**zk-SNARK工作****原理**

**目录**

[0. zk-SNARK的定义 1](#_Toc34399341)

[1. 从数学多项式讲起 2](#_Toc34399342)

[2. 多项式下的非交互零知识证明 4](#_Toc34399343)

[2.1 进一步观察数学多项式 4](#_Toc34399344)

[2.2 改善协议第一步：模糊化计算 6](#_Toc34399345)

[同态加密 6](#_Toc34399346)

[模运算 6](#_Toc34399347)

[2.3 改善协议第二步：限制性多项式 KEA方式幂级数运算 7](#_Toc34399348)

[2.4 改善协议第三步：让零知识更具体一点 7](#_Toc34399349)

[2.5 非交互式CRS(共相关字符串) 7](#_Toc34399350)

[3. 将零知识运用在程序中 8](#_Toc34399351)

[4. 后续需要调研的内容 9](#_Toc34399352)

[5. 参考文献 10](#_Toc34399353)

# 0. zk-SNARK的定义

zk-SNARK

（succinct non-interactive arguments of knowledge）简洁无交互知识参数证明协议

一个标准的sk-SNARK协议应当满足以下要求：

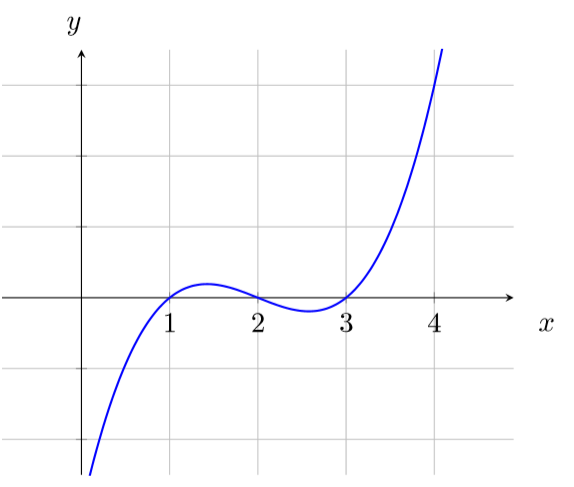
***1完备*** 声明的正确意味着Prover能够向Verifier进行证明并使其信任

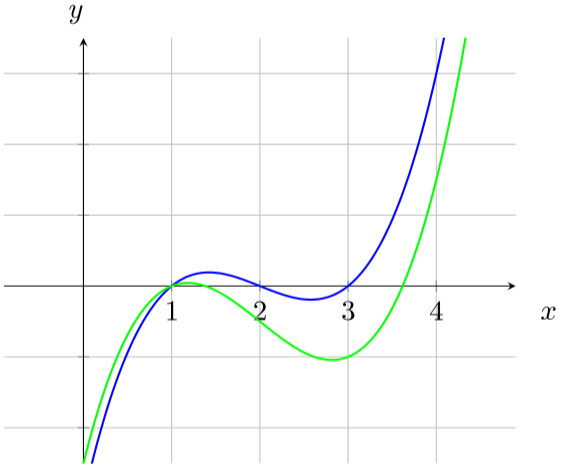
***2可靠*** Prover无法通过作弊的方式完成

***3零知识*** 交互过程中只有判断声明是否为真的信息，而不会泄露其他任何信息

# 1. 从数学多项式讲起

任何一个函数多项式都可以在坐标轴中描绘出其曲线。

例如可以描述成以下曲线

而不同的多项式所描绘的多项式曲线是不同的，例如

而两个d阶的多项式曲线，最多只可能有d个交点。例如上面两个3阶多项式，最多只可能有3个交点。因此一个零知识证明的数学多项式的**基本思路**是这样的：

* V想要寻找一个多项式
* P声称他有该多项式
* 我们需要做的事情当然就是多取几个*x*，代入多项式取验证验证*fv(x) == fp(x)*
* 当然，如前所述，就算是两个多项式不相等，那么还有会有偶然情况，取得的*xn*恰好是两曲线交点,刚好使得两个多项式计算结果相等。那么这种概率是多少呢？
* 我们假设取[1, 10^77]的范围(假设交点也在这个范围内)，那么P想用另外一个多项式曲线蒙混过关的概率就是：
* 那么这个概率可见是非常低，换句话说通过验证多项式的方式，我们能够去判断一个声明的真实性。
* 这就是zk-SNARK的核心数学工具：**数学多项式**

# 2. 多项式下的非交互零知识证明

让我们回顾一下，刚刚的数学多项式

实际上，一个数学多项式是可以线性的变成多项式因子相乘的形式：

显然，在*x=a0*, *x=a1*是其中的两个根，如果我们目前知道这一信息，定义

那么很显然，*f(x)*是*t(x)*与另一个函数的*h(x)*的乘积，也就是：

**(1)**

当前的情况就变成，我们知道t(x), 要找到h(x)，最终得到我们想要的f(x)

这样一来，刚刚我们在第一小节中提到的核心原理，目前可以做个改进：

当然，主要的还是要**证明*fp(x) == fv(x)=t(x)h(x)***

* 将上面的公式变换一下:

**(2)**

* 也就是说，我们挑选*x*的值带入这个验证等式中，我们将按照*h(x)*最后的结果来判断验证是否通过。（注意此时等式两边都是数学多项式）
* 但是此时情况和第一节中的情况不太一样，第一节中验证左右两边的多项式是否相等，有一个隐藏的前提是：我们已经知道等式右边的数学多项式了！但是这里*h(x)*恰好就是我们要寻找的，这样一来不是很矛盾吗？在这种情况如何来完成证明？

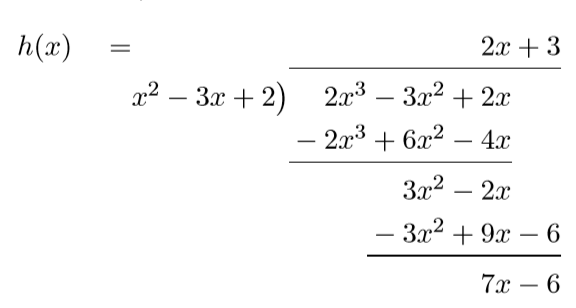
## 2.1 进一步观察数学多项式

举个例子，假设Prover持有的是这么一个函数

而Verifier持有的,代入到(2)中，我们会发现结果

**(3)。**

这叫做多项式整除。但是如果Prover持有的是另一个数学多项式呢？比如说

带入(2)中我们会发现得到多项式商2x+3和余数7x-6：

也就是说 **(4)**

比较(3)和(4)我们发现，挑选整数值进行计算，如果Prover真的持有符合预期的数学多项式，那么最终的结果也必然是一个整数！

如果Prover想要作弊，用另一个并不能满足要求的多项式尝试验证，那么当然，虽然偶然能够得到一个整数结果，但是正如第一节中所分析讨论的那样，能够碰到这种情形的概率非常低。最终作弊者会在多次的测试中败下阵来，无法通过验证。

那么目前数学多项式的零知识证明协议我们已经让它变得更具体了一些：

* V想要找一个多项式，他知道前提条件*t(x)*
* P声称其有这么一个多项式*fp(x)*
* V挑选一个随机的整数*r*,通过计算，**V观察最终的结果是否为一个整数**。如果不是，那么则说明P的声明为假

接着，在这个协议中V当然还需要做一些事情：

* P计算出*h(r),fp(r)*交给V来验证，**V计算*t(r)h(r)*的结果和*fp(r)*是否相等**
* 如果验证到这里都通过了，那么说明可以认可P的声明

在这个过程中，V挑选的只是一些随机的整数，并没有泄露其他的数据信息。P也只是给出了相应的计算结果，V也无法从这些计算结果中，得到验证通过与否的信息，其他的信息并未泄露出。

至此，多项式的零知识证明的基本协议已经完成，后续的主要工作都是围绕着如何让证明协议更加完善而进行的操作。

因为目前协议的过程存在一些问题：

* P可以因为知道随机数*r*的存在而任意伪造多项书企图蒙混过关。

比如说，P可绕过规则提供一个整数*h’*，和*p’=h\*t(r)*作为结果进行作弊，当前协议无法防范。

或者因为没有多项式阶数的限制，P进行构建更高阶的多项式尝试作弊。

因此，我们需要进一步改进

我们来回顾一下第0节的零知识证明的要求再继续下一小节的说明。

一个标准的sk-SNARK协议应当满足以下要求：

***1完备*** 声明的正确意味着Prover能够向Verifier进行证明并使其信任

***2可靠*** Prover无法通过作弊的方式完成

***3零知识*** 交互过程中只有判断声明是否为真的信息，而不会泄露其他任何信息

## 2.2 改善协议第一步：模糊化计算

上一小节的遗留问题中，Prover因为知道随机数的存在，从而试图伪造结果或者多项式。在这一小节中，我们首先解决这个问题来限制Prover

### 同态加密

同态加密取一个最简单的操作幂级数运算，例如取5为底的幂函数

如果我们需要对某个数据加密，例如3，那么

在此基础上，我们还可以进行加减操作,比如:

加运算3+2=5就变成：

减运算3-2=1就变成：

系数乘法3\*2就变成：

（注意这里3待加密的数据，2是公开的系数）

当然，仅仅使用同态加密做数据保护是不够的，如果底数被泄露，则被加密的数据就能够被Prover得到。

### 模运算

使用模运算的原因在于，不同的数字再进行模运算后的结果相同，从而对于Prover来说，无法准确知道源随机数是多少。例如：

刚刚的运算模6则会得到

加运算3+2=5就变成：

减运算3-2=1就变成：

系数乘法3\*2就变成：

将同态加密和模运算结合起来，我们可以构建一个加密函数：

**(2.2.1)**

在这个基础之上开始构建加密多项式，那么原来的多项式：

变为 **(2.2.2)**

和2.1节中的协议过程类似，我们使用类似的过程，只是提供随机数之后需要在(2.2.2)的加密多项式函数中进行计算和验证数据的操作。

如此通过同态加密运算和模运算的方式，我们已经可以做到限制Prover获取随机数源数据，但Prover仍然可以做到任意构造数据来作弊，因此下一步继续进行协议的优化。

## 2.3 改善协议第二步：限制性多项式 KEA方式幂级数运算

对于verifier来说,在取用秘密值处理后送给prover之后,无法保证prover就使用该秘密值进行运算,prover有可能使用捏造的数据进行运算后返回给verifier,只需要其能保证zp=zhT(s)的等式.因此需要限制prover使用verifier给出的秘密值s通过加入随机数的α方式,向prover添加秘密值对(s, s\*\*α),prover按照接收到的秘密值对进行运算后,返回给verifier.在这个过程中保证是由verifier提供.由此具体带入多项式中,每个多项式中的个阶数都进行运算,将各类运算结果以及附加的随机数运算结果对返回给verifier进行验证.

## 2.4 改善协议第三步：让零知识更具体一点

在上述的过程中,为了防止verifier提取prover返回结果中的系数知识,所以在prover端,也使用类似于使用随机数的方式,对计算的结果进行返回一组结果对(g, g'),从而能保证prover的知识不会泄露.无成本δ变换

## 2.5 非交互式CRS(共相关字符串)

一次交互式的零知识方案,只能够对某个prover或者verifier进行验证.,无法重复利用该方案向别人去证明这个声明.因此,对于一开始由verifier交给prover的秘密值和随机数对(s, α)需要进行重新保护,用于重复,公开使用.对于α以及秘密值s的各级幂运算变换被加密,而删除原始数据,而这些参数通常就被成为CRS.CRS生成后,就可以分成两组,分别分发给prover以及verifier.而为了保证秘密s, α确实被删除,使用秘密分发的方式,多个参与者进行共同生成随机数.再使用s的1次幂作为标准,来检验每一个次幂的结果与前一次的值是否保持一致来保证使用伪造的秘密分值.接着要求出去第一个参与者其余人公开其参数来防止尾链证明人伪造数据.

# 3. 将零知识运用在程序中

前两章节是用于说明零知识证明本身的原理，作者在第三章开始进行说明如何将程序中的运算通过零知识证明进行转化，成为一个QAP方程组多项式。

将p(x) = h(x)\*t(x)转变为操作数方程组

L(x)\*R(x) = t(x)\*h(x) + O(x) （3.1）的一个变换。

在转换成方程组的过程中，需要使用到前面章节中介绍到的同态加密，椭圆曲线对的使用来实现加密环境下的一个基本加减乘除的运算。

在一个二次方程组下进行验证的过程中，需要不断进行约束，来保证最终协议的安全性。需要在各个环节添加随机数。

在一个完整的zk-SNARK协议中，基本需要以下参数，以及其涉及的过程和作用:

* s在协议运算的一开始就需要确立，s的作用是选定好最终多项式曲线的根
* α在协议的set up 阶段用于分立的KEA检查，用于限制Prover选取的多项式的曲线参数L(x)以及R(x)的曲线在特定点位的参数设置
* β用于限制Prover选取多项式的整体一致性检验
* γ用于在方程多项式一致性检验的过程中完成零知识的功能，β是一个随选取的数值，但是仍然一定的概率，是的Verifier能够反推L(x)以及R(x)中的知识。因此需要使用一个值对β进行加密
* ρ在进行多项式一致性检验的过程中，使用β进行一致性校验，会使得计算变得复杂，多出几组椭圆曲线对的运算，因此使用了随机数ρ用于改变验证过程，减少计算量的同时保证一致性的检验

# 4. 后续需要调研的内容

QAP方程组多项式

FHE 同台加密

ECC椭圆曲线机密以及椭圆曲线加密对运算

Pinocchio协议的调研

# 5. 参考文献

[MP 19] Maksym Petkus “Why and How zk-SNARK Works: Deﬁnitive Explanation” 2019