



**学士 学位 论文**

***Shandong University* Bachelor’s Thesis**

论文题目：基于区块链的PKI私有链实现

|  |  |
| --- | --- |
| 学院 | 数学学院 |
| 专业 | 信息安全专业 |
| 年级 | 2014级 |
| 姓名 | 侯柏韬 |
| 学号 | 201400090110 |
| 指导老师 | 王美琴 |

**2018年5月12日**

目录

[摘要 3](#_Toc513919351)

[关键词 3](#_Toc513919352)

[Abstract 3](#_Toc513919353)

[前言 4](#_Toc513919354)

[结构及安全性介绍 4](#_Toc513919355)

[1． 区块链基本概念及安全机理 4](#_Toc513919356)

[1.1 区块链基本概念 5](#_Toc513919357)

[1.2 区块链安全机理 5](#_Toc513919358)

[Hash算法 5](#_Toc513919359)

[共识过程 6](#_Toc513919360)

[工作量证明（Proof of Work） 一 6](#_Toc513919361)

[2． PKI的基本概念及核心功能 6](#_Toc513919362)

[2.1 PKI基本构成 7](#_Toc513919363)

[2.2 PKI的安全性依赖 8](#_Toc513919364)

[基于区块链的PKI系统 9](#_Toc513919365)

[1. 体系介绍 9](#_Toc513919366)

[注册操作 9](#_Toc513919367)

[更新操作 9](#_Toc513919368)

[注销操作 10](#_Toc513919369)

[查询操作 10](#_Toc513919370)

[2. 结构对比 10](#_Toc513919371)

[3. 改进建议 11](#_Toc513919372)

[3.1 空间占用问题 11](#_Toc513919373)

[设置过程 12](#_Toc513919374)

[检索方式 12](#_Toc513919375)

[3.2查询效率问题 13](#_Toc513919376)

[应用 13](#_Toc513919377)

[结语 14](#_Toc513919378)

[参考文献： 15](#_Toc513919379)

[附件 16](#_Toc513919380)

[PKI智能合约运行代码 16](#_Toc513919381)

摘要

PKI是现代电子通讯在非对称私钥加密通讯环境下所必须的一种基本通讯设施，为通讯双方的公钥查询提供了安全的认证保障。本文通过区块链技术构建PKI，从理论和实际应用两个方面解决了现今通讯环境中PKI易受攻击和伪造的问题，并在私有链环境下搭建了可满足基本功能的PKI合约，与现有的PKI对比，探索了真实使用中该PKI可能出现的问题及改进意见。

关键词：区块链；公钥基础设施；以太坊；智能合约

Abstract

PKI is a basic communication facility for modern electronic communications under asymmetrical encrypted communication environment. It provides secure authentication for public key queries of both parties. This paper constructs PKI through blockchain technology, solves the problem of PKI vulnerable to attack and forgery in the current communication environment from both theoretical and practical applications, and builds a PKI contract that can meet basic functions in the private chain environment. Some PKI comparisons have explored possible problems and improvements in PKI in real use.

**Keywords:** block chain; PKI; Ethereum; Smart contract

# 前言

区块链是一种以去中心化的分布式计算框架为技术核心的数据库展示方案，其独特的去中心化、不可修改、公众维护、集体安全等特性，使得它在现实生活中的应用愈加广泛[1]。而一个安全的PKI系统(Public Key Infrastructure)，恰恰需要区块链的这些卓越特性，来保证其用户在公钥的存储、更新、获取等操作安全可信。因此，本文详细介绍了在区块链下构建的PKI模型，分析了其理论上与传统模型在运作和安全性的区别，并探讨了改进意见；随后通过以太坊GETH环境下的私有链，搭建一个可以满足基本PKI要求的操作过程，为区块链下的PKI用户化实现提供了参考。

# 结构及安全性介绍

简单来说，基于区块链的PKI系统就是一个建立在区块链上的公钥基础设施，它需要具有PKI体系对公钥一系列操作的功能，同时又需要通过区块链来消除传统PKI中心化的问题并保证其公钥的安全性。故在这一章节，我们将在章节1中介绍传统区块链的构成方式和运行过程中的安全保障；在章节2中介绍传统PKI的基本概念和核心功能。

## **1． 区块链基本概念及安全机理**

区块链在近年来作为比特币等数字模拟货币的技术支撑，成为了研究的热点，但去除经济等人为因素，其作为分布式交互环境的一个新型结构，在我们的生活中正扮演着重要的作用，尤其在金融领域，区块链以去中心化记账的代表而应用广泛[2]。

### 1.1 区块链基本概念

顾名思义，区块链的基础就是两个方面：区域区块化和链式结构。区块是指账本中的交易数据通过区块的形式被记录下来，全系统共享并且共同进行维护；而链式结构中，不同区块由时间戳进行前后的区分，各个区块之间又通过链式结构进行连接。很显然，区块链是以分布式记账系统的形式存在的。但与此同时，就会存在分布式网络应用的典型问题：共识问题和拜占庭将军问题。而区块链的出现可以认为是以在2008年中本聪发表的论文《Bitcion: A Peer-to-Peer Electronic Cash System》[4]为代表，其中关于比特币的构想，解决了区块链应用的难题。遵循比特币虚拟货币、分布式区块链的基本思想，现如今区块链已经有很多变种，其中最为著名的就是开源以太坊[5]，以太坊是2013年Vitalik Buterin提出的分布式应用平台，其应用十分广泛，下文也将以以太坊为代表描述其安全机理。

### 1.2 区块链安全机理

区块链在构成的过程中，需要通过hash函数对每个区块内的数据进行检验保证其完整性，并且每个区块通过添加上一个区块的hash值来进行链接，使之成为一个链式结构。与此同时，hash函数的抗碰撞性也使得修改已入链的区块信息是多项式不可能的[6]。而区块链接到区块链主链的过程中，每个区块的时间戳和共识算法使得同时出现两个支链的情况可以被避免，从而保证所有节点接受的信息是一致的。当区块链接到主链上后，区块链的分布式记账结构使得每一个参与节点都保存有该主链的全部信息，就算敌手通过了hash碰撞检验，妄图修改链中已有信息也需要克服工作量证明而推翻系统一半以上的算力。在运行中的大型区块链中，这是近乎不可能的事情。其安全机制的具体参照如下：

Hash算法 hash是密码学算法的重要部分，它的作用是将任意长度的明文通过单向函数转化成固定字长，方便进行文本处理并提供安全的MAC。区块链技术大范围采用了Hash算法，作为密码加密的预处理和区块之间的链接、同时也给Mapping函数的DHT（distinction hash table）提供了简便的查找方式。区块链中信息在上传的过程中，会在所在区块生成该信息的Hash函数，同时在每个区块中含有上个区块的Hash值，也计算入该区块的Hash过程。系统中的每个用户都可以在任意时段对开放的区块信息进行Hash检测，Hash算法的抗碰撞性完美的保障了区块链信息的完整性和链式结构的不可否认性。我们用到的以太坊GETH命令环境采用的是SHA-3-256算法，输出为32字节，安全指数为168.

共识过程 [安全性研究]以太坊的共识过程是通过对备选链条的既有工作量总额进行筛选完成的，对于短时间内出现的不同分叉同时指向前一个区块，各节点总是选择已有工作量证明最大的链作为主链，其他的分链将会被抛弃。我们知道，区块链的运行依靠着数字模拟货币，而模拟货币的重要问题是重复支付问题，需要避免货币拥有者使用同一笔钱进行两次支付。区块链提供的共识算法一方面通过时间戳进行排序，另一方面通过共识算法选择累积工作量最大的链条（一般比特币中是需要通过6个区块周期确定主链），从而避免出现两笔交易同时发生的状况。

工作量证明（Proof of Work） 一个交易在被整个系统承认之前需要完成检验，而工作量证明提供的就是这么一种验证过程。以太坊的工作量证明是通过大量的计算去寻找一个动态的区块hash值完成的，这个计算的复杂度由当前链中工作节点的数量和hash值的难度要求决定，使得完成一个区块的验证能够维持在15s。而工作量证明的完成是依靠挖矿，通过计算资源完成一个包含有效交易的区块创建的矿工拥有区块的记账权。POW的存在使得伪造一个可通过检验的区块变得十分困难，同时共识算法也使得推翻既有节点要求伪造者拥有总结点至少一半的算力，从而解决了拜占庭将军问题，即分布系统下各节点的信任问题。

## 2． PKI的基本概念及核心功能

PKI是公钥基础设施（Public Key Infrastructure）的简称，传统PKI是由认证中心将用户的公钥与用户认证信息结合认证中心的签名，构成数字认证证书，并以证书的方式提供用户公钥的认证服务。PKI的通信方式采用标准C/S模式，沟通非对称签名标准并颁发证书，在这个过程中，认证中心的权威性是采纳用户公钥的唯一可信源。

### 2.1 PKI基本构成

构成一个完整的PKI系统，需要包括以下五个主要方面：

1．认证机构，可信的证书签发机构；

2. 证书库，用于存放可供公众查询的、其他经过认证机构认证并签名的证书；

3. 密钥恢复和备份系统，对用户的公钥密钥进行备份，并提供恢复服务；

4. 证书撤销处理系统，对于作废的证书及时实现终止使用，撤销证书的CRL列表；

5. PKI应用接口，保证与PKI构成交互的网络连接安全可靠。

而上述的PKI构成要素同时也包含着PKI需要提供给用户的服务。在PKI系统中每个用户u拥有自己的密钥对keypair（sk,pk），该密钥对可以实现非对称加密过程，算法如下[4]：

* ,给定安全参数，生成密钥对。
* ，使用密钥对完成签名操作，得到。
* ，使用对进行验证操作，验证成功返回结果1，不然返回结果0.

PKI最基本的工作就是要记录用户提供的，并在其他用户查询该时返回该值。因此PKI所提供的服务在用户方面是：用户注册其、核对通过后颁发认证中心的证书CA、用户可以更新其提交的、用户可注销已提交的pk；而在使用用户方面是：给定用户可查询对应的。

其实现过程为

* ，用户提交可用，认证中心验证该的正确性后颁发证书CA，证书CA是认证中心通过自己私钥对用户和该用户相关的证书信息的签名；
* ，使用者向认证中心发送对的查询请求，认证中心返回该的证书CA；
* ，使用者得到证书CA，通过已知的认证中心公钥对签名进行认证得到所求的；
* ，用户提交更新请求，并告知旧和新，认证中心核对信息通过后，颁发新证书，并更新自己存储的CA；
* ，用户提交注销请求，认证中心核对信息后，通过核对后，如注销成功则返回1，否则返回信息0.

### 2.2 PKI的安全性依赖

PKI系统在使用过程中的安全性完全依赖于安全的非对称签名算法，非对称签名算法的安全性直接决定了包括用户使用过程的私钥安全和证书CA是否可被伪造的安全性。而对于PKI体系的安全性，依托于认证中心的层次模型。各级CA认证机构构成了层次模型中的信任链，彼此之间的链接通过上级向下级颁发证书来实现。“这种认证体系将证书分成，个证书都有上级CA的数字签名，当用户查看对方证书是否可信时，首先看其上级CA的签名，上级CA真实可信，则继续查看其对应的而上级CA的公钥来解密其数字签名，以此类推，一直查到证书颁发者的名字与证书的主体名字相同时为止，也就是查到根证书与自签名证书为止”[3]。

但是这样的层次结构会引发出这样的问题:根证书或自签名证书的安全性如何保障？在传统PKI体系中，我们不得不去相信至少一个第三方机构的可信度，在上述的层次模型中，我们至少要相信根证书的颁发机构，这是一切PKI体系运转的前提。当然这就引发了一系列质疑和一直困扰我们的隐患：如何使人相信证书颁发机构？证书颁发机构如何规避受攻击风险？证书颁发机构受到攻击时该如何保障PKI 的正常运转？也正因此，我们提出了依靠区块链的去中心化结构来实现PKI的运作，使得PKI体系中必须信任某一第三方机构这一问题得到较好的解决。

# 基于区块链的PKI系统

## 1. 体系介绍

基于区块链的PKI系统构想为通过区块链技术，将某一固定范围区域链声明PKI：节点用户发布的以及一些相关信息经验证后以区块的方式在区块链上公布作为公钥的注册操作；其后，通过验证的发布者可以通过提交新的公钥和注销在链中新建区块覆盖原有信息；节点内任意用户均可对链中已公布进行查询操作。

注册操作 注册拥有者向PKI提交命名信息为name的注册信息，PKI通过验证后将该消息公布在自己所拥有的区块链中

* ，

给定，注册拥有者生成并发送注册信息，为验证签名，通过公布的公钥所对应的私钥对对的签名来实现；

* ，

PKI接受验证信息，首先查询链中是否具有可用且相同的信息，没有则对进行签名验证，验证通过则公布该的及相关信息；

更新操作 任何用户可以向PKI提交已存在的命名信息为的更新消息，PKI通过对旧签名的验证来确认更新信息是否可信，验证通过即在所拥有区块链中公布更新消息并对该进行更新标记

* ，

给定，用户生成更新信息，为验证签名，通过公布的公钥对应的私钥对对和对的签名来实现；其中公钥类型pkf比pkn具有更高的更新权限；

* ，

PKI接受验证信息，首先查询链中是否具有可用且相同的信息，有则对进行签名验证，验证通过则公布该更新后的及相关信息；

注销操作 任何用户可以向PKI提交已存在的命名信息为的注销消息，PKI通过对签名的验证来确认注销信息是否可信，验证通过即在所拥有区块链中公布更新消息并对该进行注销操作并标记

* ，

给定，用户生成更新信息，为验证签名，通过公布的公钥对应的私钥对对和对的签名来实现；

* ，

PKI接受验证信息，首先查询链中是否具有可用且相同的信息，有则对进行签名验证，验证通过则返回并注销的及相关信息；

查询操作 任何用户可向PKI提交命名信息为name的查询信息，PKI收到请求后将在自己所拥有的区块链中查询对应pk并返回

* ，

给定，使用者生成并发送查询信息，为请求的公钥类型；

* ，

PKI接受查询请求，首先查询链中是否具有可用的信息，有则返回对应对的值；

## 2. 结构对比

较之传统PKI系统，区块链PKI具有着天然的优势。首先，区块链透明化的分布式记账规则使得用户公布在区块链上的公钥可以被公共清楚地获取，的注册、更新、注销等消息也均会详细地记录在区块链MK树上，可供查询，针对PK信息的每一次更新和注销等操作也是对每一个参与节点开放可查的；其次，区块链由工作量证明和Hash算法所提供的天然的不可更改性，使得用户公布的 很难被攻击者篡改，有效的抵抗了存在性攻击保证了的可信度；最后，区块链的分布式规则，完美取消了公众对传统PKI系统中认证中心容易受到攻击的顾虑，因为区块链中的信息是同步在区块链的每一个节点中的，攻击者的每一个可行的攻击都可以在被修改后被系统推选出的公共节点信息恢复，何况，攻击者的攻击难度是被扩大到了系统内至少一半节点容量的规模大小，而这一情况在polynomial条件下是不可能存在。

但同时区块链环境下的PKI也会存在一些问题。最明显的就是区块链节点信息存储量问题：区块链解决了取消中心节点的难题，但代价是系统中每个或大部分节点都需要保存公共链中的全部数据，不仅占据了大量的空间资源，还增加了节点对链内信息搜查的难度；其次是区块链的不可更改性：相较于传统数据存储库对数据可以进行增加、删除、修改、查询四种操作，但区块链的数据操作因为区块链信息一旦公布就无法修改的特性而没有修改和删除操作，故对区块链PKI中的PK数据修改操作只有更新和注销两种，更新是通过再次声明的方式对原有数据进行覆盖，因此查询过程就要求用户需要通过时间戳对每一个声明区块进行查找操作，无疑再次增加了对信息查询的难度。

## 3. 改进建议

### 3.1 空间占用问题

对于空间占用大的问题，我们采用P2P(Peer-to-Peer)技术解决此问题，借助P2P节点的经典Chord环算法，我们提出了区域性多级节点RML(Regional Multi-level Technology) 构想完成节点存储：

对于区块链上的每个节点，理论上是要求包含有整个区块链的全部区块信息

1

二级节点

n-1

一级环

2

3

n-2

一级节点

4

lelnode2

三级环

二级环

lelnode1

图表 1：RML存储结构

才能参与区块链的操作，随着区块链长度的不断增长，无疑会不断地增加节点的存储空间。因此我们将总节点中的某些节点在逻辑上看做一个独立的区域，设置其为二级节点环roll2，如图所示。

#### 设置过程

* 参与计算的区块链节点为，节点按通讯区域划分为若干组，每组数量n满足条件，选出组中存储空间最大、算力最高的节点作为该组与区块链的链接节点（一级节点），该节点同步存储区块链中的全部信息；
* 分组中个节点形成逻辑上的放射环roll2结构，除中心节点以外，其余个节点成为彼此之间保持环结构通讯的二级节点，节点为，并与中心节点保持放射型通讯；
* 节点按照经典Chord环结构进行主副节点划分，其中个节点成为，并且这些主节点只需存储的节点信息，每个主节点和其含有的副节点按步骤2继续划分；
* 环rollx中的主节点划分为2个时，划分结束。

#### 检索方式

* 源节点a先查询自己存储的数据中是否含有目标信息，有则查询结束返回结果；没有则向后续主节点发布迭代查询请求；
* 节点接到查询请求，查询自己存储数据中是否含有目标信息，有则查询结束向源节点a返回结果；没有则探勘自己是否位于环中最后一位，是则向上节点a发布迭代查询请求；不是则重复步骤1；
* 源节点a收到查询失败回应则返回查询失败；

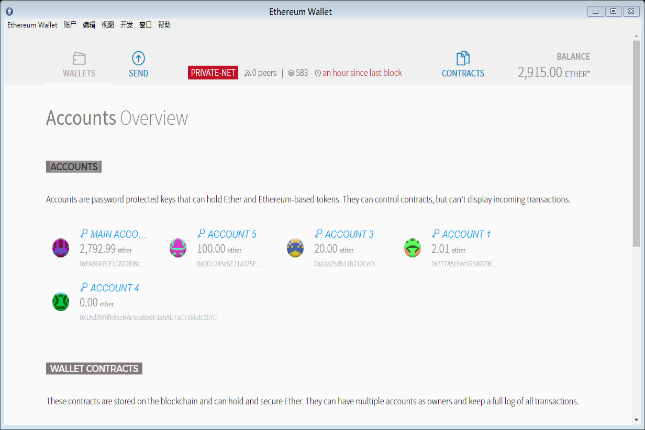
RML算法使得每个节点存储信息， 为全链信息；对于第x级环的主节点来说，其存储空间,远小于。

### 3.2查询效率问题

对于解决节点信息的查询效率问题，我们通过散列表DHT（distributed hash table）技术进行解决。散列表是一种用于存储映射和集合的数据结构，其存储和检索数据可以在时间复杂度内完成，一个强健的的分布式散列表，可以有效地完成许多分布式算法的需求。在区块链中，每一个区块都有自己在该区块中独一无二的地址(address)编号。以太坊中的地址分为两种：用户地址和合约地址，但是地址编号和区块在链中位置并无必然关系，每次查询需要逆MK树进行逆向查询，在总区块数为的规模下，查询时间复杂度为 []。但我们通过在节点中建立一个随pk信息增长而变化的列表的方式，以可用pk的用户的name或id信息为键值构建映射，组成散列表，使该列表存储区块的位置信息。这样可将查询的时间复杂度降低为，极大提升查询效率。

# 应用

本文通过以太坊私有链采用solidity语言部署具有PKI交互功能的智能合约。其中以太坊（Ethereum）是第一个也是目前全球最活跃的区块链2.0 公有链，在区块链体系中极具代表性，它是有2013年末Vitalik Buterin提出的一个开源的有智能合约功能的公共区块链平台，通过专用加密货币以太币支持以太坊内的交互和信息处理工作。其采用的智能合约由solidity语言完成合约编程过程，进而由EVM（Ethereum Virtual Machine）语言完成编译。合约部署完成后，交易方的交互均是在GETH完成。而区块链的私有链是一条个人搭建



图表 2

的、不对外公开的区块链，其构成原理与公有链是一致的，但具有只有拥有者

具有写入权限、高度隐私的优点，方便我们对PKI 的性能进行相关测试。

如pic2显示了我们的私有链的一些基本信息，可以看到该私有链结构完整、功能完善，共5个可用的用户节点并已分配好以太币以用于交易，且已对示例代码进行试验，显示运行通畅。

我们将已经编译好的合约部署在私有链中，已经签名检验成功的用户向私有链PKI地址提交拥有的pk，依次完成注册、更改操作，并使用用户地址向PKI发起查询操作，最后对pk进行注销。其演示结果如pic3所示。可以看到各项操作均成功，程序运行流畅。

# 结语

综上所述，区块链环境下搭建PKI是一种可行的尝试，它给我们提供了全新的保存和获取用户公钥的方式。相较于现有的PKI体系，它有力地解决了现有体系中用户必须信任某一第三方中心的窘境，并有效降低了用户pk信息受攻击的风险，具有极强的鲁棒性，能抵抗存在性伪造和信息篡改；同时上述实验表明，它在现实中的应用是可行的，且当系统节点足够大时，对误操和攻击具有一定的韧性。在区块链技术正蓬勃发展的现在，区块链环境下的PKI的出现，将会是该技术在电子非对称通讯网络上的又一次重要尝试。

# 参考文献：

[1] 何蒲, 于戈, 张岩峰, 鲍玉斌. 区块链技术与应用前瞻综述[J]. COMPUTER SCIENCE, 2017, 44(4): 1- 8

[2] 刘德林.区块链智能合约技术在金融领域的研发应用现状、 问题及建议[J]. 海南经济, 2016, 335(10): 26-31

[3] Stiglitz Joseph E，Weiss Andrew．Credit Rationing in Market with Imperfect Information [J]．The America Economic Review，1981(3)

[4] NAil<AMOTO S Bitcoin：a peer-to-peer electronic cash system[EB/OI ]．https://bitcoin．org/bitcoim pdf

[5] Ethereum. Ethereum homestead document[EB/OL]. 2018.3[2018.5.12]. http://ethdoc.cn/.

[6] Carter J, Wegman M. Universal classes of hash functions (Extended Abstract) [J]. Acm Symposium on Theory of Computing , 1977, 18(2):106-112.CCarter

[7] HANCOCK M，VAIZEY E.Distributed ledger technology：beyond block chain[R].UK：Government Office for Science，2016

[8] 冯登国. Cryptography: Theory and Practice[M]. WaterLoo:Douglas R.Stinson, 2003.

[9] 谢冬青. PKI原理与技术[M]. second sersion. 清华大学:清华大学出版社, 2004 :116-127.

[10] Fromknecht C, Velicanu D, Yakoubov S. A Decentralized Public Key Infrastructure with Identity Retention[D]. MIT, 2014.

[11] Wytse, Oortwijn. A Distributed Hash Table for Shared Memory[D]. University of Twente:Wytse Oortwijn(B), Tom van Dijk, and Jaco van de Pol, 2017. 1-10

[12] Antony Rowstron1 and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems[D]. Rice University MS-132, 6100 Main Street, Houston, TX 77005-1892, USA.:Rice University, 2001.

[13] Zhuo C, Gang F, Yi L, etal. Improving Playback Quality of Peer——to--Peer Live Streaming Systems by Joint Scheduling and Distributed Hash Table Based Compensation[D]. ,Atlanta 30332，USA:College ofComputer Science and Engineering，Chongqing University ofTechnology, 2004.

[14] Kosba，Ahmed E．，Miller，Andrew，Shi，Elaine，Wen，Zikai and Papamanthou．Charalampos．The blockchain model of cryptography and privacy-preserving smart contracts[J】．IACR Cryptology ePrint Archive，2015：67

[15] Ethereum. Solidity Documentation[M]. Release 0.4.23. Ethereum :Ethereum , 2018 :16-235.

# 附件

## PKI智能合约运行代码

//PKI BUILDING

contract PKI **{**

// name owner

struct Owner **{**

int id**;**

bytes32 name**;**

**}**

struct Identity **{**

address \_owner**;**

int id**;**

bytes32 name**;**

bool regitered**;**

bool updated**;**

bool revoked**;**

bytes20 pkn**;**

bytes20 pkf**;**

**}**

address public pki**;**

address public user**;**

mapping **(**address **=>** Owner**)** public owner**;**

Identity**[]** public identitys**;**

event PkRegister**(**int id**,**bytes32 name**,**bytes20 pkn**,**bytes20 pkf**);**

event PkUpdate**(**int id**,**bytes32 name**,**bytes20 pko**,**bytes20 pkw**);**

event PkRevoke**(**int id**,**bytes32 name**,**bytes20 pkn**,**bytes20 pkf**);**

//signation

//input ‘web3.eth.sign(key, m)’.in console for singation m

//sha3(...) returns (bytes32)

// ecrecover（hash，r， s， v）return address in 20bytes；

//signed m is in hex type

function decode**(**bytes m**,** bytes memory signedString**)** public returns **(**address**)** **{**

bytes32 r **=** bytesToBytes32**(**slice**(**signedString**,** 0**,** 32**));**

bytes32 s **=** bytesToBytes32**(**slice**(**signedString**,** 32**,** 32**));**

byte v **=** slice**(**signedString**,** 64**,** 1**)[**0**];**

**return** ecrecoverDecode**(**m**,** r**,** s**,** v**);**

**}**

function slice**(**bytes memory data**,** int start**,** int len**)** returns **(**bytes**){**

bytes memory b **=** **new** bytes**(**len**);**

**for(**int i **=** 0**;** i **<** len**;** i**++){**

b**[**i**]** **=** data**[**i **+** start**];**

**}**

**return** b**;**

**}**

function ecrecoverDecode**(**bytes m**,** bytes32 r**,** bytes32 s**,** byte v1**)** returns **(**address addr**)** **{**

uint8 v **=** uint8**(**v1**)** **+** 27**;**

addr **=** ecrecover**(**m**,** v**,** r**,** s**);**

**}**

function bytesToBytes32**(**bytes memory source**)** returns **(**bytes32 result**)** **{**

assembly **{**

result **:=** mload**(**add**(**source**,** 32**))**

**}**

**}**

//

function verifysign**(**bytes m**,** bytes value**,** bytes pk**)** public returns **(**bool**)** **{**

**if** **(**pk **==** bytes**(**decode**(**m**,** value**)))** **{**

**return** **true;**

**}**

**return** **false;**

**}**

//

function inttoBytes**(**int256 x**)** returns **(**bytes b**)** **{**

b **=** **new** bytes**(**32**);**

assembly **{** mstore**(**add**(**b**,** 32**),** x**)** **}**

**}**

//find all in identitys[],and return id’s location in arrary

function Findaddress**(**int id**)** public constant returns **(**int**)** **{**

**for** **(**int p **=** 0**;** p **<** identitys**.**length**;** p**++)** **{**

**if** **(**identitys**[**p**].**id **==** id**)** **{**

**return** p**;**

**}**

**}**

**return** **-**1**;**

**}**

//注册时同时提交on、off钥对,valuex为pkx为钥对id的签名

function Register**(**int id**,** bytes32 name**,** bytes pkn**,** bytes pkf**,** bytes valuen**,** bytes valuef**)** public **{**

require**(-**1 **==** Findaddress**(**id**)** **||** identitys**[**Findaddress**(**id**)].**revoked**);**

**if** **(**verifysign**(**id**,** valuef**,** pkf**)** **&&** verifysign**(**id**,** valuen**,** pkn**))** **{**

identitys**.**push**(**Identity**({**

owner **:** msg**.**sender**,**

id **:** id**,**

name **:** name**,**

register **:** **true,**

updated **:** **false,**

revoked **:** **false,**

pkn **:** pkn**,**

pkf **:** pkf

**}));**

**}**

**}**

//更新密钥，密钥类型规定on为1，off为2；valueo为pko为钥对(id+pkw)的签名，valuew为pkw对id的签名

function update**(**int id**,** bytes32 name**,** int keytype**,** bytes pkw**,** bytes pko**,** bytes valuew**,** bytes valueo**)** public **{**

require**(-**1 **!=** Findaddress**(**id**));**

int thisaddr **=** Findaddress**(**id**);**

require**(!**identitys**[**thisaddr**].**revoked **&&** pko **==** identitys**[**thisaddr**].**id **&&**pkw **!=** pko**);**

bytes idpkw **=** inttoBytes**(**id**)** **+** pkw**;**

require**(**verifysign**(**idpkw **,** valueo**,** pko**)** **&&** verifysign**(**id**,** valuew**,** pkw**));**

**if** **(**keytype **==** 1**)** **{**

identitys**[**thisaddr**].**pkn **=** pkw**;**

**}**

**else** **if** **(**keytype **==** 2**)** **{**

identitys**[**thisaddr**].**pkf **=** pkw**;**

**}**

**else** **{**

**break;**

**}**

identitys**.**updated **=** **true;**

**}**

//注销操作,密钥类型规定on/off都要；valuenr为pkn为钥对(id+revoke)的签名，valuefr为pkf对id+revoke的签名

function revoke**(**int id**,** bytes32 name**,** bytes valuenr**,** bytes valuefr**)** public **{**

require**(-**1 **!=** Findaddress**(**id**));**

int thisaddr **=** Findaddress**(**id**);**

require**(!**identitys**[**thisaddr**].**revoked **);**

bytes idrevoke **=** inttoBytes**(**id**)** **+** bytes**(**"revoke"**);**

require**(**verifysign**(**idpkw **,** valueo**,** identitys**[**thisaddr**].**pkn**)** **&&** verifysign**(**id**,** valuew**,** identitys**[**thisaddr**].**pkf**));**

identitys**[**thisaddr**].**revoked **=** **true;**

**}**

//LOOKUP

function Lookup**(**int id**)** public view

returns **(**bytes20 pk**)** **{**

require**(-**1 **!=** Findaddress**(**id**));**

int thisaddr **=** Findaddress**(**id**);**

require**(!**identitys**[**thisaddr**].**revoked **);**

**return** identitys**[**thisaddr**].**pkn**;**

**}**

**}**