恒生研究院|高深的密码学+复杂的区块链，其实也可以通俗易懂

作者： 童世红
2017-06-16
7215
原创

密码学，在很多人看来是极为高深、只有数学家才能玩转的技术科学，不清楚如何才能应用于实际开发的过程中。本文将结合应用密码学比较深入的区块链，从扫盲、实战，再到实验、提高，一层层剥开密码学的“神秘面纱”。

【扫盲班—密码算法解析】

如果您还是密码学小白，那你需要先简单了解一下常用的密码算法：**对称加密、非对称加密、数字签名和摘要算法。**

＜对称加密＞

对称加密又叫传统密码算法，就是加密和解密使用同一个密钥。潜伏里面孙红雷通过电台收听到一堆数字，然后拿出一本书（密码本）比对，找到数字对应的汉字，就明白上级传达的是什么指令了。而军统的监听台没有密码本，只看到一堆没有意义的数字。

用数学公示表示就是：

▲加密： $E_k(P) = C$

▲解密: $D_k(C) = P$

这里E表示加密算法，D表示解密算法，P表示明文, C表示密文。留意以后会经常看到。常见的对称加密方法有DES、3DES、Blowfish、RC2、AES以及国密的SM4。

有同学会问，什么是国密啊？很机密么？没那么夸张，其实它的全称叫“国家商用密码”，是为了保障商用密码安全，国家商用密码管理办公室制定了一系列密码标准。

<非对称加密>

对称加密又快又方便，但是有个很大的坑——密码本容易被偷或被破解。从红军到二战，胜利的最大贡献其实就是破解密码。红军在数十倍的包围圈里面自由跳来跳去，那两台大功率电台功劳莫大。

怎么能够防止这种情况呢？1977年三位数学家Rivest、Shamir 和 Adleman 设计了一种算法(所以叫RSA)，把密钥分成两个，一个自己持有叫私钥(Private Key)，另一个发给对方，还可以公开，叫公钥(Public Key)，实现用公钥加密的数据只能用私钥解开：

▲加密: $E_{公钥}(P) = C$

▲解密: $D_{私钥}(C) = P$

这下就不用再头痛如何把密码本给对方或被破解了，私钥由自己保管，敌方拦截到密文也没有办法。

除了RSA之外，常见的非对称算法还有Elgamal、背包算法、Rabin、D-H、ECC（椭圆曲线加密算法）以及国家商用密码SM2算法。

非对称算法核心原理其实就是设计一个数学难题，使得用公钥和明文推导密文很容易，但是很难根据公钥、明文和密文推导私钥。

RSA是基于大整数因式分解难度，也就是两个质数相乘很容易，但是找一个大数的质因子非常困难，理论上破解RSA-2048(2048-bit)的密钥可能需要耗费10亿年的时间。

这儿说点题外话：强烈不建议使用RSA，原因如下：

▲容易被破解：RSA-768可以在3个小时内破解，1024在理论上100小时内也可以破解。所以使用RSA，长度起步要2048。但是数学家彼得·舒尔研究了一个针对整数分解问题的量子算法 (舒尔算法)，理论上破解2048的RSA在100秒之内(好在量子机还未投入使用)。

▲慢：密钥长度加到2048可以提升安全，但是计算过慢。

<数字签名>

有了非对称加密，数字签名就很容易理解了。

乙方收到甲方传过来的一串信息，怎么能够确定确实是甲方而不是有人伪造呢？

我们把非对称加密反过来做就可以了，因为只有甲方自己才持有一份秘密的私钥，他拿这个私钥对数据进行加密得到密文 C = EA私(M)，乙方持有甲方的公钥，解密明文P = DA公(C)，如果能够解密成功就证明信息确实是甲方所发。

不过通常不需要对发送信息的整个内容都加密，那样太慢。只需要计算一个信息的唯一信息摘要并对信息摘要加密解密即可，下面就会讲到数据摘要算法（俗称HASH算法），这也是数字签名的算法名称，很多时候是一个摘要算法+非对称算法，例如SHA1RSA, SHA256RSA等。

<摘要算法>

俗称HASH算法，学名杂凑算法，也就是从明文P生成较短的固定长度的杂凑值，保证不同的输入产生的输出是唯一的(重复几率非常非常小)。这样就可以广泛用于完整性检查、数字签名等场景。

常见的摘要算法有MD5、RIPEMD、SHA和国密的SM3。MD5不建议使用，已经被爆。

【实战班—区块链应用】

区块链提供了通过机器算法解决参与人之间的信任问题的全新方案，其核心的核心就是在不完全信任的各方，通过深度使用密码学算法来保证数据的不可篡改特性。

本节结合实际区块链中的应用，让大家在了解区块链的同时，一起惊叹原来加密算法还可以这么用。

<比特币之谁能动我的钱>

比特币是公有链，账本分布在无中心的节点上，任何一个节点都可以发出一个转让比特币的交易。那我的比特币是如何保证不被别人转走的呢？

假设你拥有100比特币（好有钱哟），那么在公开账本上存有一个数据结构，即所谓的UTXO，其主要内容有：

▲index: 索引

▲value: 金额

▲hash：一个SHA256的数据摘要

▲script: 脚本，这个是重点要讲的

这个script是一串可执行的二进制代码，比特币定义了一个基于堆栈的脚本执行器，可以执行加减乘除、移位、HASH、验签等算法，类似于常见的科学计算器。当你想花费持有的比特币时，首先需要执行作为输入交易对应UTXO的脚本(script)，称之为“解锁脚本”，只有执行成功才能继续。

最常用的一个解锁脚本就是P2PKH脚本：

```
OP_DUP  OP_HASH160  <  Public  Key  Hash>  OP_EQUAL
```

解锁时传入签名和公钥组成完整脚本：

```
< Signature> < Public Key> OP_DUP OP_HASH160 < Public Key
```

翻译起来就是：“公钥的HASH160等于<脚本里面的值>并且用这个公钥对HASH值验证签名能够通过”。计算通过，才可以花费这笔资金。

因为私钥保存在你自己手上，其他人无法计算出一个满足条件的签名，从而保证了这笔资金只有你自己可以使用。

这里除了数字签名外，还有一点体现了中本聪真的很聪明，账本上不会存储你的公钥，而是其HASH160(双HASH,SHA256+RIPEMD160)，由于HASH是单向的，从HASH无法反向推导公钥，这样大大减少未来量子机会带来的风险。

<区块链-HASH链之如何防止篡改>

前面讲了数字签名在比特币的用法，这里结合区块链数据结构本身讲一下HASH的用法。

一个块只是组织数据的结构，这里暂不详述，关键是块里面有个重要的参数 – 前一块的HASH，这样就形成一个链式结构。

我们把数据竖起来看，就像是玩积木游戏，节点你一块我一块向上罗，每一块和前一块都有个钩子。如果这个时候有人试图篡改之前的一笔交易，势必会导致那个块的HASH变了，那为了使得改过的交易被大家认可，他可以以这个被改过的块为起点，重新计算后面所有的块，关键是还得比拼得过全世界其他的节点，目前还没人能够做到。

这里就突出了HASH算法的特点：

▲数据改变一点点，HASH改变非常大。

▲无法给不同的数据计算出相同的HASH(或者说非常难)。

<比特币和以太坊的公私钥—ECC算法>

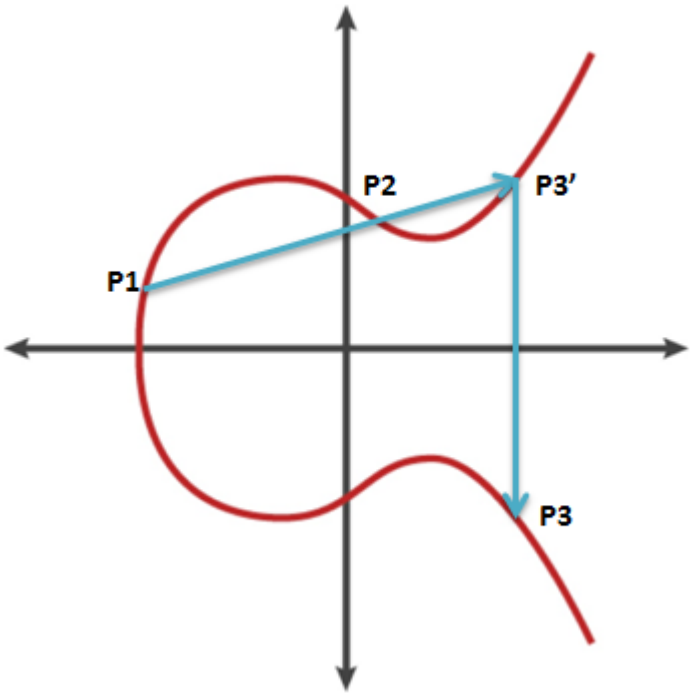
RSA又慢又不安全，所以比特币和以太坊都不采用，而是使用了更安全的椭圆曲线算法－ECC来做非对称加密基础算法。ECC的210位算法难度就相当于RSA 2048的难度，性能则是数量级的区别。那么椭圆算法又是何方神圣呢？

前面讲过非对称算法无非是设计一个数学难题，使得单向计算很方便，而反向计算很难，如RSA使用因式分解的原理，两个大质数相乘很容易，但大数分解质因子很难。

椭圆算法ECC其实就是利用乘法容易，而除法难的特点，设计一个乘法：K = k * G，其中大K是公钥，小k是私钥，G是生成点。由私钥推导公钥很容易，只需要k个G相加即可。但是从公钥推导私钥很难，也就是无法计算公钥K除以G。

当然这个加法不能用我们日常的整数加减法，而是利用函数 所定义的一个特殊椭圆曲线上散列点的特性定义的加法。其中p是一个常数。不同p可以设计成不同的曲线，比特币使用的$p = 2^{256} – 2^{32} – 2^9 – 2^8 – 2^7 – 2^6 – 2^4 – 1$，这个曲线的名称就叫secp256k1。这是一个非常大的数，曲线上的点是一个复杂散点，为了方便展示，这里用小很多的17阶曲线

表示加法的定义：



加法定义就是曲线上任意两个点P1和P2，必有第三个点P3，是P1和P2连线的延长线与曲线相交点的x轴映射的点，定义P1+P2=P3。通过数学算法可以证明这种点满足加法乘法交换律：

▲A + B +C = A + (B + C)

▲A*(B+C) = A*B + A*C

暂且不进行证明赘述，需要说明的是这里为了完善计算还定义了无穷远点O(相当于0)，满足：P1 + O = P1。（思考一下：如果P1 + P2 = O，那么P1 = -P2了吗？）

ECC加密过程：

▲K = k * G， 大K是公钥，小k是私钥；

▲把明文编码成曲线上的点M；

▲生成一个随机数r；

▲计算密文C1=M + r*K, C2 = r*G，其中大K是公钥；

▲对方收到密文后，可以计算C1 - kC2 = M，其中小k是私钥；

▲攻击者得到C1、 C2，公钥K以及基点G，没有私钥是无法计算出M的。

ECC算法用很短的密钥就能达到RSA2048的安全强度，而且计算速度有数量级的提高，所以目前应用很普遍，国密中的SM2就是基于ECC算法的。

【实验班-如何使用算法】

这些算法感觉还是挺复杂的，我们小白能用起来么？

不要只说不练，我们就实际操刀体验一下。

其实是别人把框架和算法都写好的啦，比如JAVA，在JDK里面就集成了Java密码学框架(Java Cryptography Architecture - JCA)，直接拿来用就行了，其他如C#，C++甚至JavaScript都有类似的。

<对称加密>

把大象关进冰箱需要三步，把明文转成密文也只需要四步：

▲生成一个密钥（如果已经有密钥，这步也省了），如：

```
SecretKey key = KeyGenerator.getInstance("SM4 ").generateKey();
```

▲取一个加密器:

```
Cipher cipher = Cipher.getInstance("SM4/ECB/PKCS7Padding");
```

▲初始化成加密模式:

```
cipher.init(Cipher.ENCRYPT_MODE, key);
```

▲加密:

```
byte[] ciphertext = cipher.doFinal(cleartext);
```

怎样把密文解密呢？初始化成解密模式就可以了。

SM4是国密4算法，初始化的时候斜杠后面的ECB、PKCS7...是什么？那是因为SM4是分组算法，在加密的时候会把明文先分成固定长度段，那就需要定义分组的模式和填充模式，只要加密解密用同样的模式就行了。当然不同的分组和填充模式各自有特点，那超出本文范围了，有兴趣的同学自学吧。

<非对称加密>

复习一下非对称加密和对称加密有什么区别啊？密钥分成公私钥对。

所以和对称加密区别只是：

▲在生成密钥的时候是一对，叫KeyPair。

▲加密的时候用一个如公钥，解密用另一个。

<摘要算法>

和加密提供了Cipher帮助类一样，HASH算法Java提供了MessageDigest帮助类，只需要调用getInstance就可以获取一个实例：

```
MessageDigestdigest=MessageDigest.getInstance(algorithm)
```

其参数是HASH算法，如SHA-256, SM3, MD5等。

调用update方法设置内容，然后调用digest就拿到HASH了。

```
digest.update(source.getBytes());
byte[] hash = digest.digest();
```

<数字签名>

签名:

```
Signature signer = Signature.getInstance(SIGN_ALGORITHM);
signer.initSign(keypair.getPrivate());
signer.update(plain);
signature = signer.sign();
```

验签:

```
Signature signer = Signature.getInstance(SIGN_ALGORITHM);
signer.initVerify(keypair.getPublic());
signer.update(plain);
boolean result = signer.verify(signature);
```

又比大象多一步。

<内参必读>

有几个要点在实际使用过程中必须要注意。

▲JDK自带的JCE实现算法不全

这里有两个原因：

(1)国家安全出口保护规定（美国）

根据美国安全出口规定，不能对某些国家出口RSA2048、AES256等以上安全算法。解决办法是到JDK的下载站上下载Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy，解压缩到JRE的lib/security下即可。

(2)国密和扩展算法缺失

免费提供的不能强求，不过还是有很多开源和商用的加密组件，这里推荐使用Bouncy Castle，虽然不是最快的，但是完全开源，支持C#, C++, Java多种语言。

把Bouncy Castle集成到JRE有两种方法，一种是修改JRE的java.security增加一个Provider，另一种直接在代码初始化的时候调用Security.addProvider加进去即可:

```
Security.addProvider(new BouncyCastleProvider());
```

▲Android 使用Bouncy Castle注意

Bouncy Castle(BC)很强大，Google的android内核也集成了。但是，由于安全要求，这个加密包是阉割过的，自己再集成BC又导致包冲突。

解决办法是换个包名，到https://rtyley.github.io/spongycastle/可以获取。

【提高班-隐私保护】

由于区块链是在非完全信任的一组参与人之间，通过算法解决信任问题，之前讲述的算法保证了数据不可篡改，只有自己才可以操作自己的数据，但是还欠缺一个很重要的课题 – 隐私。

隐私和不可篡改其实有些相互矛盾，要实现不可篡改，就得让其他人来验证数据，比如公有链是全网用户都来验证；但是隐私又想只有授权的人才可以验证，甚至希望其他人能验证但是不知道数据，比如盲签名、同态算法等。本节讲述在非安全环境下处理安全数据的一些方法。

<数字信封>

用非对称算法可以把机密信息安全传给指定的接收人，通常我们会使用对方的公钥进行加密，同时使用自己的私钥对数据进行签名。数字信封提供了一个更方便强大的方法，使得信息只有特定的接收人才可以阅读。

数字信封的功能类似于普通信封，内容被包起来，上面写了接收人，只有接收人才能拆信。

制作信封方法：

▲准备一个生成器

```
CMSEnvelopedDataGenerator edGen = new CMSEnvelopedDataGenerator();
```

▲添加接收人:

```
edGen.addRecipientInfoGenerator(<接收人>)
```

接收人可以是公钥证书、普通公钥或者密码，可以有多个。

▲制作信封

```
edGen.generate(<内容>,<加密器>);
```

拆信封时，只要凭自己的公钥找到自己的收件人信息，然后用持有的私钥抽取内容即可。

<组签名和环签名>

通常一个合同是以公司的名义进行签署的，例如公司A有三个合同经办人C1、C2、C3，均可以代表公司签署合同。

这里有几个要求：

▲所签署的合同使用公司的公钥可以验证确实是公司所签署；

▲能够进一步确定合同经办人的身份；

▲经办人如离职被吊销个人证书，不影响已有业务数据。

按照孙子定理，n个整数(公钥)的同余方程组是有唯一解的，那么理论上根据组员公钥集合{K1, K2, ..., Kn}选择一组模M，可以求解x做组因子, 实现组员使用自己的私钥ki和x可以对密文进行解密 D(ki, x, C) = P。

类似的原理可以应用到数字签名，实现：

▲群组签名：机构使用群组公钥做自己的公钥，可以通过验证签名确定签名属于指定的机构，而机构管理员可以进一步确定是那个成员签署的。

▲环签名：对于匿名要求，可以确定签名是来自于一个群组的成员，但是无法确定是具体哪个成员签署的。

<同态加密>

私密数据的处理通常是在组内进行，但是使用区块链技术后，私密数据的处理可能会需要在无中心的节点上，甚至是第三方的节点进行处理。这时就需要把要处理的数据在保密状态下进行。

例如股东A有100股，卖出60股剩余40股，这是一个减法操作。如果这个过程在智能合约中，智能合约又运行在多个非完全信任的节点上，如果需要将真实股份数量加密，则需要实现一个减法同态：

C3 = C1 - C2, 其中C1,2,3均是密文，执行减法的节点无法知道实际余额和发生额，但是股东A可以使用自己的密钥解密 D(C3) = P = P1-P2, 其中P表示明文，D表示解密算法。

目前已实现的算法主要有：

▲Paillier方案

概率公钥加密，基于复合剩余类的困难问题。满足加法和数乘同态。

▲BGV和RLWE方案

BGV和RLWE都是基于LWE(Learning With Errors)难题的同态算法, 支持加法、乘法、减法和移位运算的同态。源码在github上开源- HElib。

▲基于其他数学难题的方案

如基于决断问题等。

全同态算法虽然实现已经取得很大进展，但其实现效率还远未达到实用要求。

全同态算法是密码学的圣杯，等待您来夺取！

最后编辑：admin 于 2017-06-20 08:41:14

- 内存泄漏的原因有很多，但这个可能是你所不知道的！
- 如何巧用雅虎14条优化原则，从页面级到代码级全面提升前端性能？

发表评论

称呼

邮箱

接收邮件提醒

内容

保存 评论通过审核后显示。

友情链接

恒生区块链 Light官网

联系我们

TEL:0571-28829811 Email:rdcsupport@hundsun.com

恒生技术之眼

站长统计