时使用Docker网络进行通信，以及2) 使用Docker卷挂载数据库。实际上，这两种方法并不是完全相互排斥的，并且经常一起使用以增强安全性和可持久性。

首先，让我们分析这两种方法的特点：

1. \*\*将数据库放在容器内，并使用Docker网络进行通信\*\*：

- \*\*优势\*\*：

\* \*\*隔离性\*\*：每个容器运行在自己的环境中，与其他容器隔离。使用Docker网络可以进一步限制容器之间的通信。

\* \*\*灵活性\*\*：您可以轻松移动和复制整个容器，包括其数据。

- \*\*劣势\*\*：

\* \*\*持久性\*\*：如果容器被删除，数据可能会丢失，除非您已经设置了卷或其他持久存储解决方案。

2. \*\*使用Docker卷挂载数据库\*\*：

- \*\*优势\*\*：

\* \*\*持久性\*\*：Docker卷提供了一个持久的存储解决方案，这意味着即使容器被删除，数据仍然存在并可以被重新挂载。

\* \*\*性能\*\*：直接访问宿主机的文件系统通常比在容器的叠加文件系统上操作更快。

\* \*\*灵活性\*\*：您可以轻松备份、迁移或共享Docker卷。

- \*\*劣势\*\*：

\* \*\*安全性\*\*：任何可以访问宿主机的文件系统的实体都可以访问Docker卷，因此需要确保适当的文件和目录权限。

当我们谈论"更安全"时，这取决于您的具体安全需求。但在大多数情况下，结合这两种方法是比较理想的：

1. \*\*使用Docker网络来隔离数据库容器\*\*，以减少不必要的网络访问和潜在的攻击面。

2. \*\*使用Docker卷来持久化数据库数据\*\*，并确保宿主机上的卷权限正确设置，以避免未经授权的访问。

通过这种方式，您可以从每种方法中获得最大的好处，同时最大限度地增强数据的安全性和可持久性。

实现上述方案需要多个步骤。以下是大致的步骤和相关的Docker命令：

1. \*\*创建Docker网络\*\*：

我们首先创建一个Docker网络以确保数据库和应用容器之间的通信安全。

```bash

docker network create my\_network

```

2. \*\*创建Docker卷\*\*：

接下来，我们创建一个Docker卷以持久化数据库数据。

```bash

docker volume create db\_data

```

3. \*\*运行数据库容器\*\*：

假设我们要运行一个PostgreSQL数据库容器，我们可以使用以下命令：

```bash

docker run --name db\_container --network my\_network -v db\_data:/var/lib/postgresql/data -e POSTGRES\_PASSWORD=mysecretpassword -d postgres

```

这里，我们使用`-v`标志将我们之前创建的Docker卷挂载到容器中PostgreSQL默认存储其数据的路径。

4. \*\*运行应用容器\*\*：

假设您的应用需要连接到上述数据库，并且您已经有了一个Docker镜像（例如`my\_app\_image`）为您的应用：

```bash

docker run --name app\_container --network my\_network -p 8080:80 -d my\_app\_image

```

在这里，我们使用`-p`标志将应用容器的80端口映射到宿主机的8080端口，这样外部用户就可以通过宿主机的8080端口访问您的应用。

5. \*\*应用连接到数据库\*\*：

当应用需要连接到数据库时，它应使用`db\_container`作为数据库的hostname。因为两个容器都在同一个Docker网络中，所以它们可以通过容器名相互解析。

注意：

- 这只是一个基本的例子。在实际生产环境中，您可能需要更多的配置和安全措施。

- 您需要确保应用程序的Docker镜像和配置允许它使用容器名（如`db\_container`）作为数据库的主机名。

- 这个例子使用了PostgreSQL数据库容器，但您可以使用任何其他数据库容器，只需相应地调整命令和路径。

如果您想让一个容器可以向数据库写数据，但不希望该容器被外部访问，您可以采用以下策略：

1. \*\*不暴露容器的任何端口\*\*：当您运行容器时，不要使用`-p`或`--publish`标志。这意味着容器内的应用程序可以运行，但外部系统无法直接访问该容器的任何端口。

2. \*\*使用Docker网络\*\*：创建一个自定义的Docker网络，并确保您的数据库容器和该写入容器都连接到此网络。这样，这两个容器可以互相通信，但不会与外部系统通信，除非您明确地发布了端口。

下面是如何实现这一策略的步骤：

1. 创建一个自定义的Docker网络：

```bash

docker network create my-custom-network

```

2. 运行数据库容器，并连接到自定义网络：

```bash

docker run --name my-database-container --network my-custom-network -e POSTGRES\_PASSWORD=mysecretpassword -d postgres

```

3. 运行写入容器，不暴露任何端口，并连接到自定义网络。在这个容器中，您可以使用数据库容器的名字（在这个例子中是`my-database-container`）作为数据库的主机名，从而连接到数据库：

```bash

docker run --name my-write-container --network my-custom-network my-write-image

```

在`my-write-image`中的应用程序可以使用`my-database-container`作为数据库主机名来连接和写入数据。

这样，`my-write-container`就无法从外部访问，但它可以在Docker网络内部与`my-database-container`通信。

在此实现中，用户首先需要通过 /token 端点获取一个短时令牌。然后，他们可以在之后的 API 请求的 Authorization 头中使用此令牌。这样，只有持有有效令牌的用户才能访问 API。

短时令牌（也称为临时令牌或过期令牌）是为了提供一种短暂的、有时效的身份验证方法。在上述代码中，短时令牌的作用主要有以下几点：

身份验证：令牌用作身份验证方法，确保请求的用户或系统之前已经获得了有效的令牌，从而证明它们有权访问 API。

增强安全性：由于令牌有时间限制（在上述代码中设置为600秒或10分钟），即使令牌被拦截或泄露，攻击者只有很短的时间可以使用这个令牌。一旦令牌过期，它就无法再用于身份验证。

减少持久性凭证的使用：短时令牌可以减少长期有效凭证（如密码或长期 API 密钥）的使用。用户只需使用长期凭证获取令牌一次，然后可以在令牌有效期内使用该令牌进行身份验证，而无需再次使用长期凭证。

防止重放攻击：由于令牌有时效性，攻击者不能重复使用捕获的旧令牌进行重放攻击。

简化会话管理：短时令牌提供了一种无状态的会话管理方式。服务器不需要保存活动会话的状态，因为每个请求都带有一个令牌，服务器可以通过验证令牌来确认请求的有效性。

docker run --name db\_container --network my\_network -v db\_data:/var/lib/postgresql/data -e POSTGRES\_PASSWORD=mysecretpassword -d postgres

PS D:\区块链\保护个人用户的隐私\app1> docker run --network my\_network my-write-container

Write database time: 0.0517268180847168 seconds

Write database time: 0.11095404624938965 seconds