# 摘要

略略略

**关键词：**门限签名，盲签名，ECDSA，同态加密

# ABSTRACT

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.……

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.……

……

**Key words:** Threshold Signature, Blind Signature, ECDSA, Homomorphic Encryption1

# 目录

目录

[摘要 1](#_Toc3522)

[ABSTRACT 2](#_Toc13528)

[目录 3](#_Toc8615)

[第一章 绪论 5](#_Toc12596)

[1.1 论文研究背景及意义 5](#_Toc19465)

[第二章 相关技术和理论介绍 6](#_Toc31944)

[2.1 Shamir秘密共享 6](#_Toc3525)

[2.2 2.2 ECDSA数字签名方案 6](#_Toc15442)

[2.3 Paillier同态加密修改方案 7](#_Toc27278)

[2.4 判定性Decisional Diffie-Helman(DDH)假设 9](#_Toc22244)

[第三章 基于ECDSA的门限盲签名方案 11](#_Toc32430)

[3.1 引言 11](#_Toc2842)

[3.2 角色定义 12](#_Toc15337)

[3.3 签名者初始化算法 12](#_Toc17140)

[3.4 接收者初始化算法 15](#_Toc20064)

[3.5 签名算法 15](#_Toc17318)

[3.6 安全性分析 28](#_Toc29094)

[3.7 计算性能分析 31](#_Toc226)

[3.8 本章小结 35](#_Toc32614)

[第四章 基于ECDSA的门限盲签名系统实现 35](#_Toc22370)

[4.1 开发环境搭建 35](#_Toc3144)

[4.2 系统功能实现 35](#_Toc20662)

[4.3 系统测试 35](#_Toc19708)

[第五章 总结与讨论 36](#_Toc25259)

# 绪论

## 论文研究背景及意义

2008年，中本聪提出了比特币这一去中心化加密货币的设计概念。随着2009年比特币系统的启动，比特币正式问世。在2010年到2015年期间，比特币逐渐走入了公众视野。而2016年到2018年，则见证了各国相继对比特币公开表态，以及全球主要经济体不确定性的增加，这促使比特币备受关注，需求飞速增长。事实上，比特币可谓是区块链技术最为成功的应用之一。随着以太坊等开源区块链平台的涌现，以及大量去中心化应用的落地，区块链技术正被更多行业所应用。随之而来的是越来越多的用户开始涉足区块链世界。数字签名是用户与区块链世界交互的主要方式，比如在比特币中用户使用私钥签署交易信息生成数字签名，再将其发布到区块链网络中由各个区块链节点进行共识以完成一笔转账操作，其他更复杂的交互也需要基于数字签名，比如各种基于智能合约的去中心化应用也需要用户签署特定消息来引发相关的接口调用。

为了使用户更便利地管理其数字资产，数字钱包等产品应运而生，成为用户与区块链之间的桥梁。然而，随着区块链技术的普及和用户群体的扩大，安全隐患也逐渐暴露出来。

遗失私钥或私钥被黑客攻击是区块链用户面临的重大风险之一。许多用户可能因个人疏忽，如存储私钥的不当或遗忘备份等，而导致私钥丢失，从而失去对其链上资产的所有权。更为严重的是，黑客可能利用各种技术手段入侵数字钱包或交易所，窃取用户的私钥，进而窃取用户的数字资产，给用户带来巨大的经济损失。//数据或新闻引用

这些恶意事件的发生凸显了当前数字资产管理和交易过程中的安全性问题。用户对于自身数字资产的安全性越来越关注，迫切需要更加可靠和安全的解决方案来保护其资产免受损失。因此，区块链技术领域亟需加强安全意识和技术防范手段的研发，以应对不断演变的安全威胁，为用户提供更加可靠的数字资产保障。

# 相关技术和理论介绍

## Shamir秘密共享

Shamir秘密分享是由Adi Shamir在[]中提出的一个算法。该算法可以将一个秘密值分成n个分片，当得知的分片数量大于一个阈值时，才可以从分片中还原出秘密值。基本原理是利用多项式插值。具体来说，原始秘密被表示为一个多项式的常数1项，多项式的次数取决于需要的最小部分数量。然后，多项式在不同的点上被计算得到秘密分片值，这些分片值被分发给不同的参与者。当需要恢复原始秘密时，只需收集足够数量的秘密分片值，然后使用插值技术即可重建出原始秘密。

对于一个秘密值，以为常数项构造一个次多项式:

(2-1)

则对于不同的取值，如可以计算出不同的函数值，这些函数值与其自变量共同构成一个秘密分片值，当收集到t+1个分片值时，可以对t+1个分片通过以下运算重建原始秘密:

(2-2)

(2-3)

## ECDSA数字签名方案

数字签名算法DSA是1991年由Karivtz在文献[]中提出，随后在1994年由NIST修改成数字签名标准。而ECDSA是数字签名算法的一种变体，在区块链和加密货币中受到广泛应用。一个椭圆曲线数字签名方案包括四个部分。

1. 公共参数。
   1. 一个阶为的椭圆曲线循环群，以及其生成元，其中为素数。
   2. 哈希函数，可以将任意类型的消息M映射到
2. 密钥生成算法：输入一个安全参数，输出一对公私钥。私钥为从中随机选取的整数，公钥为。
3. 签名生成算法：输入一个任意的消息,输出签名。
   1. 计算。
   2. 随机选择整数。
   3. 计算，取的横坐标为。
   4. 计算。
   5. 输出。
4. 签名验证算法：输入，和。
   1. 验证和是否都属于。
   2. 计算。
   3. 取的横坐标为，如果则签名验证通过，否则该签名无效。

## Paillier同态加密修改方案

### Paillier同态加密方案

Paillier同态加密方案[]是一个基于离散对数困难问题构建的加密方案，它具有加法同态的性质。

1. 密钥生成算法：
   1. 随机选取两个大素数，需要满足条件与互素，不满足则重新选择随机数。
   2. 计算，其中表示最小公倍数。
   3. 随机选取一个正整数，要求并存在，若不存在，则需要重新选取,其中：

(2-4)

* 1. 公钥为，私钥为。

2、加密算法：

对于要加密的消息，它需要满足，它的加密过程如下：

* 1. 随机选取一个正整数，要求，且还需满足与互素。
  2. 计算密文如下：

(2-5)

3、解密算法：

对于密文，解密过程为：

(2-6)

4、加法同态性质：

Paillier加密具有加法同态的性质，使用表示对进行加密得到的密文，对于任意的消息，加法同态表示为：

(2-)

(2-)

将定义为Paillier的同态加运算，定义为Paillier的同态数乘运算，则上述性质可以表示为：

(2-)

(2-)

### Paillier同态加密修改方案

Paillier同态加密因其同态加密的性质而得到广泛应用，然而在文献[]中提到将Pailiier方案直接应用于构建盲ECDSA签名可能导致更多信息的泄露，而这部分信息的泄露所带来的安全性影响是未知的。因此文献[]在此基础上提出了一个Paillier同态加密的修改方案，并证明了它在选择明文攻击下是语义安全的，它的密钥生成如下：

1. 密钥生成算法：
2. 随机选取两个大素数，需要满足条件与互素，不满足则重新选择随机数。

（2）计算，其中为ECDSA中的公共参数椭圆曲线群的阶。得到的公钥为，私钥为

2、加密算法：

（1）选择要加密的消息，它需要满足

（2）随机选取一个正整数，要求，且还需满足与互素。

计算密文如下：

(2-)

3、解密算法：

对于密文，解密过程为：

(2-)

对于修改后的Paillier同态加密方案，同样具有同态加法性质，即密文的相加等于明文相加后再加密。为了方便表达，本文第三章中提到的Paillier同态加密方案指的都是Paillier同态加密的修改方案。

## 判定性Decisional Diffie-Helman(DDH)假设

判定性DDH假设是一个密码学中的一个重要数学假设，在数字签名、承诺和数字加密方案中有着广泛应用。判定性DDH假设的定义如下：

给定阶为的循环群，其中是一个大素数，为循环群的生成元，无法在多项式时间内区分以下两个四元组和，其中：





# 基于ECDSA的门限盲签名方案

## 引言

在本章中，我们提出了门限盲签名方案TB-ECDSA（Threshold Blind ECDSA）。该方案涉及两个主要参与角色：接收者和签名者。我们假设接收者是用户本身，具有诚信性，而签名者可能存在恶意行为。在这种前提下，我们设计了TB-ECDSA的三个算法，包括签名者初始化算法、接收者初始化算法和签名算法。本文提出的TB-ECDSA方案具有以下几个特点：

1. 门限签名：系统中允许有位签名者，当位签名者同意参与签名时才可以产生合法的签名，其中,满足。签名者必须达成共识并同意参与签名操作才能产生合法的签名。这种门限签名机制确保了多个签名者的合作，增加了签名的可靠性和安全性。
2. 盲签名：参与签名的签名者无法获知签名的相关信息。通过使用同态加密技术，签名者在签名过程中只能对经过同态加密的密文进行同态运算处理，无法还原出原始消息的内容。这种消息的盲性保护了签名的机密性和隐私性，确保消息内容不会被签名者获取。
3. 签名的不可链接性：防止签名者推断出签名的过程与后来公开的签名之间的确切对应关系。不可链接性对于维护请求者的隐私和匿名性非常重要。通过防止签名者将签名过程与最终签名联系起来，请求者的身份和签名消息的内容保持机密。这个特性确保签名者无法追溯请求到具体签名，从而保护请求者免受潜在的隐私泄露或重新识别的风险。
4. 错误识别：在签名算法中，如果签名者违反了算法规定的计算方式，导致最终产生的签名不合法，接收者可以通过分析签名过程中产生的中间变量和公开的参数来追溯是哪个签名者的错误行为导致了签名的无效。通过对比签名者之间的计算结果和公开参数，接收者可以定位并识别出签名过程中的错误行为，从而可以追究责任或采取相应的措施来防止类似错误的再次发生。这种错误识别机制增强了签名算法的安全性和可靠性，确保签名者按照规定的方式进行签名，从而保护签名的有效性和准确性。
5. 用户匿名性：用户匿名性是指在签名过程中，即使存在多个签名者共谋，它们也无法确定特定数据是否来自同一个用户。这种匿名性保证了用户的身份和行为无法被追踪或识别。此外，由于用户匿名性的存在，签名者之间也无法确定私钥与特定用户之间的对应关系，这增加了共谋者窃取私钥的难度，进一步保护了用户的隐私和匿名性。因此，用户匿名性在签名过程中为用户提供了更高级别的隐私保护，防止了身份泄露和数据关联。

## 角色定义

在整个TB-ECDSA方案中，有两种角色：接收者和签名者。

1. 接收者：接收者用符号表示，他代表了使用TB-ECDSA的用户。接收者是最初发起签名请求的人，并最终接收到签名分片并完成聚合。在TB-ECDSA方案中，我们将接收者视为可信的实体。在进行签名算法之前，接收者会使用私钥作为秘密构造Shamir秘密共享，并将私钥分片分发给签名者。在签名算法过程中，接收者提供待签名的消息，并参与签名的生成过程，作为签名者之间的消息中转，以避免签名者之间直接联系破坏用户的匿名性。完成签名后，签名者会检查签名的合法性。如果签名不合法，接收者会对签名的中间数据进行排查，以识别违反协议的签名者。
2. 签名者：签名者用符号表示，下标用于区分不同签名者。在TB-ECDSA中，个签名者组成签名者集合。签名者是签名算法的执行者，并拥有私钥分片。在TB-ECDSA方案中，签名者被视为不可信的实体。在进行签名算法之前，签名者需要以个签名者为一组进行初始化，生成代表该组签名者的相关私有或公共参数。在进行签名算法时，签名者使用私钥分片和随机数与接收者交互逐步计算签名分片。

## 签名者初始化算法

假设所有位签名者中的位组成签名者集合，签名者表示为，每个都要在初始化的过程中得到一个属于自己的保密掩盖因子分片，以及一个公共参数掩盖因子，其中：



进行一次初始化还需要一个协调签名者进行，协调签名者可以由集合以外的任意成员担任。一次初始化的过程如下：

公共参数：

1、每个或都有一个正整数或作为自己的唯一标识。

算法步骤：

1. 随机选取一个大素数，作为掩盖因子分片。
2. 以为常数构造一个一次多项式，其中是随机选取的系数。
3. 计算，暂存。
4. 计算和，将发送给，将发送给
5. 或收到其他签名者发送来的或，计算：



或



1. 或计算或。其中:

(3-)

1. 或向参与初始化的签名者公布或。
2. 或收到和后计算。
3. 计算，公开和掩盖因子，完成初始化。

一次初始化流程所产生的和与当前参与流程的签名者集合对应，因此对于个签名者，阈值为的系统，需要进行次初始化过程。各个签名者持有个代表它与不同签名者组成签名者集合时应该使用的掩盖因子分片。以一个协调签名这与三个签名者进行一次签名者初始化算法为例，图X-x是一次签名者初始化算法的执行示意图。

*S3*



*Sc*



*S1*



*S2*



图x-x 签名者初始化算法执行示意图

我们对这个初始化流程做一些原理解释。在步骤5中计算的(类似)可以展开成如下形式：



其中是由和这两个常数组合相乘构成的常数，我们并不关心的具体值。式()的形式可以看作是一个多项式函数在自变量为时的取值，并且这个多项式的常数项为，即如下式：





因此可以将或看作是关于的一个Shamir秘密共享分片，它的阈值为。然后在步骤6中，利用将或转换乘了加法分片或。在所有签名者在步骤7公布自己的加法分片，而或中是不会泄露各个签名者的的。最后各个签名者就可以通过对这些加法分片求和来得到即掩盖因子。

## 接收者初始化算法

接收者需要准备个Paillier加密方案的公私钥对，其对应的加密和解密操作记作和。

接受者在请求签名者签名前需要将私钥以秘密共享的方式将私钥分片发送给签名者，签名者将私钥分片保存好，向接收者返回存储索引，接收者需要保存存储索引，以便在签名过程中使用。

接受者拥有私钥和对应公钥*，*接收者密钥分发步骤如下：

1. 以为秘密构造一个Shamir秘密共享，阈值为t，秘密分片为。
2. 记录，分别将发给。
3. 接收到后，存储并发送向发送存储索引。
4. 保存索引，删除，将秘密备份完成注册。

以一个接收者与三位签名者完成接收者初始化算法为例，图x-x是接收者初始化算法的执行示意图。

*R*

*S1*

*S2*

*S3*



以*x*为秘密构造Shamir秘密共享



储存*u1*

储存*u2*

储存*u2*



保存索引

图x-x 接收者初始化算法执行示意图

## 签名算法

签名算法是TB-ECDSA方案中最核心的部分，在签名算法中，由接收者提供待签名的消息，然后接收者与签名者共同计算出符合ECDSA数字签名方案的数字签名。

为了实现用户匿名性，TB-ECDSA的签名算法在设计时受到了以下规则约束：

1. 在一次签名算法中签名者不能发送含相同内容的数据。
2. 在一次签名算法中签名者不能接收含相同内容的数据。
3. 在一次签名算法中签名者不能计算出含相同内容的数据。
4. 在一次签名算法中签名者不能直接向另外一名签名者发送数据。

当上述规则中的任意一条无法满足时，签名者共谋时就可以相关的数据匹配到对应的接收者，使得TB-ECDSA的签名算法无法满足用户匿名性。在这些规则的约束下，接收者和签名者的交互有以下特点：

1. 签名者只与接收者来往数据。
2. 接收者收到的所有数据都会经过处理后再发送给下一个签名者。

### 阶段一

签名算法的阶段一目的是为了生成椭圆曲线签名中的随机点，在这个过程中只有接收者最终得知了，签名者参与的生成但是无法得知的真实值。

公共参数：

1、椭圆曲线群的相关参数

私有参数：

1. 签名者:
   1. 掩盖因子分片

算法步骤：

1. 选择一个随机数，随机数。
2. 计算。
3. 计算。
4. 发送给。
5. 接收到，后，随机选择一个，计算并将发送给：







1. 接收到后，计算，如果成立，继续协议，否则中止协议。
2. 计算，将作为步骤2中的，对其他签名者依次进行步骤3~6直到与所有的签名者都交互一次。
3. 完成步骤7,会得到







1. 计算。

我们对阶段一的步骤作详细的解释。步骤1~4中，接收者选择了一个随机数，然后接收者对随机数做倍点运算得到签名随机点的中间结果，同时接收者再用另外一个随机数与倍点运算得到，然后将和发送给一个签名者。

步骤5中，签名者接收到相关数据后将他选的随机数对及做相同的倍点运算得到和，再将随机数与掩盖因子分片的逆的乘积做倍点运算得到，再将发给接收者。

步骤6中，接收者收到数据后，计算，验证。因为与之间有数学关系，假如签名者对及没有作相同的运算，或者偷换了其中的数据就会导致不成立，证明签名者偏离了协议，没有依照协议对做指定的运算，此时中止当前协议。如果验证通过了则继续进行协议，接收者保存，以便进行后续的验证。

步骤7中，签名者在将中间结果发送给下一个签名者继续处理前，需要用自己的随机数与这个结果做倍点运算，然后再发送给下一位签名者。这样做是为了混淆前后两个签名者发送或者接受的数据，使后收到数据的签名者无法推测出上一个处理该数据的签名者是谁，避免他们将运算中间结果联系起来推测出更多关于接收者的信息。接收者对完成处理后，发送给下一个签名者。如此往复直到所有的签名者都将自己的随机数与进行了倍点运算。此时接收者得到的依旧是一个中间结果，需要对这个结果使用再做一次倍点运算才得到生成椭圆曲线签名需要的随机点。经过上述协议后得到的的具体值为：



那么对应的为：



虽然在步骤4中首先将数据发送给了处理，实际上接收者与签名者的交互顺序是可以任意的，只要保证所有签名者都将自己的随机数与中间结果作了倍点运算即可。以接收者与三个签名者共同进行阶段一为例，图x-x是进行一次阶段一的执行示意图。

*R*

*S1*

*S2*

*S3*

选择随机数*kr*∈Z*q，*α∈Z*q*



验证



验证



验证



图x-x 签名算法阶段一执行示意图

### 阶段二

签名算法阶段二的目的是继续完成签名。在阶段二中，接收者会与签名者们共同完成的计算，接收者会对签名数据进行同态加密后再发送给签名者。签名者们会对密文进行同态运算，将自己掌握的秘密随机数，掩盖因子分片，私钥分片运算到加密后的签名数据中，逐步完成被遮盖的签名分片。最后由接收者使用公开的掩盖因子去除掩盖，再聚合签名分片获得，再最终组合成。

公共参数：

1. 掩盖因子

私有参数：

1. 接收者
   1. 个Paillier加密方案的私钥
   2. 个Paillier加密方案的公钥
   3. 私钥分片的倍点
   4. 私钥分片对应的索引
   5. 随机点
   6. 待签名的消息
2. 签名者
   1. 私钥分片
   2. 掩盖因子分片
   3. 随机数

协议步骤：

1. 接收者取的横坐标为。
2. 接收者对要加密的消息进行加法分片得到份，其中满足:



1. 接收者使用个Paillier加密方案加密和得到和。
2. 接收者选取个随机数，计算和。
3. 接收者将发送给。
4. 签名者接收到后，根据找到对应的，然后计算签名分片中间结果：



(3-)



1. 签名者将发送给接收者。
2. 接收者接受到，验证下列式子是否成立。



如果验证通过，则继续协议，否则中止协议。

1. 接收者计算



1. 接收者重新选取个随机数，计算



1. 接收者将发送给。
2. 签名者收到后，计算：





1. 签名者将发送给接收者。
2. 接收者接收到。
3. 接收者将作为步骤8中的，重复步骤8~14，直到所有签名分片中间结果被所有签名者都处理了一遍。以“\*”代表任意签名者的下标组合，在步骤11中，每次都需要发送给一个未处理过该部分签名分片的签名者。
4. 完成步骤17接收者会得到份经所有签名者处理过的中间结果
5. 接收者验证下列式()是否成立，如果验证通过，则继续协议，否则中止协议：



1. 接收者计算：









19、接收者组合出完整的签名

我们对阶段二的步骤做详细的解释。在阶段二中，首先步骤1~2由接收者完成的计算以及对消息的分片，分片的结果对应每个签名者。对消息分片的具体方式可以使用Shamir秘密共享，或者其他更方便的方法，只需要保证。

接着在步骤3中，接收者使用不同的Paillier公钥加密和得到。在这一步中使用了不同的Paillier公钥加密是为了得到不同的密文，以便发送给不同的签名者，保证他们不会接收到相同的信息。同时由于后续需要一起进行同态运算，所以消息分片需要与在同一个公钥下加密。

在步骤4中，接收者选取了个随机数与进行同态数乘运算得到。的作用是使得后续步骤9中接收者可以对签名者返回的消息进行验证，确保签名者没有偏离协议。

在步骤6中，签名者接收到了加密的和即，以及其用于校验的数据。签名者首先通过式()将它的密钥片段转换成加法分片，然后进行了式()的运算得到了。根据同态加密的性质可以写作下式：

，

从式()可以看出是明文的密文。其中括号外的因子中是当前签名者在阶段一中选择的随机数，是在初始化算法中确定的掩盖因子分片。因子的目的是为了掩盖信息，其作用在后续步骤中将进一步说明。与经过了相同的运算，由于同态加密的性质可以写作下式：



从式()可以看出对应的明文是。它只与的明文相差了一个因子。

步骤8中接收者对和进行验证计算，验证是否由同态数乘所得。如果验证不通过说明签名者没有依照协议对和做指定的运算，此时中止当前协议。如果检查通过了则继续进行协议。

步骤9，接收者使用与作同态乘法得到。这个步骤有两个作用，一方面进一步完成了签名分片的计算，此时的明文为；另一方面也避免了将上一个签名者的数据直接发送给下一个签名者的情况。

步骤10的作用与步骤4类似。

步骤11将发送给下一个从未处理过当前签名的分片数据的签名者。我们举个例子来更好地说明此步选取下一个签名者的要求，假设一个签名分片数据，它的值如下：



对应的明文是，从左边的因子可以看出它已经被和处理过了，因为这个数据都包含了这两个签名者选取的随机值以及掩盖因子分片。因此这个不需要再发给或进行处理，只能发给除他们以外的任意一个签名者，比如。接收者可以根据每次数据交互的对象来记录每个签名分片数据已经被哪些签名者处理以及还需要发送给哪些签名者。

步骤12~13，签名者使用同态运算将乘入和后回发给接收者。在这一步中，乘入是为了完成签名，而的作用是为了掩盖。的作用具体而言可以从接收者视角进行分析，在步骤13中，接收者发送给签名者，在步骤16中，接收者接收到了来自处理过的。此时接收者可以对和进行解密获得明文，假设的明文为,那么的明文为,接收者对两个明文进行以下运算：



而正是乘入的数值，假如没有这个因子，那么接收者可以直接分析和得知的秘密值，这样是不安全的。因此需要引入一个掩盖因子，在接收者未知和的情况下，从计算出是相对困难的，由此保证签名者选取的秘密随机值不会泄露。

步骤17的作用类似步骤8。

步骤18，接收者已经获得了所有签名分片数据，此时这些签名分片数据的明文是



与真正的签名分片还差并且还需要去除掩盖因子。式()的作用是将运算到签名中，因此有：



此时得到的是依旧被掩盖因子遮盖的签名分片。将求和得到被掩盖的签名：



式()是使用公开的掩盖因子去除掩盖因子得到，即下



再结合式()则可将化为：



与ECDSA数字签名方案中的式()一致，说明最终结果是一个有效的ECDSA数字签名。以接收者与三个签名者完成签名算法为例，图x-x和图x-x是一次签名算法阶段二的执行示意图。

*R*

*S1*

*S2*

*S3*

取*K*的横坐标为*r，*构造消息分片*m1,m2,m3*，选取随机数***β1,β2,β3***



验证



图x-x签名算法阶段二步骤1~14执行示意图

*R*

*S1*

*S2*

*S3*



验证



图x-x 签名算法阶段二重复步骤8~14以及完成签名执行示意图

### 签名验证和错误识别

#### 签名验证

TB-ECDSA算法生成的数字签名与ECDSA的签名形式是一致的，因此验签方式与ECDSA一致，接收者只需要计算



取的横坐标位，验证是否成立。如果验证通过，说明生成了合法的签名。如果验证不通过，说明签名者中有人偏离了协议，造成了最终验证失败。

#### 错误识别

当接收者发现生成的签名无法正常验签时，可以通过签名算法中的中间数据来排查出偏离协议导致签名失效的签名者。TB-ECDSA方案可以排查出以下两类偏离协议的签名者：

1、第一类偏离协议：如果签名者在签名算法阶段二步骤7中偏离了协议，发生了以下几种情况之一：

* 1. 没有使用正确的私钥分片签名。
  2. 没有使用正确的掩盖因子分片。
  3. 没有使用阶段一所选取的随机数。

则接收者可以通过以下步骤排查出该签名者：

1. 计算：

。

1. 验证下式是否成立。

()

假如成立，说明在步骤7没有出现偏离协议，否则说明出现了第一类偏离协议。

我们对验证的原理进行解释，对于第一类偏离协议的签名者的排查，假如签名者没有偏离协议，应该有如下取值：



我们对式()等号两边分别逐级展开如式()和式()。





可以看出当签名者遵循协议流程时，式()左右两边应该是相等的。而当签名者偏离了协议时会导致，使得式()不成立。

2、第二类偏移协议：如果签名者在签名算法阶段二步骤14中偏离了协议，发生了以下几种情况之一：

* 1. 没有使用正确的掩盖因子分片。
  2. 没有使用阶段一所选取的随机数。

则接收者可以通过以下步骤排查出该签名者；

1. 计算：





1. 验证下式是否成立：



假如成立，说明在步骤14中没有出现偏离协议，否则说明出现了第二类偏离协议。

我们对验证的原理进行解释，对于第二类偏离协议的签名者的排查，假如签名者没有偏离协议，和应该有如下取值：





我们对式()等号两边分别逐级展开：





可以看出当签名者遵循协议流程时，式()左右两边应该是相等的。而当签名者偏离了协议时会导致，使得式()不成立。

## 安全性分析

### 不可伪造性

假设有一个攻击者，他已知个消息，以及这些消息对应的签名。假如无法根据和对一个未签名过的消息生成签名，则称这个签名方案是不可伪造的。

TB-ECDSA是满足不可伪造性的。TB-ECDSA数字签名等价于一个ECDSA数字签名算法计算出的签名。攻击者获取到个TB-ECDSA算法计算出的数字签名相当于获取到个ECDSA数字签名。而ECDSA数字签名具有不可伪造性，因此攻击者无法从个TB-ECDSA数字签名中生成一个新的有效的签名。因此TB-ECDSA具有不可伪造性。

### 门限特性

如果在一个签名方案中，私钥以分片的形式保存在个对象手中，完成一次合法的签名需要其中的个对象参与，如果不足个对象参与签名，则无法完成合法签名，称这样的签名方案是满足门限特性的。

TB-ECDSA是满足门限特性的。在TB-ECDSA接收者初始化算法中，接收者以私钥为秘密构造了Shamir秘密共享，将私钥分片分发给了个签名者。根据Shamir秘密共享的特性，只有当位签名者中的位提供他们的私钥分片才可以通过式()将私钥还原出来。根据Shamir秘密共享的特性，假如不足位签名者提供私钥分片，则无法还原私钥，进而也不可能完成一个合法的ECDSA签名。综上TB-ECDSA是满足门限特性的。

### 盲性

盲性指的是在整个生成签名的过程中接收者不会向签名者揭露任何与签名相关的信息。

TB-ECDSA是满足盲性的。在签名算法的阶段一，假设接受者与签名者交互的顺序是则在阶段一中，接收者分别向签名者揭露了，而与签名相关的信息是和，其值分别是：





而在不知道的情况下，每个无法由推导得到和。因此在阶段一过程中，没有接收者得知了和的具体值。假如签名者试图篡改数据来获取更多信息，也会因为其偏离了协议在阶段一中的步骤6的验证或者在最后进行错误识别时被检查出来。

在阶段二中，接收者首先对消息进行分片，再对其加密得到，发送给签名者和，这些数据都是经过同态加密的。假如Paillier加密方案是语义安全的话，在没有对应的解密密钥的情况下，签名者是将这些数据与随机数区分开的，因此他们无法从这些数据中得知任何关于签名的敏感信息。在阶段二的后续步骤中签名者接收的都是经过其他签名者以及接收者同态运算过的加密的数据，这些数据同理也是无法被签名者解密或者提取更多有用的信息的。假如签名者试图通过篡改数据的方法来获取更多信息，也会因为其偏离了协议，在阶段二中步骤9的验证或者最后进行错误识别时被检查出来。

因此签名者在签名的整个过程中都无法得知关于签名的任何相关的信息，TB-ECDSA满足盲性。

### 用户匿名性

用户匿名性是指在签名过程中，即使存在多个签名者共谋，它们也无法确定特定数据是否来自同一个用户。用户匿名性可以由以下例子进行说明。假设有接收者和，签名者和，接收者和都向两位签名者请求了签名，两位签名者依照TB-ECDSA签名算法完成了签名。设整个签名算法中向发送的所有数据的集合为。此时持有，持有。当某个攻击者控制了和获得了所有的数据，此时持有，但他无法根据数据的内容确定四组信息中的哪两组是来自同一个接收者。再进一步说明，知道和是来自两个不同的接收者，和是来自两个不同的接收者，但无法确定的接收者对应的是还是的接受者，的接收者对应的是还是的接收者。这样的特性我们称为用户匿名性。

我们分别从TB-ECDSA签名算法的两个阶段说明该算法具有用户匿名性，其中签名算法阶段一的部分采用较为严谨的证明的方式，签名算法二的部分采用直观地说明的方式。

1、证明TB-ECDSA签名算法阶段一满足用户匿名性：

定理 如果TB-ECDSA签名算法阶段一是不满足用户匿名性的，那么决定性DDH假设不成立。

证明 假设有接收者和，签名者和。和分别与签名者们完成一次阶段一的算法。和选择的秘密随机数分别为和。和选择的另一随机数为和。在响应和的过程中选择的随机数分别为和。此时，收到的所有数据如下：





因为TB-ECDSA中的阶段一是不满足用户匿名性的，则存在一个攻击者控制了，得知了他们收到的所有数据，有能力通过多项式时间内的函数以不可忽略的优势确定两个数据是否来自同一个接收者，如式()所示，当输出1时表示输入的两个循环群元素来自同一个接收者，如式()当输出0时表示输入的两个群元素来自不同的接收者：





一个等价于判定性DDH假设的命题如下：

给定阶为的循环群，其中是一个大素数，为循环群的生成元，且对于任意的，总能找到。无法在多项式时间内以不可忽略的优势区分以下两个四元组：





利用的能力可以在多项式时间内以不可忽略的优势区分和如下：





因此判定性DDH假设不成立。

定理的逆否命题为：如果判定性DDH假设成立，那么TB-ECDSA签名算法阶段一是满足用户匿名性的。而判定性DDH假设是本文的依赖前提，故在此前提下TB-ECDSA签名算法阶段一是满足用户匿名性的。

2、TB-ECDSA签名算法阶段二满足用户匿名性：

假设有接收者和，签名者,和。和分别与签名者们完成一次阶段二的算法。此时,和接收到的部分数据如下：







假设一个攻击者控制了,和，那么他得到了以上数据。由于Paillier加密方案是语义安全的，在没有获得对应的私钥的情况下，他无法从这些数据中提取有效的信息，因此它无法无法根据数据的内容对这些数据按接收者进行分类。例如，无法判断数据与中的哪一个来自同一个接收者。同理对于其他的数据组合也是一样的。所以TB-ECDSA签名算法阶段二满足用户匿名性。

## 计算性能分析

### 理论分析

代表一次椭圆曲线的倍点运算，代表一次模指数运算，代表一个整数的比特长度，代表参与运算的签名者数量。因为模加运算和模乘运算的运算资源消耗相对于椭圆曲线倍点运算和模指数运算是比较小的，因此我们没有统计TB-ECDSA中的这两类运算。

表x-x统计了在签名算法阶段一的计算和通信复杂度，其中为公共参数椭圆曲线群的阶，为参与的签名者数量。并且，由于步骤7是重复步骤3~6，因此步骤7的统计为步骤7以前的计算复杂度的累加再乘以重复的次数，重复的次数取决于参与的签名者数量，在此处为。

表x-x 签名算法阶段一的计算和通信复杂度

|  |  |  |  |
| --- | --- | --- | --- |
| 步骤 | 接收者计算复杂度 | 签名者计算复杂度 | 通信复杂度(bit) |
| 1~4 | 2*mp* |  | 4*|q|* |
| 5 |  | 3*mp* | 6*|q|* |
| 6 | *mp* |  |  |
| 7 | (*t*-1)3*mp* | (*t*-1)3*mp* | 10(*t*-1)|*q*| |
| 8~9 | *mp* |  |  |
| 总计 | (3*t*+1)*mp* | 3*tmp* | 10*t|q|* |

表x-x统计了签名算法阶段二的计算和通信复杂度，其中为私钥索引，为Paillier公钥，由于在阶段二中用到了多个Paillier公钥，因此表示的是多个Paillier加密密文构成的平均值。步骤15是重复步骤8~14，因此步骤15的统计为步骤8~14的计算复杂度乘以重复的次数，重复的次数取决于参与的签名者数量，在此处为。另外虽然Paillier加密的计算是，但是因为，所以，所以Paillier加密计算就可以等效为。可以看出Paillier加密计算只需要进行一次模指数运算。因此统计时对于每次Paillier加密都当作是一次模指数运算。

表x-x 签名算法阶段二的计算和通信复杂度

|  |  |  |  |
| --- | --- | --- | --- |
| 步骤 | 接收者计算 | 签名者计算 | 通信复杂度(bit) |
| 1~5 | 4*t* exp |  | *t*(*|N2|+|index|*) |
| 6~7 |  | 4*t* exp | 2*t|N2|* |
| 8~14 | 3*t* exp | 2*t* exp | *4t|N2|* |
| 15 | (*t*-2)3*t* exp | (*t*-2)2*t* exp | (*t-2*)*4t|N2|* |
| 17~19 | 3*t* exp |  |  |
| 总计 | (3*t2*+4*t*) exp | (2*t2*+2*t*)exp | (4*t2*-*t*)|*N2*|+*t*|*index*| |

综合表x-x和表x-x，我们可以得出TB-ECDSA签名算法接收者计算复杂度为：



签名者的计算复杂度为：



通信复杂度为：



### 实验分析

实验环境如下：

操作系统：Windows 10

CPU：

内存：16G

在实验中，我们使用了tss-lib[]提供的程序框架和相关密码学工具库来实现TB-ECDSA方案。首先，我们测试了TB-ECDSA方案中各个算法在不同阈值下的运行时间。随后，我们将TB-ECDSA方案与同样由Go语言实现的几个方案的签名进行了对比。

表X-X展示了TB-ECDSA各个算法在不同阈值下的运行耗时。我们利用协程来模拟参与算法的各个参与者，每个参与者在对应的协程中运行算法，并通过通道进行信息的收发。

随着参与者数量的增加，三个算法的运行耗时也随之增加。签名者初始化算法的主要运算在于Shamir秘密共享，其中的数学运算主要涉及整数的乘法和加法，因此签名者初始化算法是三个算法中运行耗时最低的。另一方面，接收者初始化算法需要选取一个大素数，而大素数的生成算法很大程度上依赖于随机性，因此接收者初始化算法是三个算法中运行耗时最高的。

表x-x TB-ECDSA方案各个算法运行耗时

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算法 | t=4(ms) | t=8(ms) | t=12(ms) | t=16(ms) | t=20(ms) |
| 签名者初始化 | 1.223 | 2.141 | 3.410 | 5.136 | 7.589 |
| 接收者初始化 | 5772.319 | 17150.084 | 25080.908 | 34697.574 | 45148.909 |
| 签名 | 148.575 | 357.151 | 623.568 | 981.897 | 1419.268 |

表x-x 门限签名方案比较

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 方案 | 盲性 | 用户匿名性 | 签名类型 | 错误识别 | 通信轮次 | 文献 |
| TB-ECDSA | 是 | 是 | ECDSA | 是 | 密钥生成：1轮  签名：2t轮 |  |
| GG18-ECDSA | 否 | 否 | ECDSA | 是 | 密钥生成：4轮  签名：9轮 |  |
| GG18-EdDSA | 否 | 否 | EdDSA | 是 | 密钥生成：4轮  签名：9轮 |  |
| FROST | 否 | 否 | Schnorr | 是 | 密钥生成:3轮  签名：3轮 |  |

我们选取了四种门限签名方案与TB-ECDSA进行比较，这些方案的特性差异如表X-X所示。

我们对TB-ECDSA与这四种门限签名计算方案的签名算法耗时进行了比较。测试中，我们涵盖了四种门限签名方案，从签名者数量从4逐步增加至20，并记录了它们的运算耗时。根据图表X-X的显示结果，所有的方案的运算耗时都随签名者数量增加而增加。TB-ECDSA的运算耗时始终低于GG18-ECDSA和GG18-EdDSA。在签名者数量大于4之后，尽管TB-ECDSA的通信轮次比这两个方案更高，但随着签名者数量的增加，其与它们相比的运算耗时优势逐渐增加。FROST方案由于采用Schnorr签名的性能优势以及更低的通信轮次，在四个方案中运行耗时最短。然而，与此相比，TB-ECDSA在几个生成椭圆曲线签名的方案中的运行耗时表现与FROST最为接近。值得一提的是，TB-ECDSA是四个方案中唯一提供了盲性和用户匿名性的。

图x-x 签名算法耗时

## 本章小结

本章重点介绍了TB-ECDSA方案。首先，我们从宏观的角度介绍了该方案的几个特性，包括盲性、门限特性、签名的不可链接性、错误识别以及用户匿名性。接着，我们介绍了方案中的两个角色，即接收者与签名者。随后，我们对TB-ECDSA方案的三个主要算法进行了详细介绍，包括签名者初始化算法、接收者初始化算法以及签名算法。在签名算法中，我们从用户匿名性出发，总结出为了满足匿名性算法的设计应该受到的几个约束规则，再从这些约束总结接收者与签名者的交互模式，随后详细地阐述了签名算法的步骤以及原理。

针对提出的TB-ECDSA方案，我们进行了安全性分析，阐明了其具备的几个特性。同时，我们还从理论和实验的角度对该方案的计算性能进行了分析，并与同类型的几个门限签名方案进行了比较。结果显示，TB-ECDSA相对于GG18-ECDSA和GG18-EdDSA在门限签名方案中的运算效率更高。随着签名者数量的增加，TB-ECDSA的性能优势逐渐增加。尽管FROST方案的运行时间最短，但在门限椭圆曲线签名方案中，TB-ECDSA的性能表现最接近FROST，并且是唯一提供盲签名和用户匿名性的方案。

# 基于ECDSA的门限盲签名系统实现

## 开发环境搭建

在本研究中，我们选择Go语言作为开发工具，并以Visual Studio Code (VSCode)作为集成开发环境，以便高效地进行代码编写。系统中涉及对象间的数据交换，我们采用了Protocol Buffer（ProtoBuf）构造数据格式。另外，我们基于开源的密码学库tss-lib构建了本系统，该库提供了实现门限签名方案所需的密码学基础功能以及多方计算的系统框架，保障了系统的安全性和可靠性。这样的选择和构建为我们的研究工作提供了强大的技术支持和保障。

### Go

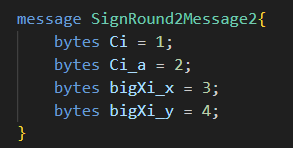
Go语言是一种由Google开发的开源编程语言，其突出的特性之一就是对并发的支持。Go语言内置了轻量级的协程，使得并发编程变得简单且高效。通过使用 go 关键字启动一个新的协程，可以轻松地实现并发操作，而不需要显式地管理线程。

### Protocol Buffer

我们采用了ProtoBuf作为数据交换格式。ProtoBuf是一种由Google开发的轻量级数据交换格式，旨在提供高效的序列化机制。其设计的目标包括跨平台、跨语言的支持、自描述性和可扩展性。

通过使用ProtoBuf，我们能够定义数据结构和消息格式，并生成相应的代码用于数据的序列化和反序列化。这种方式不仅能够减少数据传输时的开销，还能提高数据传输的效率和可靠性。

使用ProtoBuf需要首先需要使用ProtoBuf语法编写proto文件来定义消息类型。如图x-x种代码所示，这段ProtoBuf代码定义了一种消息类型，消息名称为SignRound2Message2，其中包含四个字段，名称分别为Ci、Ci\_a、bigXi\_x以及bigXi\_y。



图x-x 使用ProtoBuf语法定义消息类型示例

编写完proto文件后通过命令行或者相关插件，调用protoc.exe对指定文件进行转译。ProtoBuf会根据proto文件生成Go代码，代码中会包含数据的序列化和反序列化方法。图x-x是图x-x中定义的消息经过ProtoBuf后生成的部分Go代码。



图x-x 使用ProtoBuf生成的部分代码

### tss-lib

tss-lib[]是一个开源的密码学库，由Binance Smart Chain (BSC)团队开发和维护。它提供了实现门限签名方案所需的密码学基础功能以及多方计算的系统框架。

其中tss-lib库中的crypto包提供了实现TB-ECDSA需要的椭圆曲线密码学算法，包括倍点运算、标量乘法等操作。

表x-x crypto.ScalarBaseMult函数各参数的意义

|  |  |
| --- | --- |
| 包名 | crypto |
| 函数名 | ScalarBaseMult |
| 输入 | |
| 名称：类型 | 含义 |
| curve ：elliptic.Curve | 椭圆曲线的参数 |
| k ：\*big.Int | 一个大整数，用作倍点的乘法因子 |
| 输出 | |
| 类型 | 含义 |
| \*ECPoint | 椭圆曲线基点作k倍点运算的结果 |

如表x-x所示，调用crypto.ScalarBaseMult函数可以完成一次椭圆曲线基点的倍点运算，它接受一个椭圆曲线 curve 和一个大整数 k 作为参数，并返回一个椭圆曲线上的点的结果。

表x-x ECPoint.ScalarMul函数各参数的意义

|  |  |
| --- | --- |
| 包名 | crypto |
| 函数名 | ECPoint.ScalarMul |
| 输入 | |
| 名称：类型 | 含义 |
| k ：\*big.Int | 一个大整数，用作数量的乘法因子 |
| 输出 | |
| 类型 | 含义 |
| \*ECPoint | 该点经过k倍点运算的结果 |

如表x-x所示，调用ECPoint.ScalarMul函数可以完成一次任意椭圆曲线点的倍点运算，它接收一个大整数 k 作为参数，并返回椭圆曲线上的一个点，该点是调用ScalarMul的椭圆曲线点与k倍点运算的结果。

tss-lib库中的common包提供了获取随机数还有大素数的程序调用接口。

表x-x common.GetRandomPositiveInt函数各参数的意义

|  |  |
| --- | --- |
| 包名 | common |
| 函数名 | GetRandomPositiveInt |
| 输入 | |
| 名称：类型 | 含义 |
| Rand：io.Reader | 一个随机数生成器，用于生成随机数 |
| lessThan：\*big.Int | 生成随机数的上限 |
| 输出 | |
| 类型 | 含义 |
| \*big.Int | 一个小于 lessThan 的随机正整数 |

如表x-x所示，调用common.GetRandomPositiveInt函数可以生成一个小于给定大整数 lessThan 的随机正整数。它接收一个随机数生成器 rand 和一个上限值 lessThan 作为参数，并返回一个小于 lessThan 的随机正整数。

表x-x common.GetRandomSafePrimesConcurrent函数各参数的意义

|  |  |
| --- | --- |
| 包名 | common |
| 函数名 | GetRandomSafePrimesConcurrent |
| 输入 | |
| 名称：类型 | 含义 |
| Ctx：context.Context | 上下文对象 |
| bitLen：int | 生成的安全素数的比特位长度 |
| numPrimes：int | 要生成的安全素数的数量 |
| Concurrency：int | 并发级别，控制同时进行的生成任务数量 |
| rand io：Reader | 随机数生成器，用于生成随机数 |
| 输出 | |
| 类型 | 含义 |
| []\*GermainSafePrime | 安全素数组成的数组 |

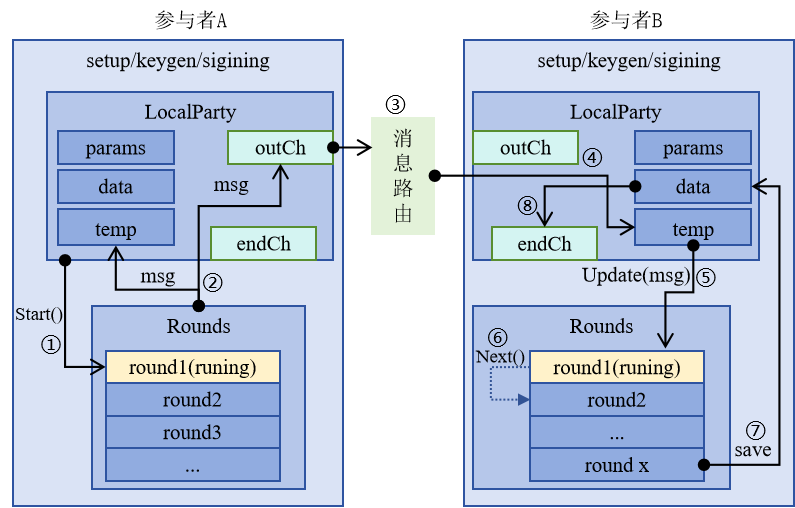
如表x-x所示，调用 common.GetRandomSafePrimesConcurrent可以并发生成安全素数。通过使用多个核心并行地生成安全素数，函数可以更快地找到多个有效的结果。

tss-lib还提供了Paillier加密方案的实现，包括密钥生成，加密，解密，同态加法，同态乘法等功能。我们在tss-lib的代码基础上做出了修改，实现了Paillier同态加密修改方案。

## 4.2 系统功能实现

### 系统框架

图x-x展示了本系统框架在完成一次消息发送/接收时的情况。我们以参与者A向参与者B发送一次消息为例，阐述本系统框架的构成以及工作过程。



图x-x 系统框架示意图

在本系统的实现中，我们定义了三个包对应TB-ECDSA方案的三个算法：

1、setup包：签名者初始化算法。

2、keygen包: 接收者初始化算法。

3、signing包：签名者初始化算法。

每个包中，都定义了两种重要的对象：LocalParty和Rounds，它们在不同的包中具体会有不用的定义，但总的来说LocalParty会有以下五类数据结构：

1. params：当前LocalParty的一些参数，包括采用的椭圆曲线，算法的参与方信息，当前配置的门限值等等。
2. data: 当前LocalParty持有的重要数据，在算法开始时从外部读取，在算法结束时持久化储存，包括一些公共参数，私有数据，签名结果等等。
3. temp：在进行算法时需要保存的临时数据，包括所有收到的消息，中间变量或是需要在后续步骤用到的数据等。
4. outCh: 用于发送传出消息的通道。
5. endCh：用于返回计算完成的data的通道。

Rounds类型中包含了当前算法每个轮次需要运行的算法步骤，分别写在round1、round2等结构中。

在图x-x中一个轮次会这样进行:

1. 当参与者A完成了LocalParty的初始化后，会调用Start()来运行第一轮round1的算法。
2. 当round1的算法运行完成后，会将要发出的消息msg保存到temp中，然后再将msg通过outCh发送出去。
3. 消息经过外部的消息路由。
4. 消息到达指定的其他LocalParty的temp中，在图x-x中接收msg的是参与者B。
5. 参与者B接收到msg后，会将msg从temp中取出调用Update(msg)来把消息更新到round1中。
6. round1收到msg后，就会调用Next()，来进行下一轮的运算。
7. 当参与者B完成了当前算法的所有轮次运算后，就会将结果保存到data
8. 接着data通过endCh传出。

### 签名者初始化模块

根据第三章3.3小节的算法，我们将签名者初始化分为三个轮次。

第一轮次，签名者选取掩盖因子分片，并进行Shamir秘密共享的相关计算，将相应的数据发送到对应的节点，然后开始监听消息，辅助签名者在第一轮直接进行消息监听。当各个签名者收到来自其他签名者的消息时，进入到下一轮次。

第二轮次，各个签名者按照算法对接受到的数据进行处理，得到掩盖因子加法分片，然后广播该数据并进入消息监听状态。当各个签名者接收到了其他签名者的广播后，进入到下一轮次。

第三轮次，从消息中提取掩盖因子加法分片，计算出掩盖因子，将掩盖因子持久化储存后广播，最后结束算法。

如表x-x根据这三个轮次的通讯的消息，我们可以定义三种消息类型，分别对应三个轮次中发送的消息。

表x-x 签名者初始化中的消息类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 消息名称 | 字段名称 | 类型 | 含义 | 对应符号 |
| SURound1Message | prime\_share | bytes | 掩盖因子分片的分片 |  |
| SURound2Message | gama\_share | bytes | 掩盖因子加法分片 |  |
| SURound3Message | prime\_mask | bytes | 掩盖因子 |  |

### 接收者初始化模块

根据第三章3.4小节我们将接收者初始化算法分为三个轮次。

第一轮次，签名者直接进入监听状态，接收者生成私钥，并对私钥进行Shamir秘密分享的计算，将产生的私钥分片发给对应的签名者，然后进入第二轮次。当签名者收到接收者的消息时进入第二轮次。

第二轮次，签名者从消息中提取出私钥分片，在本地持久化保存，并将私钥索引发回给接收者。接收者在第二轮次直接进入监听状态，等待签名者的消息。当接收者收到签名者发来的消息时，进入下一轮次。

第三轮，签名者无需行动。接收者从消息中提取出索引，并在本地持久化保存，结束算法。

如表x-x，根据这三个轮次的通讯的消息，我们可以定义两种消息类型。

表x-x 接收者初始化中的消息类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 消息名称 | 字段名称 | 类型 | 含义 | 对应符号 |
| KGRound1Message | share | bytes | 掩盖因子分片的分片 |  |
| KGRound2Message | index | bytes | 掩盖因子加法分片 |  |

### 签名算法模块

根据第三章3.5小节，我们可以将签名签名算法分为四个轮次，其中有两个轮次是需要执行多次。

第一轮次，签名者直接进入监听状态。接收者生成椭圆曲线随机数并初步计算随机点中间结果。接收者将随机点中间结果发送给其中一个签名者，然后进入监听状态。

收到消息的签名者首先从消息中提取随机点中间结果，然后选取一个随机数，将这个随机数与随机点中间结果进行倍点运算得到新的随机点中间结果，将这个新的结果发送给接收者后，进入下一轮次。

接收者收到新的数据后，重复进行第一轮次的操作，用自己的随机数再次进行倍点运算，然后发送给下一个签名者，等待下一个签名者的消息。

当接收者与所有签名者都交互了一次后，接收者进入下一轮次。

第二轮次，签名者直接进入监听状态。接收者按照算法对待签名消息进行加法分片，同态加密等操作构造出要发送给签名者的密文数据和验证数据。为每段数据确定好发送给签名者的顺序，然后将数据发送给每段数据对应的第一位签名者，然后接收者进入监听状态等待签名者的消息。

当签名者接收到本轮此的消息后，按照算法对数据进行处理，并回发处理后的数据，然后进入下一轮次。

接收者接收到所有的签名者的消息后进入下一轮次。

第三轮次，签名者直接进入监听状态。接收者从消息中提取出上一轮此签名者处理过的数据，按照算法对这些数据进行处理后将这些数据按照在第二轮次中确定的签名者顺序发给这段数据对应的下一个签名者，然后进入监听状态。

签名者接收到第三轮次的消息后，按照算法进行处理，并回发给接收者，接着进入监听状态。

接收者接收到签名者的消息后重复第三轮次开始试点操作，将处理过的消息发送给对应的下个签名者，直到所有消息都经过了所有签名者处理后进入下一轮次。签名者在这轮次也是重复监听，处理，发送的操作直到处理了所有签名消息后，签名者即完成了签名算法的所有步骤，结束算法。

第四轮次，接收者对接收到的数据进行校验，解密，聚合，得到完整的签名。接着对签名进行验证，如果验证通过，结束签名算法。如果验证不通过，则需要运行错误识别程序，排查是哪个签名者导致了签名失败。

如表x-x，根据这四个轮次的通讯的消息，我们可以定义六种消息类型。

表x-x 签名者中的消息类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 消息名称 | 字段名称 | 类型 | 含义 | 对应符号 |
| SignRound1Message1 | BigKr\_x | bytes | *kr* 对应的倍点x坐标值 |  |
| BigKr\_y | bytes | *kr* 对应的倍点y坐标值 |
| SignRound1Message2 | BigKri\_x | bytes | 随机点中间结果的x坐标 |  |
| BigKri\_y | bytes | 随机点中间结果的y坐标 |
| BigVi\_x | bytes | 验证点的x坐标 |  |
| BigVi\_y | bytes | 验证点的y坐标 |
| BigKci\_x | bytes | 承诺点的x坐标 |  |
| BigKci\_y | bytes | 承诺点的y坐标 |
| SignRound2Message1 | Cmi | bytes | 消息分片mi的密文 |  |
| Cri | bytes | r的密文 |  |
| Cmi\_a | bytes | 用于计算验证的密文 |  |
| Cri\_a | bytes | 用于计算验证的密文 |  |
| index | bytes | 私钥索引 |  |
| SignRound2Message2 | Ci | bytes | 签名中间结果的密文 |  |
| Ci\_a | bytes | 用于计算验证的密文 |  |
| SignRound3Message | Cix | bytes | 签名中间结果的密文 |  |
| Cix\_a | bytes | 用于计算验证的密文 |  |

## 4.3 系统测试

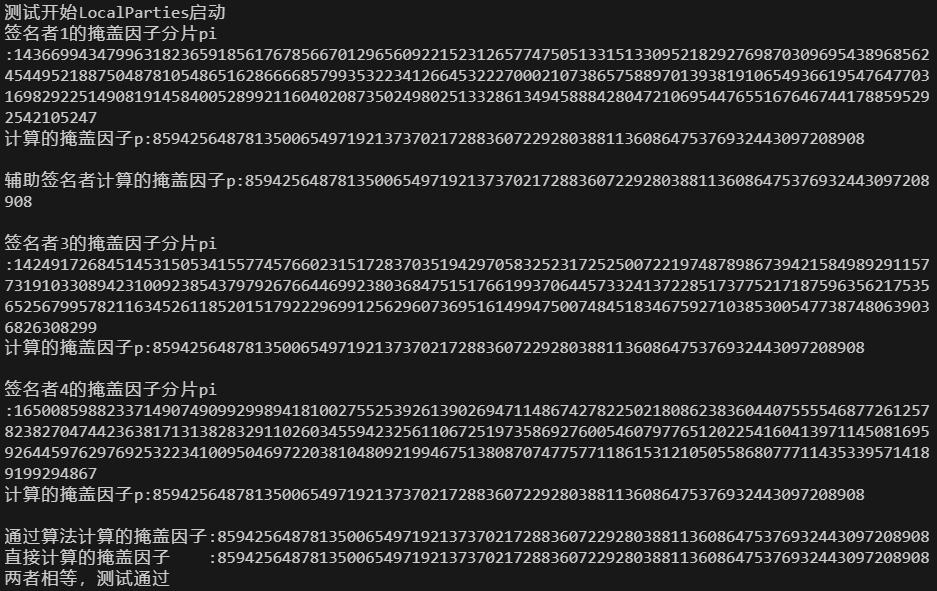
本系统在Windwos平台实现，CPU型号：ixxxxxx，内存16G，使用Go+VsCode作为开发环境。我们使用Go语言自带的测试框架进行测试，使用不同的协程来运行不同的LocalParty来模拟不同参与者进行多方计算，协程之间通过通道进行信息交换。

因为三个算法都封装到了一个相同的系统框架中，所以我们对于三个算法的测试可以采用几乎相同的方式，我们在Go语言的测试框架中编写了测试代码，这些代码依次进行以下步骤：

1. 初始化四个setup/keygen/signing包的LocalParty，其中三个为签名者，一个为辅助签名者/接收者。
2. 使用Go协程调用所有参与者的LocalParty.Start()函数启动算法。
3. 进入监听循环，进行以下操作
   1. 持续监听四个LocalParty的outCh通道，当从通道中接收到消息时，解析消息内容确定消息的目的地，通过LocalParty.Update(msg)向该LocalParty传入消息。
   2. 持续监听四个LocalParty的endCh通道，当监听到四个LocalParty都从endCh中传出算法结果时，跳出监听循环。
4. 从算法结果中提取出通过共同计算的结果，针对不同的算法结果使用不同的方式检验结果的正确性，如果检验结果正确则测试通过。

如图x-x是签名者初始化算法的测试函数运行时打印的部分日志，对于该算法结果的检验方法为：

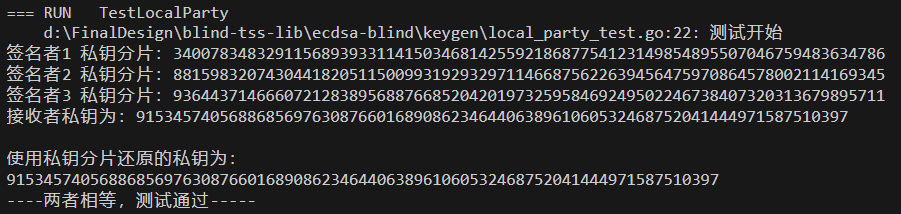
1. 检查各个签名者是否生成了相同的掩盖因子，如果是，继续后续检验步骤，否则算法测试不通过。
2. 取得各个签名者的掩盖因子分片。
3. 对这些分片累乘，直接计算得到一个掩盖因子值为理论值。
4. 将掩盖因子理论值与算法计算得到的掩盖因子对比，如果相同，则测试通过。



图x-x 初始化接收模块测试结果

如图x-x是接收者初始化算法的测试函数运行时打印的部分日志，对于该算法结果的检验方法为：

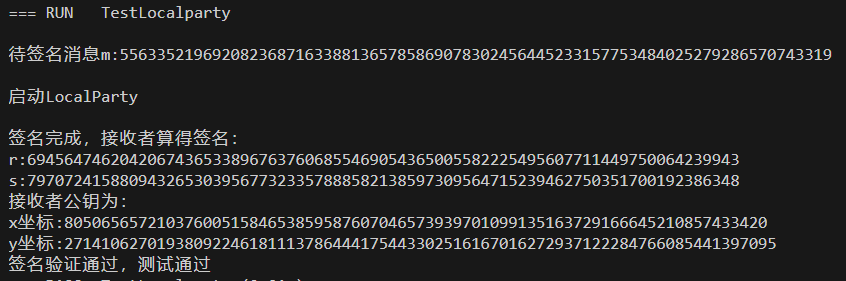
1. 取得各个签名者的私钥分片，使用Shamir秘密共享方案提供的还原方法得到一个还原私钥。
2. 将还原私钥与接收者秘密保存的私钥对比，如果相同，则测试通过。



图x-x初始化算法模块测试结果

如图x-x是签名算法的测试函数运行时打印的部分日志，对于该算法结果的检验方法为：

1、将算法结果中的签名，和接收者保存的公钥，以及待签名的消息输入到椭圆曲线签名验证算法中，观察是否通过签名验证。如果通过验证，则测试通过。



图x-x 签名算法模块测试结果

我们记录了三个算法在不同签名者数量的情况下的运行耗时，如图x-x。

(a)签名者初始化算法 (b)接收者初始化算法 (c)签名算法

图x-x各算法运行耗时

# 总结与讨论