选题来源与选题意义；与选题相关的国内外研究现状；主要研究内容及预期取得的成果；拟采取的研究方法、技术路线、实验方案及可行性研究；论文工作进度的初步安排

零、一些零碎备注

[二次方差程序：从 0 到 Hero |作者 Vitalik Buterin |中等 (medium.com)](https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649)

[探索椭圆曲线配对 |作者 Vitalik Buterin |中等 (medium.com)](https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627)

[zk-SNARKs：引擎盖下。这是系列文章的第三部分... |作者 Vitalik Buterin |中等 (medium.com)](https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6)

1. ZkSNARK 原始论文：“Pinocchio: Nearly Practical Verifiable Computation”Bryan Parno Jon Howell, Craig Gentry Mariana Raykova

2. QAP 原始论文：“Quadratic Span Programs and Succinct NIZKs without PCPs” Rosario Gennaro, Craig Gentry, Bryan Parno, Mariana Raykova

QSP/QAP问题的思想都是出自2012年一篇论文：Quadratic Span Programs and Succinct NIZKs without PCPs。论文的下载地址：https://eprint.iacr.org/2012/215.pdf。

3.关于proof和argument

**Proof**

* **Definition**: A proof is a type of demonstration where the validity of the statement is guaranteed even if the prover and verifier have unlimited computational power.
* **Example**: Mathematical proofs, such as proofs in number theory, where soundness is absolute.
* **Key Property**: It’s **information-theoretically secure**; cheating is impossible, regardless of computational power.

**Argument**

* **Definition**: An argument is similar to a proof, but the guarantee of validity holds only if the prover and verifier are bounded by polynomial-time computation (e.g., they use practical, real-world computers).
* **Example**: Cryptographic systems like zk-SNARKs, where soundness relies on computational assumptions (e.g., it’s hard to break a cryptographic hash).
* **Key Property**: It’s **computationally secure**; cheating is infeasible under standard cryptographic assumptions but not impossible with unlimited computational power

**Proofs** - 在零知识证明的场景下，Proofs指具有完美的完备性（Completeness）以及完美的可靠性（Soundness）。也就是，具有无限计算资源也无法攻破。

**Arguments** - 在零知识证明的场景下，Arguments是指具有完美的完备性以及多项式计算的可靠性。也就是，在多项式计算能力下，是可靠的

4.术语简介

**SP - Span Program** ，采用多项式形式实现计算的验证。

**QSP - Quadratic Span Program**，QSP问题，实现基于布尔电路的NP问题的证明和验证。

**QAP - Quadratic Arithmetic Program**，QAP问题，实现基于算术电路的NP问题的证明和验证，相对于QSP，QAP有更好的普适性。

**PCP - Probabilistically Checkable Proof** ，在QSP和QAP理论之前，学术界主要通过PCP理论实现计算验证。PCP是一种基于交互的，随机抽查的计算验证系统。

**NIZK - Non-Interactive Zero-Knowledge**，统称，无交互零知识验证系统。NIZK需要满足三个条件：1/ 完备性(Completeness)，对于正确的解，肯定存在相应证明。 2/可靠性 (Soundness) ，对于错误的解，能通过验证的概率极低。3/ 零知识。

**SNARG - Succinct Non-interactive ARGuments**，简洁的无须交互的证明过程。

**SNARK - Succinct Non-interactive ARgumentss of Knowledge**，相比SNARG，SNARK多了Knowledge，也就是说，SNARK不光能证明计算过程，还能确认证明者“拥有”计算需要的Knowledge（只要证明者能给出证明就证明证明者拥有相应的解）。

**zkSNARK - zero-knowledge SNARK**，在SNARK的基础上，证明和验证双方除了能验证计算外，验证者对其他信息一无所知。

**Statement** - 对于QSP/QAP而言，某个计算电路的输入。Statement对证明者和验证者都是公开的。

**Witness** - Witness只有证明者知道。可以理解成，某个计算电路的输出（output）。

5.零知识证明的几种构造

zkSNARK：将 PCP 转化为 QSP 只是众多零知识证明协议中的一种构造方式，基于同态公钥密码学的一种构造。

Bulletproofs 是基于离散对数问题；

STARK 是基于 IOP (Interactive Oracle Proof) 的。

6.

argumentSNARG和argument of knowledge

一、零知识证明基础

**1.0 零知识证明**

**zkSNARK**，**z**ero-**k**nowledge **S**uccint **N**on-interactive **AR**guments of **K**nowledge的简称：

* Succinct：证明的数据量比较小
* Non-interactive：没有或者只有很少交互。
* ARguments：验证者只对计算能力有限的证明者有效。拥有足够计算能力的证明者可以伪造证明。这也叫“计算可靠性"（相对的还有”完美可靠性"）。
* of Knowledge：对于证明者来说在不知道证据（Witness，比如一个哈希函数的输入或者一个确定 Merkle-tree 节点的路径）的情况下，构造出一组参数和证明是不可能的。

零知识证明大体由四部分组成：

* **多项式问题的转化** - 需要证明的问题转化为多项式问题 t(x)h(x) = w(x)v(x)，证明者提交证明让验证者确认多项式成立。
* **随机挑选验证** - 随机选择验证的数值s，验证t(s)h(s) = w(s)v(s)。相对于验证多项式相等t(x)h(x) = w(x)v(x)，随机挑选验证，简单，验证数据少。随机挑选验证，安全性肯定不及多项式等式验证，但如果确实足够随机，安全性还是相当高的。
* **同态隐藏** - 同态隐藏指的是函数的一种特性。输入的计算和输出的计算保持“同态”。以加法同态为例，满足如下的三个条件的函数E(x)，称为加法同态：1. 给定 E(x)，很难推导出x. 2. 不同的输入，对应不同输出 3. E(x+y) 可以由 E(x)，E(y)计算出来。乘法同态类似。
* **零知识** - 证明者和验证者之间除了“问题证明与否”知识外，不知道其他任何知识（不知道随机挑选值，不知道挑选值的多项式计算结果等等）。

**约化**(Reduction)，可以理解成问题的转化。对任意一个程序A的输入，都能按某种法则变换成程序B的输入，使两程序的输出相同，那么，可以说，问题A可约化为问题B。

**P问题**指的是在多项式时间内可解的问题。

**NP问题**（Non-Deterministic Polynomial Problem，非确定性多项式问题），指不能在多项式内可解，但是可以在多项式时间内验证的问题。一般认为，NP问题不等于P问题，也就是说，NP问题不存在多项式解法。

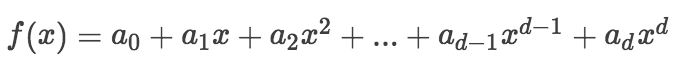
**NPC问题**，是一个NP问题，并且，其他所有的NP问题都能归约到它。简单的说，NP问题之间可以相互归约，一个NP问题求解，其他NP问题一样能求解

1/ 问题转化:  一个需要证明的NP问题转化为选定的NP问题（比如QSP问题）

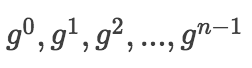
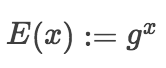
2/ 设置参数（setup）：设置参数的过程也是挑选随机数的过程，并提供CRS

3/ 证明者获取证据u，通过CRS计算证据（proof）

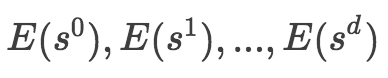
4/ 验证者验证证据以及响应的proof

假设一个多项式。证明一个多项式，即给定一个，提供的证明。

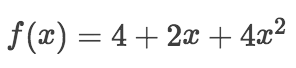
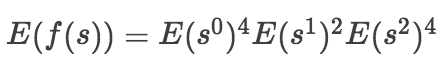
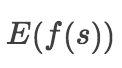
**3.1 有线群论基础（椭圆曲线）**

定一个有限群，生成元图片，阶，则该群包括如下的元素：。通过有限群加密的方式很简单：。也就是说，得知图片的情况下，不能反推。

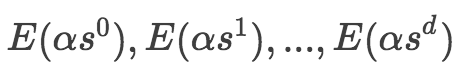
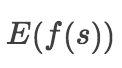
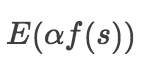
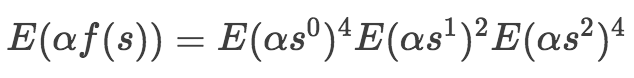
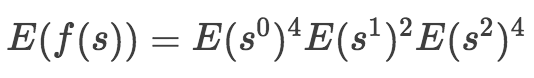
**3.2 选定随机数**

验证者随机选择一个有限群中的元素，图片。提供如下的计算结果（图片不同阶的加密结果）：  
在生成这些计算结果后，图片不需要了，可以忘记。

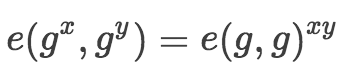
**3.3** 图片**计算**

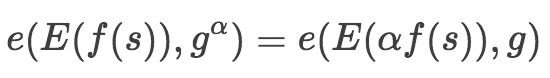
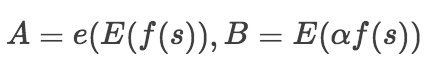
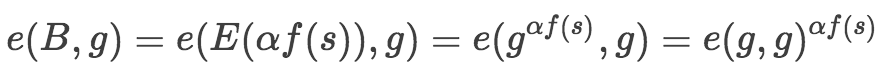
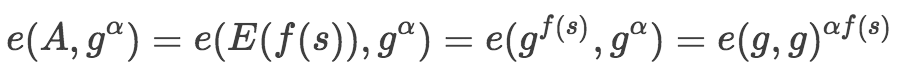
举个例子，，。显然， 可以不知道图片 情况下，通过验证者提供的数据计算出来。

**3.4**

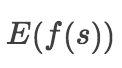
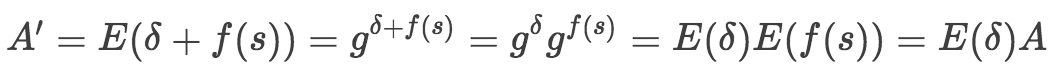
注意的是，验证者是不知道待证明的多项式参数的，即使证明者提供了图片，验证者也无法验证。的方法可以让验证者确认证明者是通过多项式计算出结果。在3.2的基础上，验证者还随机选择另外一个图片，并提供额外的计算结果：  
证明者需要提供和。  


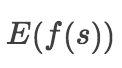
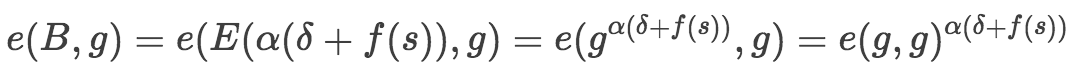
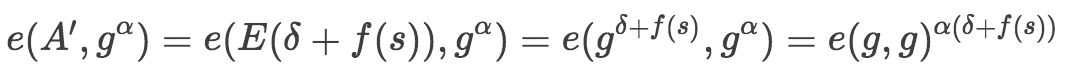
**3.5 配对函数**

配对，满足如下定义：  


验证者验证的方式很简单，检验如下的等式是否成立：  
假设推导过程如下：  
  
到此为止，验证者提供的情况下，证明者可以证明通过某个多项式计算出某个结果，验证者不知道具体的多项式的参数。

**3.6**

更进一步，证明者采用 ，甚至不想让验证者知道。采用 ，证明者不再提供，而是随机一个，提供。  
图片

很显然，验证者从推导出，但验证者一样能验证的配对函数是否成立：  


KEA

给定一组 满足（这 个称作偏移量），在不知道的情况下，如果有人能够给出另外一组数使得，那么一定有(b 一定是 a的指数线性组合)。

* V 提供加密值和他们的偏移
* P根据多项式系数计算多项式加密结果：，计算多项式的偏移加密结果，发送给V
* V校验

多项式的 SNARK

当证明着和验证者达成一致选取了目标多项式 t(x) 和证明者的多项式阶数 d：

• Setup (多方计算运行)

– 挑选随机值 r, α

– 计算加密值和 ，

– 生成 proving key：

– 生成 verification key:

**1.1多项式整除问题**

已知 3 组阶小于 n 的多项式 $u\_0(x),u\_1(x),...,u\_m(x)$; $v\_0(x),v\_1(x),...,v\_m(x)$; $w\_0(x),w\_1(x),...,w\_m(x)$；一个阶为 n 的目标多项式 z(x), z(x) = 0 的解为 $r\_1,...,r\_n$

对于任意解 $r\_j$，3 组多项式的线性组合为 0：

$0 = \left(\sum\_{i=0}^ms\_i\cdot u\_i(r\_j)\right) \cdot \left(\sum\_{i=0}^m s\_i \cdot v\_i(r\_j)\right) - \left(\sum\_{i=0}^ms\_i \cdot w\_i(r\_j)\right)$

求向量 $\vec{s}=(1, s\_1, ..., s\_m)$满足以下整除关系：

$z(x) | \left[\left(u\_0(x)+\sum\_{i=1}^ms\_i\cdot u\_i(x)\right) \cdot \left(v\_0(x)+\sum\_{i=1}^m s\_i \cdot v\_i(x)\right) - \left(w\_0(x)+\sum\_{i=1}^ms\_i \cdot w\_i(x)\right)\right]$

* 能被z(x)整除的是向量对三组多项式的线性组合得到的线性组合多项式
* 向量  的维度为 m，如果每一个元素  的取值空间 > 2，则将线性组合多项式称为二次算法多项式，简称 **QAP**。如果每个元素的  的取值空间为 0 或 1，则将线性组合多项式称为二次扩展多项式，简称 **QSP**
* 目标多项式 z(x) =0有 n 个解，而线性组合多项式=0的解数量为 2n - 2，所以除了 z(x) 还会有 n - 2 个其它解，其它解构成**商多项式** h(x)。

目标多项式 z(x) 和 QAP/QSP 之间的整除关系满足单向性，不可快速逆向求解。但能快速验证向量  的正确性，构成 NP 问题。

**1.2 零知识证明协议**

令 R 为一个高效可计算的二元运算关系。对于二元组 ， 为声明， 为秘密。对于二元运算关系 R，证明系统包含系统参数生成 SysGen，证明 P 和验证 V。

 包含 5 个多项式时间算法：系统参数生成、承诺、挑战、响应和验证。

 协议具有如下性质

* 完备性：正确的数据一定被验证通过





* 知识提取鲁棒性：对于同一套随机数生成的 2 个秘密验证 2 次，则验证方可以提取秘密



* 诚实验证方零知识：计算有限，无法提取知识



满足以上性质的  协议是一个  协议。

**1.3  协议**

****协议包括:(1)生成系统参数CRS,(2)承诺,(3)挑战,(4)响应,(5)验证。其中,系统参数CRS由多方安全计算生成。

系统参数CRS:椭圆曲线群的阶为q,生成元为;

公开输入为。证明方P证明知道秘密ω且ω满足离散对数关系。

**交互式零知识证明**

承诺:证明方P选择随机数r,计算并发送:;

挑战:验证方V选择随机数,发送:;

响应:证明方P计算并发送:;

验证:验证方V验证

如果等式成立,则接受,否则拒绝。

**非交互式零知识证明**

承诺:证明方P选择随机数r,计算：;

挑战:证明方P计算随机数;

响应:证明方P计算,证明方发送:

验证:验证方V计算,然后验证

如果等式成立,则接受,否则拒绝。

证明方P证明其知道满足离散对数关系。由于即是天然的NP问题,又是天然的离散对数关系。所以,可以直接基于这个NP问题在椭圆曲线离散对数点上构造线性关系,形成离散对数困难。

验证方V验证了NP问题的解的正确性,但是证明方P没泄露秘密。

**1.4 zkSnark 思想**

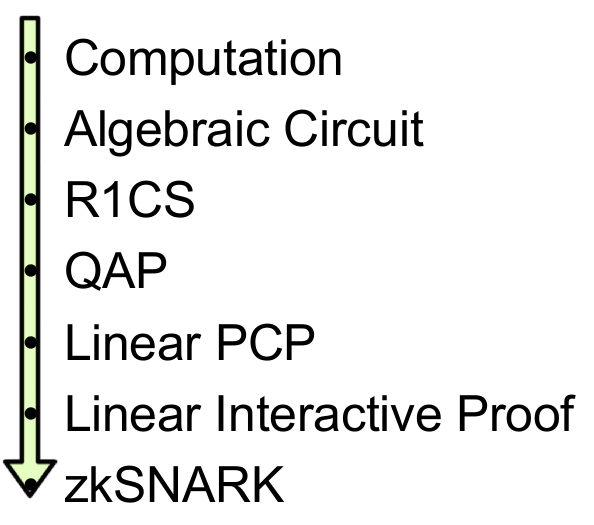
1. 证明方：基于向量  和电路，生成 QAP/QSP 多项式、目标多项式 z(x) 和商多项式 h(x)，构造 NP 问题；然后基于这 3 个多项式的系数，计算离散对数点（多项式承诺），对外暴露离散对数点
2. 验证方：对离散对数点进行双线性映射，重构整除关系，则能快速验证向量  的正确性，确不知道向量 

zk-SNARK 协议：证明秘密  满足任意多项式时间计算关系 

为了不泄露秘密 ，需要用 R1CS 约束（电路约束）等价描述算法的运算规则。

zk-SNARK 协议的等价转化：

1. 将  满足公开的计算关系  等价转化为  满足公开的 R1CS 约束
2. 等价转化为向量  与电路多维向量 的内积（基于  算出 ，基于 R1CS 约束算出多维向量）
3. 等价转化为向量  与电路矩阵 的内积
4. 等价转化为向量  对 3 组多项式  的线性组合运算
5. 等价转化为目标多项式 z(x) 整除 QAP/QSP 多项式（构成 NP 问题，向量  放到 QAP/QSP 中）
6. 等价转化为基于这 3 个多项式（z(x)、QAP/QSP、h(x)）的系数计算椭圆曲线离散对数点（构成零知识），对外暴露离散对数点。



**1.5 zkSnark 中的转化**

**1.5.1多项式时间算法等价转为 R1CS**

方程  可拆分为如下阶为 1 的等式： 



 满足方程的解，等价于  满足电路约束。

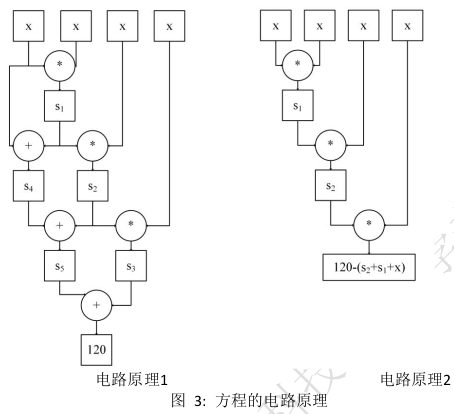
以上 6 个阶为 1 的等式包含 3 个乘法门和 3 个加法门，可以将加法约束优化到乘法约束中：



如果加法门无法耦合到乘法门，则将加法门变为乘法门：



最终R1CS 以乘法门为标准门。



**1.5.2 R1CS 转为向量内积**

 与向量  等价。

证明：令 。已知 ，可以逐步算出 ，从而构造向量 

公开数据为 statement = (1, out)，保密数据为 witness = 

对于  可进行如下向量内积等价转化：



令 ，则有：



如下阶为 1 的等式（R1CS 约束）



可进行如下向量内积等价转化：





令; ，则有：



**1.5.3转为向量与矩阵的内积**

令



则有



**1.5.4转为向量对三组多项式的组合运算**

将矩阵中的元素当做多项式的值，例如对于矩阵 



对于多项式，每列有 3 个函数值，所以取任意 3 个横坐标 ，可使用拉格朗日插值或快速傅里叶变换等方式计算多项式的系数。

则多项式系数表达 

同理，有



则有



 称为向量  对三组多项式的组合运算，运算结果称为二次算法多项式，简称 QAP 多项式。

将所有的门电路，转化为多项式表达

**1.5.5目标多项式整除 QAP 多项式构造 NP 问题**

上述拉格朗日插值多项式引入横坐标 ，则能够构造多项式



 称为目标多项式，也可以选择任意横坐标变量，例如 ，则重新使用拉格朗日插值计算函数值，并令 

当 ，目标多项式  和 QAP 多项式  均为 0

因为 QAP 多项式阶高于目标多项式 z(x)，所以 QAP 还有其它解，所以 z(x) 是 QAP 的因子，换言之，目标多项式  和 QAP 多项式为整除关系：



证明方不公开向量 ，而是使用  构造出的 QAP 多项式除以目标多项式 z(x)，得到商多项式 h(x)



然后将 QAP 多项式、目标多项式 z(x) 和商多项式 h(x) 的系数放到椭圆曲线离散对数上（多项式承诺），生成证明（离散对数点）。验证方获得离散对数点，基于双线性映射重构整除关系，验证了向量的正确性，不知道解向量 ，即零知识。

**1.6 zk-SNARK 协议框架**

zk-SNARK 协议包括以下 10 个步骤：

1. （初始化 a）生成与电路无关的局部系统参数 CRS1
2. 证明方 P 证明拥有  且  满足任意运算关系 R
3. 证明方 P 证明拥有  且  满足 R1CS 约束
4. 向量  与多维向量的内积 
5. 向量  与矩阵的内积 
6. 向量  对多项式进行组合运算 
7. 目标多项式  整除 QAP 多项式  构成 NP 问题
8. （初始化 b）基于电路多项式  生成局部参数 CRS2
9. 证明方 P 将 QAP 多项式、目标多项式、商多项式放到离散对数点上，对外暴露离散对数点，实现零知识
10. 验证方 V 重构整除关系，验证向量  的正确性，但并不知道向量 

**Groth16 vs. PLONK**

Groth16

CRS 包含 R1CS 等价转化的电路多项式 ，非常具体

优点：证明方 P 使用 CRS 中的电路多项式  生成证明，速度很快

缺点：CRS 包含的电路  是由 R1CS 转化而来的，已经固化，只能表达唯一运算电路，不能表达其它运算电路，所以表达能力差。如果对 layer2 的电路修改，则步骤 6 中的初始化 b 局部 CRS 也要修改，则 layer1 的合影参数 CRS 也需要修改。

PLONK

CRS 中不包含 R1CS 等价转化的电路多项式 ，只包含大量的离散对数点。

* 优点：CRS 表达能力强，能够用于任意多项式时间电路生成证明
* 缺点：CRS 表达能力太强，R1CS 约束力不够，不足以防止证明方 P 作弊，所以引入额外的线约束。线约束计算复杂度相对较高，导致证明生成缓慢。

二、Groth16

[**260.pdf (iacr.org)**](https://eprint.iacr.org/2016/260.pdf)

**NILP（non-interactive linear proofs）的定义：**

(1).  ：设置过程，生成, 。

(2).  ：证明过程，证明过程又分成两步：

a. 生成线性关系，其中ProofMatrix是个多项式算法。

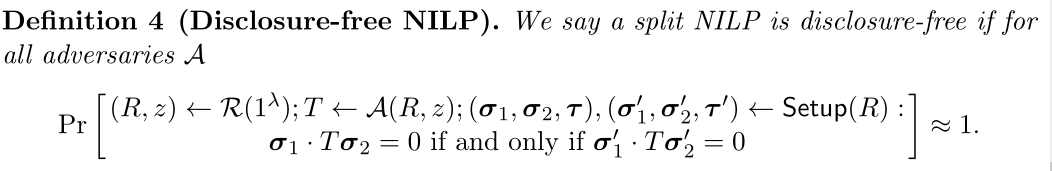
b. 生成证明：。

(3).  ：验证过程，验证者使用生成电路t，并验证是否成立。

在NILP定义的基础上，Groth16进一步定义了**split NILP**，也就是说，CRS分成两部分，证明者提交的proof也分成两部分。



希望作弊的prover不能通过generic 群运算拆分NILP，但她还是会通过CRS学习到知识并选择Π。为了对抗这种类型的攻击者，我们将定义一个无公开的公共引用字符串，在这个字符串中，证明者不会获得有用的信息来帮助她选择一个特殊的矩阵Π。



总的来说，核心在“Linear”上，证明者生成的证明和CRS成线性关系。

**构造NAR（non-interactive arguments）**

**1. QAP的NILP**

QAP的定义为"Relation"：。也就是说，statements为， witness为，并且的情况下，满足如下的等式：



的阶为n。

(1). 设置过程：随机选取，生成。





(2). 证明过程：随机选择两个参数，计算

是3×（m+2n+4）的矩阵







(3). 验证过程：

验证过程，计算如下的等式是否成立：



注意，设置过程中的x是一个值，不是代表多项式。在理解证明/验证过程的时候，必须要明确，A/B/C的计算是和CRS中的参数成线性关系（NILP的定义）。在明确这一点的基础上，可以看出和的参数能保证A/B/C的计算采用统一的 参数。因为会包含子项，要保证和相等，必须采用统一的参数。参数和增加随机因子，保证零知识（验证者无法从证明中获取有用信息）。参数保证了验证等式的最后两个乘积独立于的参数。

**完备性证明（Completeness)：**

完备性证明，也就是验证等式成立。

=……



**零知识性**

真实证明和模拟证明都具有均匀随机场元素A、B，这些元素通过验证方程唯一地确定C，因此真实证明和模拟证明具有相同的概率分布

**可靠性证明 (Soundness):**

**【for any affine prover strategy 仿射证明策略with non-negligible success probability we can extract a witness.】需要verifier确信prover确实掌握了某些秘密信息**

(如果prover能够成功说服verifier，则其必须拥有实际的witness。

如果不能提取witness，就可能出现欺骗性证明，即prover生成了一个有效proof，但实际上并不知道满足约束条件的witness。这会破坏协议的可靠性。)

Groth16算法证明的是statistical knowledge soundness，

假设证明者提供的证明和CRS成线性关系。也就是说，证明A可以用如下的表达式表达(A和CRS的各个参数成线性关系）：



域元素和阶数分别为n−1和n−2的多项式A(x), Ah(x)，分别对应于矩阵Π的第一行

同理，B/C都可以写成类似的表达：





从Schwartz-Zippel 定理，我们可以把A/B/C看作是的多项式。观察 这个验证等式，现在把验证方程看作是多元洛朗多项式的等式,根据Schwartz-Zippel引理，证明者的成功概率可以忽略不计，除非当将A， B和C视为不确定的α， β, γ, δ， x中的形式多项式时验证方程成立。发现一些变量的限制条件：

1）  (等式的右边没有)

不失一般性，可以假设。

2）  （等式右边）

不失一般性，可以假设。

3） (等式的右边没有因子)

也就是。

在上述三个约束下，A/B的表达式变成：





4）等式的右边没有

不失一般性，

5）等式的右边没有



不失一般性，。

6）等式的右边没有， 





所以，



7）等式的右边没有和



所以，。

在上述七个约束下，A/B的表达式变成：





再看验证的等式：





观察，因为不存在 ，所以，。

也就是说，

。

代入验证等式，由验证方程中包含的其余项和包含的其余项，可以推导出：

，



定义，对于，则有





代入A/B，可以获取以下等式：



在证明和CRS线性关系下，所有能使验证等式成立的情况下，等式必须成立，这个式子是statement的一个witness。也就说，能提供正确证明的，肯定知道witness。

**QAP的NIZK Arguments**

从QAP的NILP可以演化为QAP的NIZK Arguments。也就是说Groth16算法并不是完美的可靠，而是多项式计算情况下可靠。QAP的定义为"Relation"：。也就是说，在一个域中，statements为， witness为，并且的情况下，满足如下的等式（的阶为n）：



也就是说，三个有限群, 对应的生成元分别是。为了方便起见，也为了和论文的表达方式一致，有限群的计算用表示，的计算用表示。

1). 设置过程：随机选取，生成。是G1上的点，是G2上的点







标蓝色是Vk。其他部分是Pk。可以看出，Vk的大小取决于公共输入的变量个数，相对来说数量比较小。Pk的数据量大小和所有的变量个数相关。

2). 证明过程：随机选择两个参数r和s，计算

选择随机数 ， ，基于向量  和 CRS，构造线性组合关系，并将组合多项式的系数放到椭圆曲线离散对数点上，形成离散对数困难问题。对外暴露 3 个离散对数点 









 迫使采用同一条向量  参数

随机数为增加随机性，否则验证方可以求出 

3). 验证过程：验证如下的等式是否成立。



很容易发现，验证过程的等式也可以用4个配对函数表示:



//

使用椭圆曲线点重构整除关系



验证者只需要知道



4）

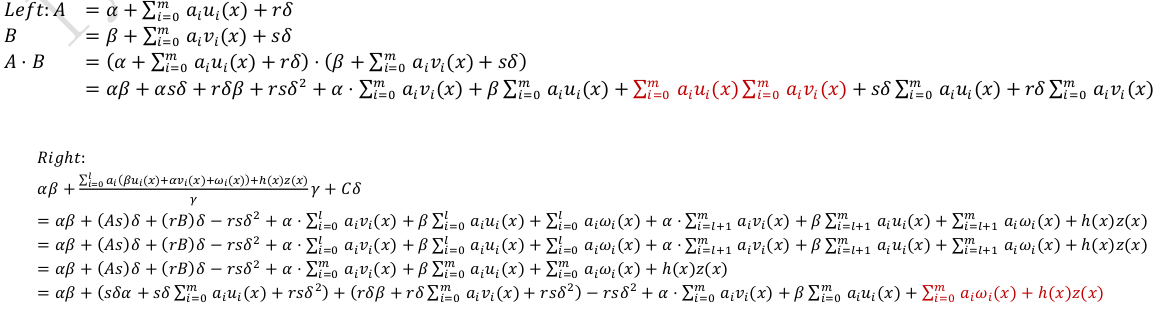
计算proof=的模拟

证明过程和QAP的NILP的证明过程类似

上述协议是一个非交互的零知识参数，具有完全完备性和完全零知识。它对仅使用多项式数量的一般双线性群操作的对手具有统计知识可靠性。



推导过程



完备性性分析

如果验证方通过了验证，则说明以上红色的部分一定相等，即



说明 z(x) 与 QAP 多项式  满足整除关系，则说明证明方知道该 NP 问题的解 。

**证明方无法作弊**

证明方只要修改任意一个参数，都无法使得验证等式成立。

**安全性分析**

1. 从验证方角度，模拟器生成证明与证明方生成证明不可区分。
2. 模拟器没有秘密，验证方使用模拟器提供的证明计算不出任何秘密。
3. 验证方基于证明方生成的证明，也计算不出任何秘密，实现零知识。

三、国密算法

3.1 SM2

**Weierstrass方程**



其中是满足某些简单条件的域上元素。

椭圆曲线有一个特殊的点，记为,它并不在椭圆曲线上，此点称为**无穷远点**。

一条椭圆曲线是由全体解再加上一个无穷远点构成的集合



由Weierstrass方程给出的定义在域上的两个椭圆曲线和



被称为在域上是**同构**的，若存在，使得变量变换



把方程变成，该变换称为**变量的相容性变换**

* 如果域的特征不等于2或3，相容性变换为：



将椭圆曲线变换为曲线：



其中，且曲线的判别式为

* 如果域的特征是2，且，相容性变换为：



把椭圆曲线变换为曲线



其中，这样的曲线称为非超奇异的，且判别式为 

* 如果域的特征是2，且，相容性变换为：



把椭圆曲线变换为曲线



其中，这样的曲线称为超奇异的，且判别式为 

* 如果域的特征是3，且，相容性变换为：



其中

把椭圆曲线变换为曲线



其中这样的曲线称为非超奇异的，且判别式为 

* 如果域的特征是3，且，相容性变换为：



把椭圆曲线变换为曲线



其中这样的曲线称为超奇异的，且判别式为 

**加法运算法则**

给定椭圆曲线  上的两个点  和 ，它们的加法  如下计算：

1.点和的直线斜率：



如果  和  是相同的点，即 ，则斜率  为切线在该点的斜率。

如果  和  不同，即 ，则斜率  通过以下公式计算：



2. 计算交点  的坐标：





如果  和  不同，上述方程成立。

如果 ，则交点  的坐标通过以下方式计算：





3. 最后，。

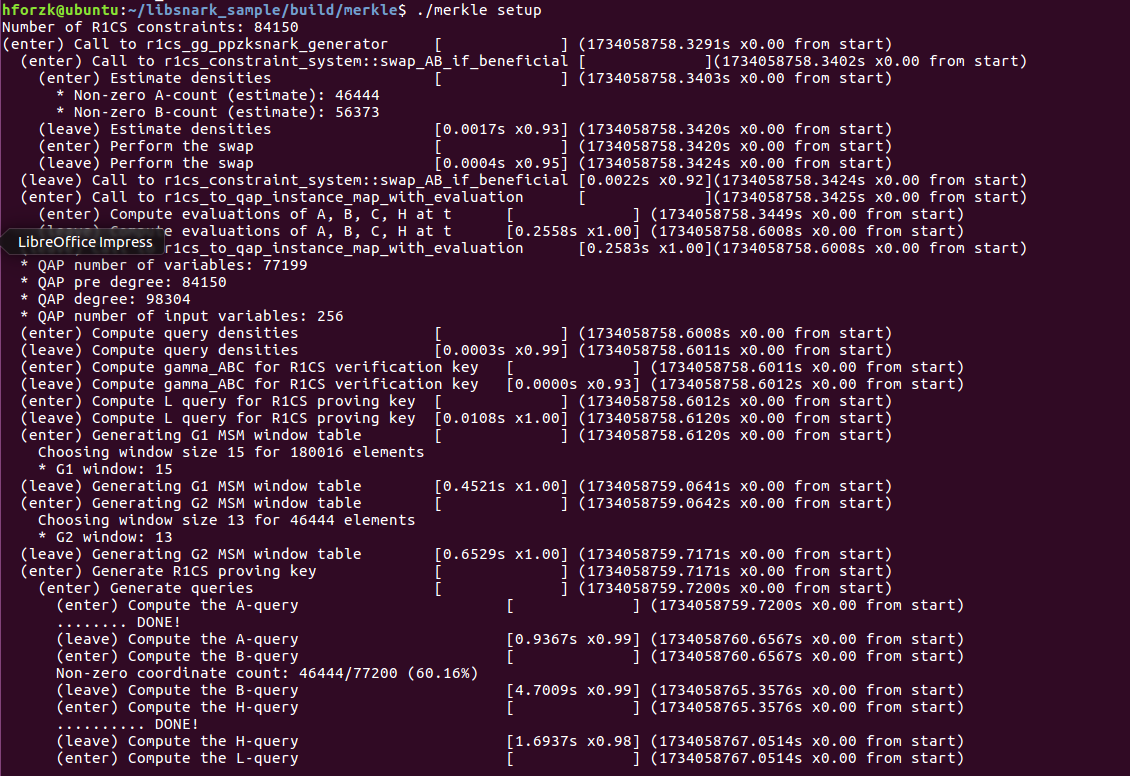
[国密SM9基于身份的密码 (gmssl.org)](http://gmssl.org/docs/sm9-cn.html)

四、libsnark

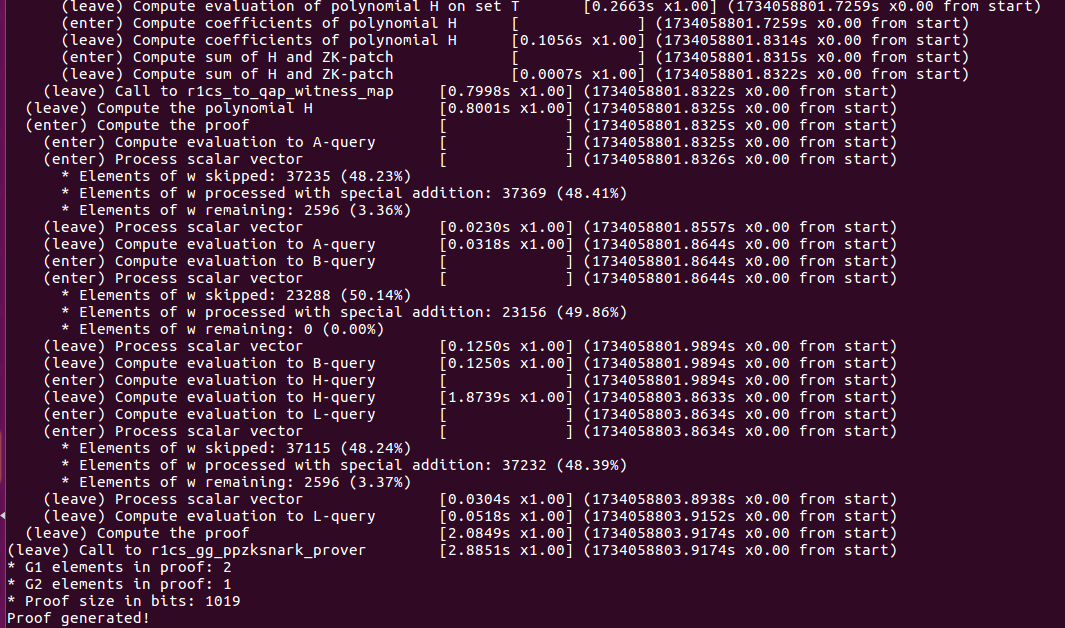
[**零知识证明 Learn by Coding：libsnark 入门篇 - SECBIT Blog**](https://secbit.io/blog/2020/01/03/zkp-learn-by-coding-libsnark-101/#5-%E5%86%8D%E6%AC%A1%E4%B8%8A%E6%89%8B%E5%AE%9E%E8%B7%B5)

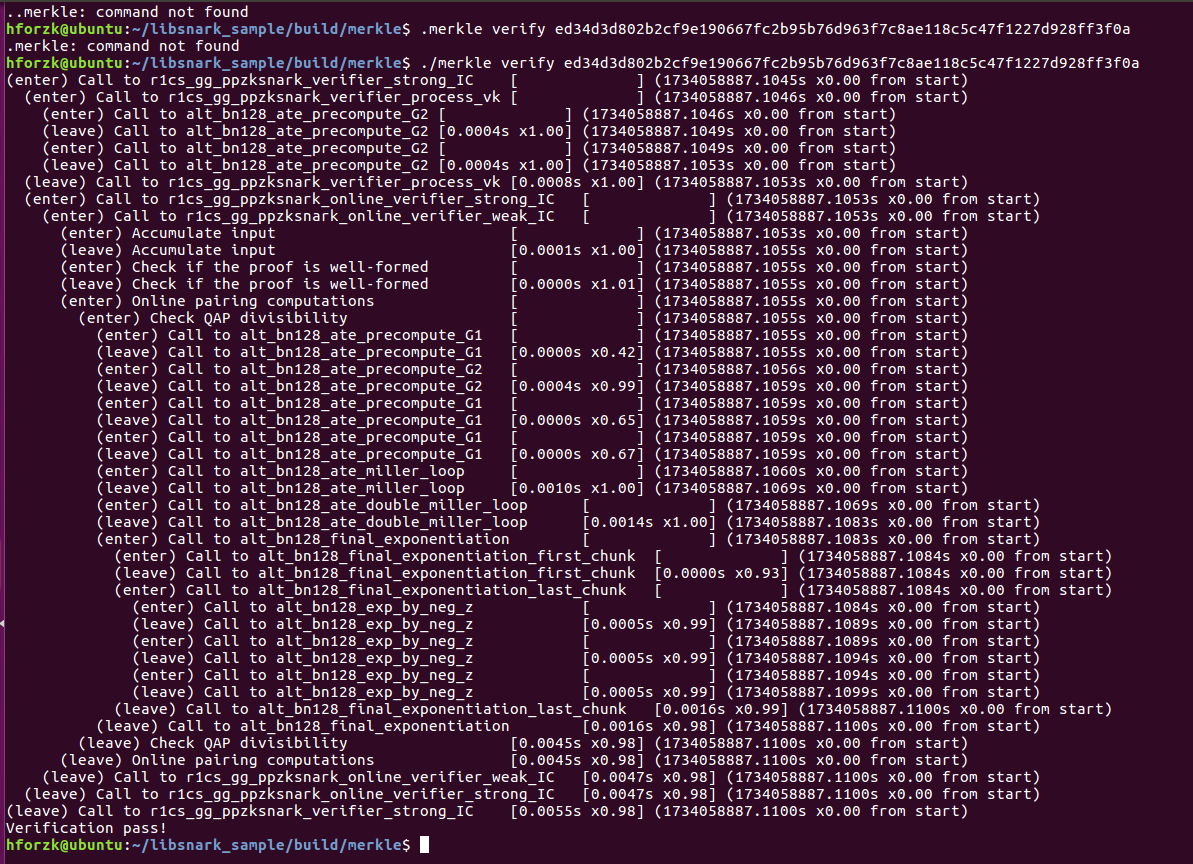
[**07 - libsnark源代码解析和实战\_哔哩哔哩\_bilibili**](https://www.bilibili.com/video/BV1qA411t7QP/?spm_id_from=333.337.search-card.all.click&vd_source=68db081a81eac7dc70e43fea643034c6)

[**Repository search results (github.com)**](https://github.com/search?q=libsnark&type=repositories)





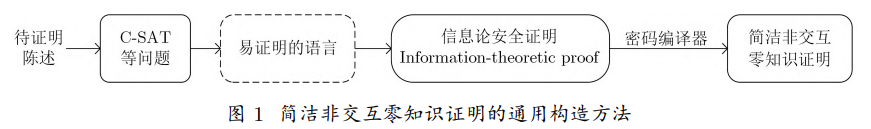




ALT\_BN128, BN128, EDWARDS, MNT4, MNT6

五、综述学习

[简洁非交互零知识证明综述\_李威翰.pdf](file:///D:\WeChat\WeChat%20Files\wxid_jfusijtgb7hf22\FileStorage\File\2024-12\%E7%AE%80%E6%B4%81%E9%9D%9E%E4%BA%A4%E4%BA%92%E9%9B%B6%E7%9F%A5%E8%AF%86%E8%AF%81%E6%98%8E%E7%BB%BC%E8%BF%B0_%E6%9D%8E%E5%A8%81%E7%BF%B0.pdf)



* 1. 将待证明陈述归约为 C-SAT 问题.

C-SAT 问题是 NPC 问题, 大多数实际问题都可用电路形式表达,

故现有的简洁非交互零知识证明的待证明陈述表示形式大都为 C-SAT 问题.

* 1. 将 C-SAT 问题转换为易证明的语言.

例如, 在基于 Linear-PCP 的零知识证 明中, 需将 C-SAT 问题转化为 QAP 可满足问题, 即判断是否存在一个多项式能够被某个公开多项式整除;

在基于 IPA 的零知识证明中, 需将 C-SAT 问题中所有的线性约束和乘法门约束归约 为判断一个多项式是否为零多项式的问题;

在基于 DEIP 的零知识证明中, 需将 C-SAT 问题转换为多元多项式的求和验证问题.

* 1. 针对易证明的语言构造信息论安全证明. 许多简洁零知识证明都是基于信息论安全证明构造的, 例如, 第一个简洁零知识论证 [15] 就是基于 PCP 构造的; 基于 MPC-in-the-Head 的零知识证明 则是首先利用 MPC-in-the-Head 构造零知识的 PCP (或 IPCP、IOP), 然后调用合适的 MPC 协 议构造的; Bitansky 等人 [58] 和 Setty 等人 [62] 也分别指出, 基于 QAP 的零知识证明本质上是 基于 Linear-PCP 构造的.
  2. 利用密码编译器将信息论安全证明转换为简洁非交互零知识证明.

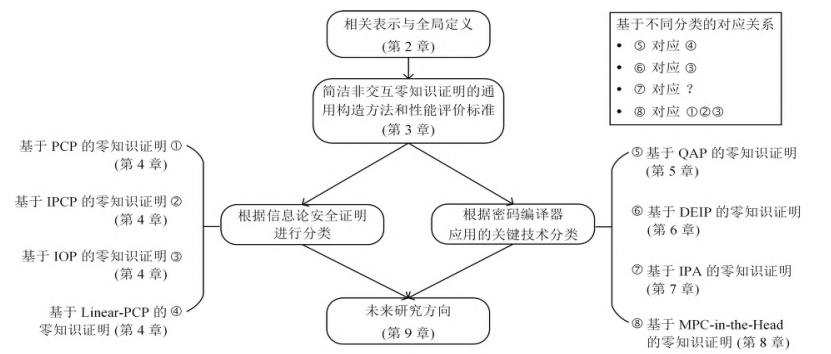
密码编译器的作用包括:

• 实现谕示. 信息论安全证明需要理想谕示 (Ideal Oracle), 而在实际的交互式证明中理想谕示是不存在的, 因此需借助承诺、哈希等密码工具实现. 由于这些密码工具大都是基于计算意义上的难题假设 (如 Pedersen 承诺的绑定性是基于 DLOG 假设的), 因此**证明也往往被削弱为论证**.

• 实现非交互. **根据实现非交互所基于的模型, 分为基于 CRS 模型和基于 ROM 的零知识证明**. 例如, 基于 QAP 的零知识证明均基于 CRS 模型实现非交互, 而基于 DEIP、IPA 和 MPCin-the-Head 的零知识证明均基于 ROM 实现非交互. 需要指出的是, 即使是基于 ROM 实现 非交互的零知识证明, 也可能需要公共参考串, 如 ZKvSQL [51]、Libra [26] 等.

• 实现零知识. 由于信息论安全证明本身可能不具备零知识性, 因此可能需利用密码编译器实现零知识性. 例如, 基于 DEIP 和 IPA 的零知识证明是通过承诺的**隐藏性及盲化多项式**实现的零知识性, 基于 MPC-in-the-Head 的零知识证明是通过 MPC 协议的隐私性实现的零知识性.

• 降低通信复杂度. 部分密码编译器有助于降低通信复杂度, 例如, 基于 DEIP 的零知识证明可通过多项式承诺降低通信复杂度, 基于 MPC-in-the-Head 的零知识证明可通过选取合适的底 层 MPC 协议降低通信复杂度.



5.1 基于 PCP、Linear-PCP、IPCP 和 IOP 的零知识证明

* 概率可验证证明PCP：

NP = PCP(log n, 1)、证明谕示、随即上述、允许访问谕示数

* 线性 PCP 与线性交互式证明LIP

利用 Linear-PCP 可自然构造两轮 LIP.

* 交互式谕示证明 IOP
* **基于PCP的ZK**

Kilian92/Micali94 ：基于 PCP 利用默克尔树和 CRHF 构造简洁的交互式 论证；利用 ROM 下的 Fiat-Shamir 启发式将上述协议修改为非交互

ZKBoo/ZKB++/KKW18

* **基于 Linear-PCP 的ZK**

zk-SNARK (包括 GGPR13[19]、Pinocchio[20]、Groth16[52]、GKMMM18[41] 等) 均是基于 Linear-PCP 实现的。首先将 Linear-PCP 转换为 LIP, 再将 LIP 转换为 SNARK, 最后将 SNARK 转换为 zk-SNARK

* **基于 IPCP 的ZK**

Ligero其也是利用 MPC-in-the-Head 的思想直接构造了零知识的 IPCP

* **基于 IOP 的zk**

在 ROM 下 IOP 可以被转换为非交互论证, 并可利用已 有的零知识编译方法, 构造系统参数可公开生成的简洁 NIZKAoK。基于不同底层关键技术构造 IOP, 如 sum-check 协议、RS 码、MPC-in-the-Head 等 可构造性能不同的简洁 NIZKAoK

基于DEIP的zk

Aurora

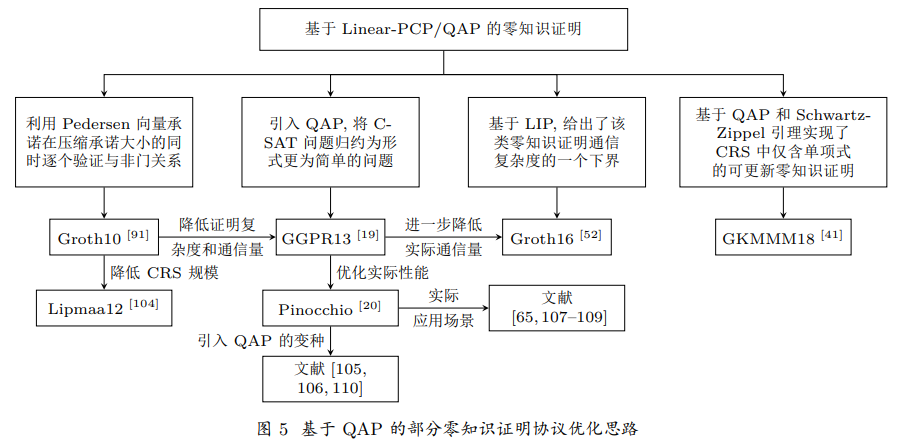
Stark

Limbo

5.2 基于QAP的zk

* **公开可信预处理 SNARG**

是算法三元组 ∏= (Setup, Prove, Verify).。满足完备性可靠性高效性简洁性

* **二次算术程序QAP**
* **q - PKE 假设、q - PDH 假设、(q - SDH 假设**
* 
* **Pinocchio**
* **Groth16**
* **GKMMM18**

5.3 基于 DEIP 的零知识证明

该类协议的核心思路是利用同态承诺、 多项式承诺、随机掩藏多项式等技术将已有的针对电路求值问题的 DEIP 转换为简洁 NIZKAoK.

* **多项式承诺**

Schwartz-Zippel 引理可将针对多项式约束的证明转化为针对多项式上某一点的证明

**6.0**

国密算法是我国自主研发创新的一套数据加密处理系列算法。SM2由国家密码管理局于2010年12月17号发布SM2是非对称加密算法，是基于椭圆曲线密码的公钥密码算法标准，其秘钥长度256bit，包含数字签名、密钥交换和公钥加密，用于替换RSA/DH/ECDSA/ECDH等国际算法。可以满足电子认证服务系统等应用需求，。SM2采用的是ECC 256位的一种，其安全强度比RSA 2048位高，且运算速度快于RSA。

SM3是一种密码杂凑算法

用于替代MD5/SHA-1/SHA-2等国际算法，适用于数字签名和验证、消息认证码的生成与验证以及随机数的生成，可以满足电子认证服务系统等应用需求，于2010年12月17日发布。

它是在SHA-256基础上改进实现的一种算法，采用Merkle-Damgard结构，消息分组长度为512bit，输出的摘要值长度为256bit。

SM9是基于标识的非对称密码算法

用椭圆曲线对实现的基于标识的数字签名算法、密钥交换协议、密钥封装机制和公钥加密与解密算法，包括数字签名生成算法和验证算法，并给出了数字签名与验证算法及其相应的流程。并提供了相应的流程。可以替代基于数字证书的PKI/CA体系。

SM9主要用于用户的身份认证。据新华网公开报道，SM9的加密强度等同于3072位密钥的RSA加密算法，于2016年3月28日发布。

zk-SNARK现成为最流行的零知识证明形式,例如Zcash使用zkSNARKs技术隐藏交易双方地址和交易金额，同时仍能验证交易的合法性。Ethereum 通过 zkSNARKs 实现私密智能合约，允许用户在不公开合同细节的情况下执行交易。

七Plonk细读

约束电路

[零知识证明 - PLONK电路原理 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/343211926)

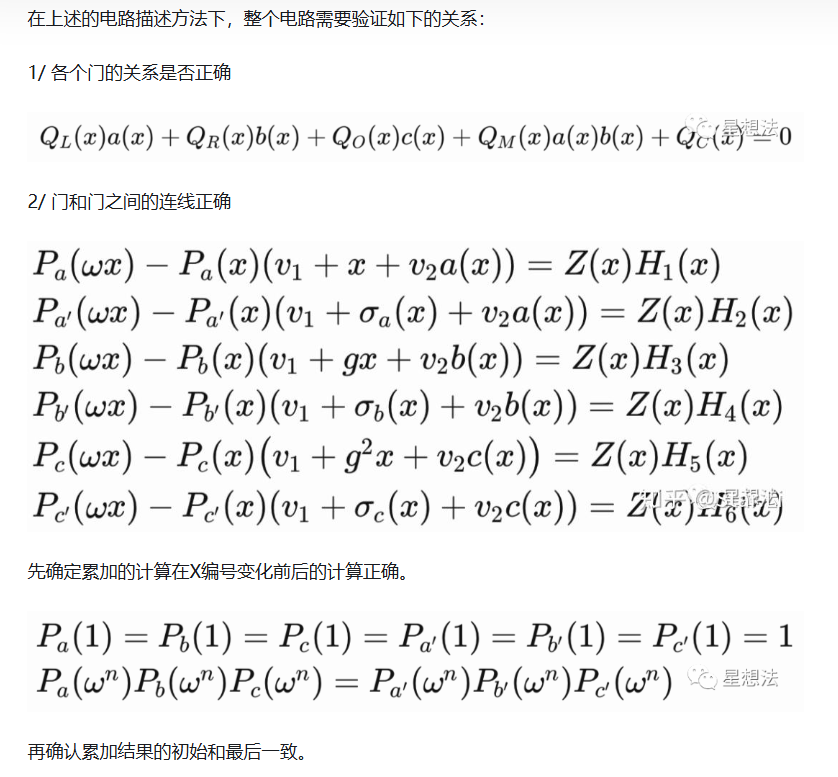
[Coordinate Pair Accumulator - SotaZK Labs docs](https://docs.sotazk.org/terms/coordinate_pair_accumulator/)

理解：wire线是有3n条的，每组n个，可以编出三组n个编号，2n个gate input和n个gate output，但是这之中会有重叠（这个门的output是另一个门的左input），所以不全是变量。

Xa(x) = x, Xb(x) = n+x, Xc(x) = 2n+x

X(x)函数是门编号到线编号的映射（域n->3n）

Y(x)函数是到这个门操作的值的映射



[Blockchain and cryptography （区块链与密码学）学习笔记6：Plonk证明系统 - Dora Dōjo - Dora Community Forum (dorahacks.io)](https://community.dorahacks.io/t/blockchain-and-cryptography-6-plonk/206)明天翻译细学Plonk

八、电路编程语言

[ZK 语言调查：Noir , o1js , Circom , Leo, Cairo, Lurk | 登链社区 | 区块链技术社区 (learnblockchain.cn)](https://learnblockchain.cn/article/6855)

领域特定语言（DSL） 将抽象概念与证明系统所需的电路表示桥接起来。

[Circom 语言教程与 circomlib 演示 | 登链社区 | 区块链技术社区 (learnblockchain.cn)](https://learnblockchain.cn/article/6811)是一个简单教学，明天看看

[iden3/snarkjs：zkSNARK 在 JavaScript 和 WASM 中的实现 (github.com)](https://github.com/iden3/snarkjs)

这是 zkSNARK 和 PLONK 方案的 JavaScript 和纯 Web Assembly 实现

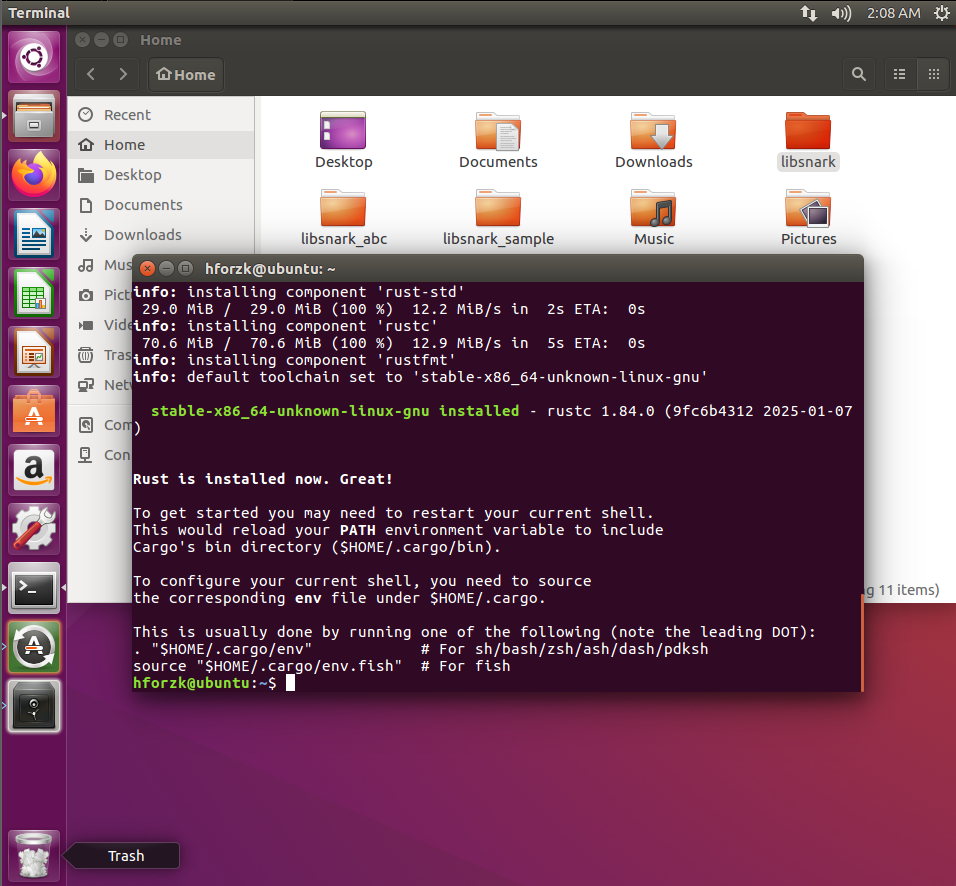
[GitCode - 全球开发者的开源社区,开源代码托管平台](https://gitcode.com/gh_mirrors/sn/snarkjs/?utm_source=artical_gitcode&index=bottom&type=card&&isLogin=1)

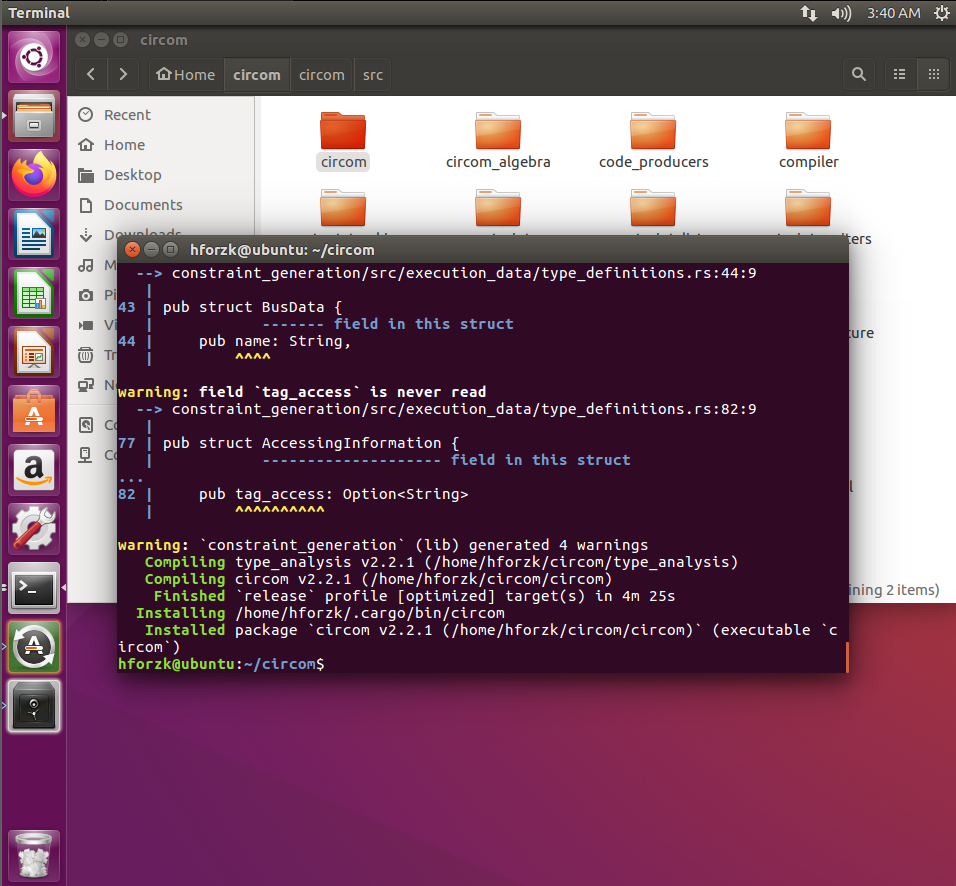
[ZK-SNARKS | 创建第一个零知识snark电路\_密钥 zkey-CSDN博客](https://blog.csdn.net/simmel_92/article/details/119514688?ops_request_misc=&request_id=&biz_id=102&utm_term=snarkjs&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-5-119514688.142%5ev101%5econtrol&spm=1018.2226.3001.4187)好像很简单，看看他们的实现过程

[circomlib/circuits/sha256 at master · iden3/circomlib (github.com)](https://github.com/iden3/circomlib/tree/master/circuits/sha256)学长给的那个sha256

LD\_LIBRARY\_PATH=/home/user/glibc-2.35/build/:/home/user/glibc-2.35/build/nptl/:/home/user/glibc-2.35/build/dlfcn/:$LD\_LIBRARY\_PATH cp

[Installation - Circom 2 Documentation](https://docs.circom.io/getting-started/installation/#important-deprecation-note)





**Node安装根据[ZK-SNARKS | 安装Node.js\_zksnarkjni库安装-CSDN博客](https://blog.csdn.net/simmel_92/article/details/119515522)**

**还看了其他有的没的用的博客[Node.js — Download Node.js® (nodejs.org)](https://nodejs.org/zh-cn/download/)**

**2024.2.14学习circomlib**

circomlib 是 **Circom 语言的标准库**，提供了一些 **预构建的电路**（比如哈希函数、Merkle 树、签名验证等），用于构建**零知识证明（zkSNARKs）**。

在 circomlib 的 doc/ 目录下，有 .monopic 文件，主要是**用于可视化 Circom 代码中的电路结构**。这些 .monopic 文件通常是 **Monodraw 生成的 ASCII 电路图**，目的是帮助开发者理解 **Circom 电路的逻辑**。

**circomlib 里的 .monopic 文件有什么作用？**

这些 .monopic 文件的作用大致包括以下几点：

1. **可视化 Circom 代码中的电路**
   * Circom 电路涉及大量 **信号（wires）** 和 **约束（constraints）**，手写代码可能不容易理解。
   * .monopic 文件提供了**电路的 ASCII 图**，方便开发者理解信号如何传递。
   * 这些图可能展示了**哈希计算**、**Merkle 树路径验证**或其他 zk 相关的操作。
2. **作为 Circom 电路的文档说明**
   * doc/ 目录通常存放**电路的文档**，包括 .md 文件和 .monopic 。
   * .monopic 可能用于**配合 Markdown 说明文档**，帮助解释某些电路的结构和工作方式。
3. **在 macOS 上用 Monodraw 直接编辑**
   * 这些文件可以在 macOS 上用 **Monodraw** 打开，开发者可以编辑、修改或优化电路图。
   * 这样可以帮助团队成员快速理解 zkSNARK 证明的结构，而不需要深入阅读 Circom 代码。

**举个例子：电路的 .monopic 可能画了什么？**

假设 Circomlib 里有个 **Merkle 树验证电路**，它的 .monopic 可能包含类似这样的 ASCII 电路图：

[叶子节点]

|

+--------+--------+

| |

[Hash] [Hash]

| |

+--------+--------+

|

[Root]

这个图可以帮助你理解 **Merkle 树** 是如何计算哈希的，并在 Circom 里实现零知识证明。

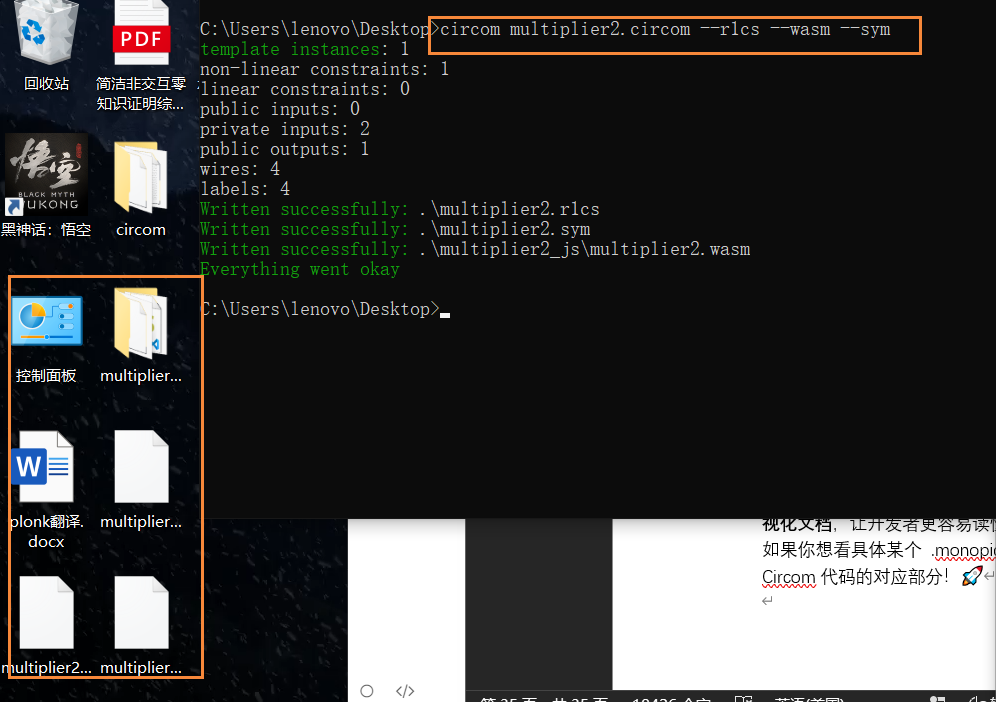
**为什么 Circomlib 需要 .monopic 文件？**

1. **Circom 代码是数学公式的实现，难以直观理解**。
2. **电路结构很复杂**，用 ASCII 图可以帮助开发者快速上手。
3. **Monodraw 让 macOS 用户可以方便地编辑和维护这些图**。

**总结**：Circomlib 里的 .monopic 文件 **不是 Circom 代码的一部分**，而是**帮助理解电路的可视化文档**，让开发者更容易读懂 Circom 代码。

如果你想看具体某个 .monopic 对应的 Circom 电路代码，可以告诉我文件名，我帮你找 Circom 代码的对应部分！🚀

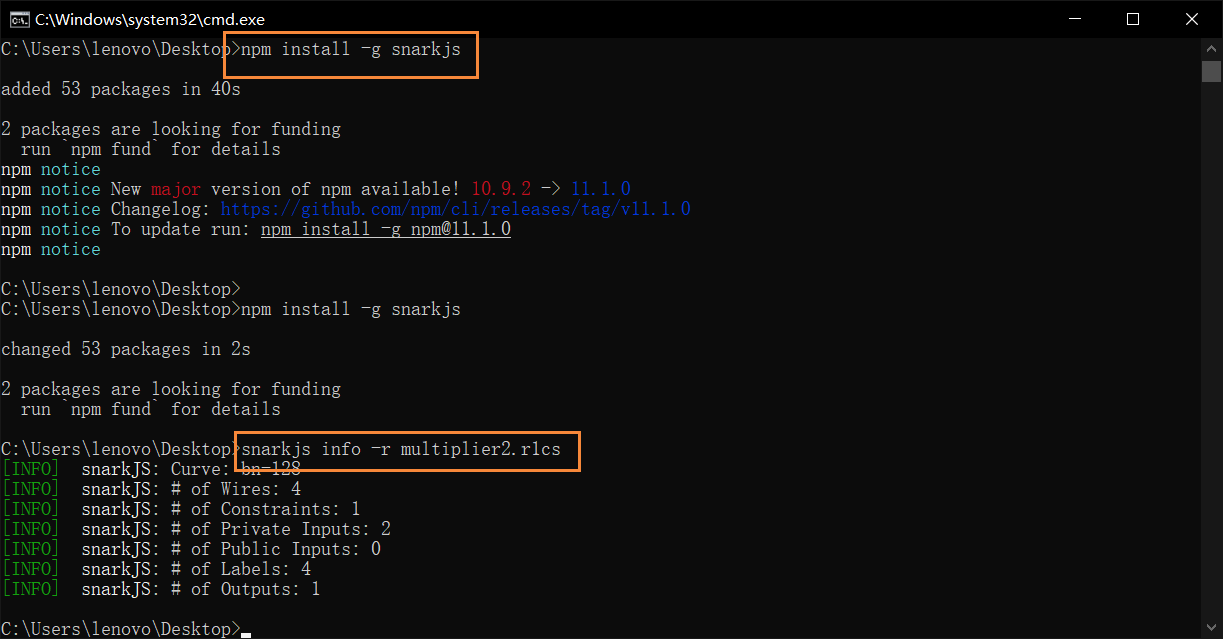
执行[构建你的第一个零知识 snark 电路（Circom2） - Foresight News](https://foresightnews.pro/article/detail/7104)



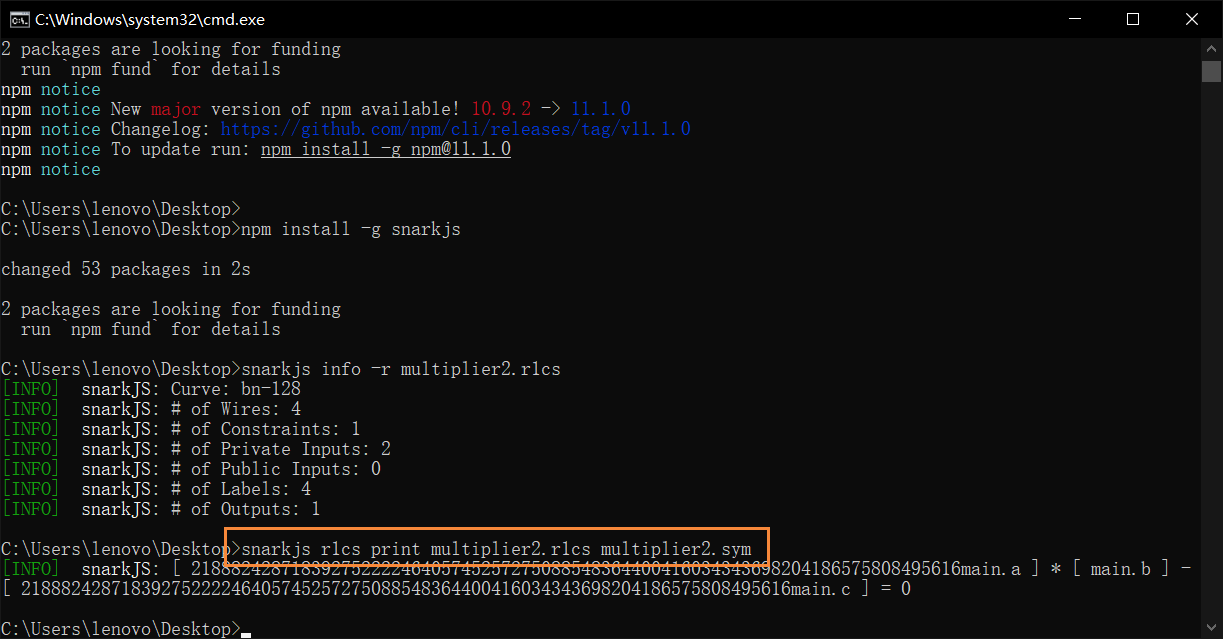
circom multiplier2.circom --r1cs --wasm --sym

使用这些选项，我们生成三种类型的文件：

* --r1cs：生成 multiplier2.r1cs ( [R1CS](https://docs.circom.io/background/background#rank-1-constraint-system) 电路的二进制格式的约束系统）
* --wasm：生成 multiplier2\_js 目录其中包含Wasm 代码(multiplier2.wasm) 和生成[见证](https://docs.circom.io/background/background" \l "witness" \t "_blank)所需要的其他文件
* --sym：生成 multiplier2.sym（以注释方式调试和打印约束系统所需的符号文件）



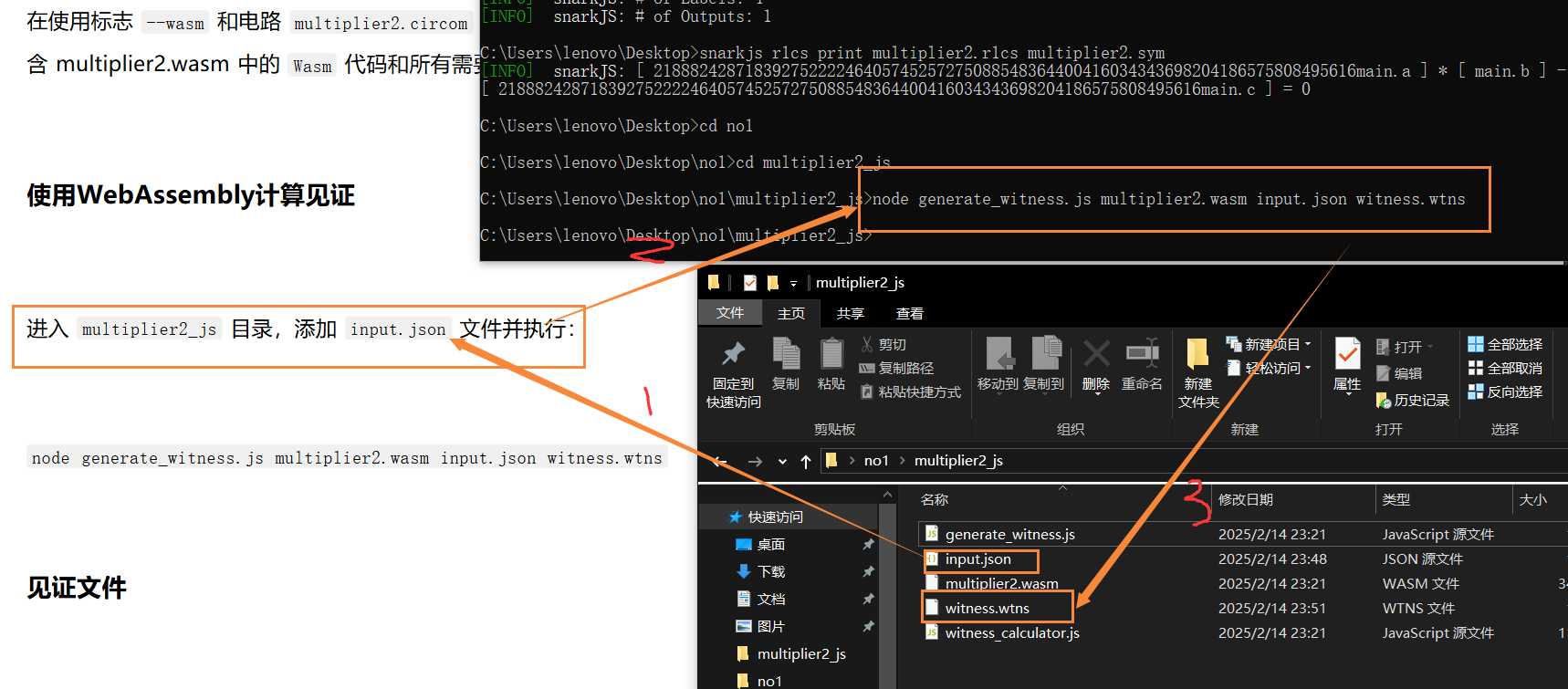
要显示电路的信息，运行：snarkjs info -r multiplier2.r1cs



可以再检查一遍，通过运行以下命令来打印电路的约束： snarkjs r1cs print multiplier2.r1cs multiplier2.sym

忽略前缀，可以读为：

a\*b-c=0



计算见证并生成二进制文件 witness.wtns，其中包含 snarkjs接受的格式。

在使用标志 --wasm 和电路 multiplier2.circom 调用 circom 编译器后，我们可以找到 multiplier2\_js 文件夹，其中包含 multiplier2.wasm 中的 Wasm 代码和所有需要的 JavaScript 文件。

生成 ẁitness.wtns 文件， 该文件以与 snarkjs兼容的二进制格式编码，这是我们用来创建实际证明的工具。

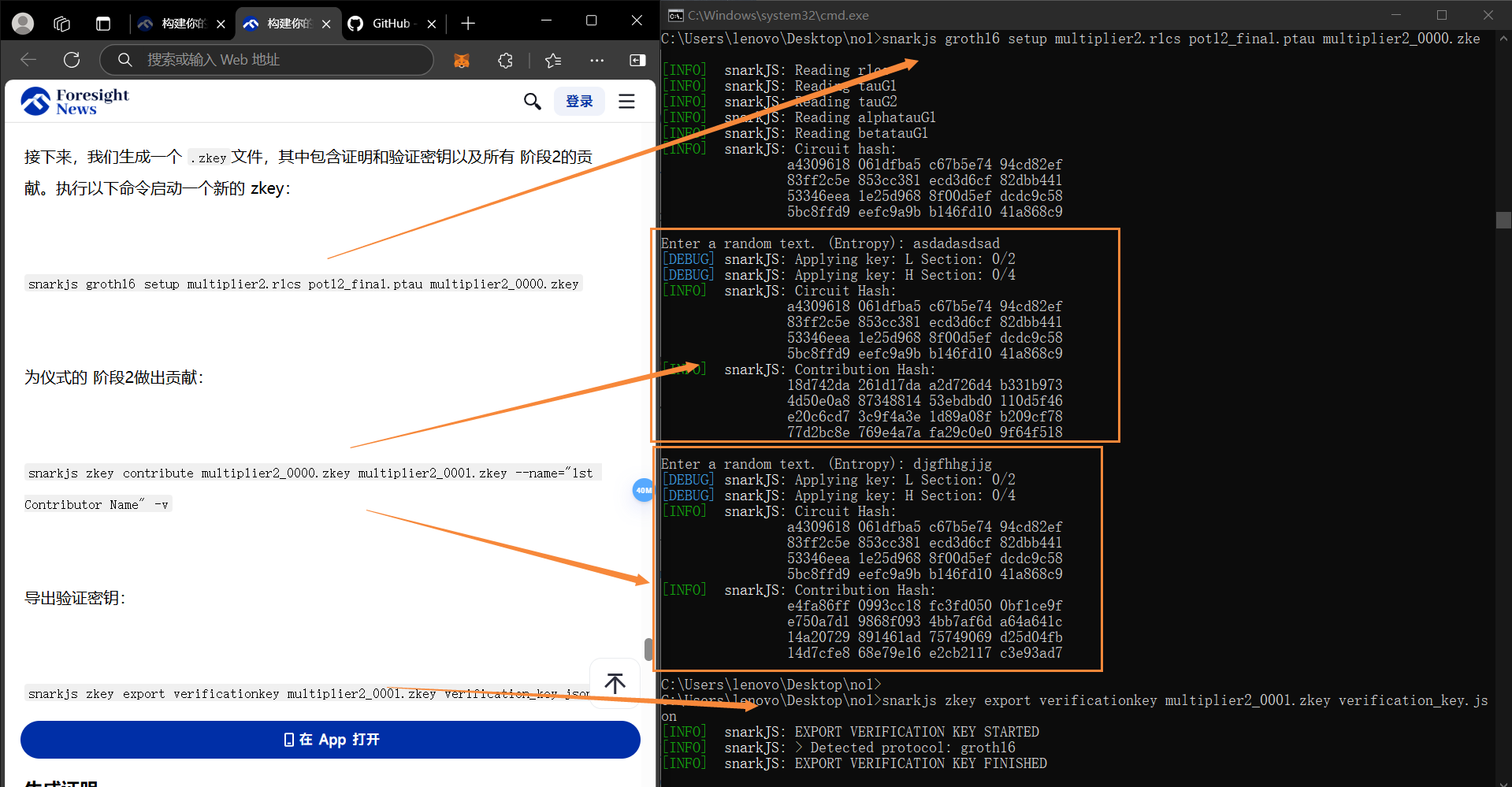
使用groth16进行验证



我们将使用 [Groth16](https://eprint.iacr.org/2016/260) zk-SNARK 协议。要使用此协议，你需要生成可信设置（[trusted setup](https://docs.circom.io/background/background" \l "trusted-setup" \t "_blank)）。 **Groth16 需要为每个电路生成可信设置。**更详细地说，可信设置由两部分组成：

* tau 的权力，它独立于电路。如上图设置（我们在文件 pot12\_0001.ptau 中有对 tau 权力的贡献，下面，我们就可以继续进行阶段 2
* 阶段2，取决于电路。





circom multiplier2.circom --r1cs --wasm –sym

snarkjs info -r multiplier2.r1cs

snarkjs r1cs print multiplier2.r1cs multiplier2.sym

node generate\_witness.js multiplier2.wasm input.json witness.wtns

snarkjs powersoftau new bn128 12 pot12\_0000.ptau -v

snarkjs powersoftau contribute pot12\_0000.ptau pot12\_0001.ptau --name="First contribution" -v

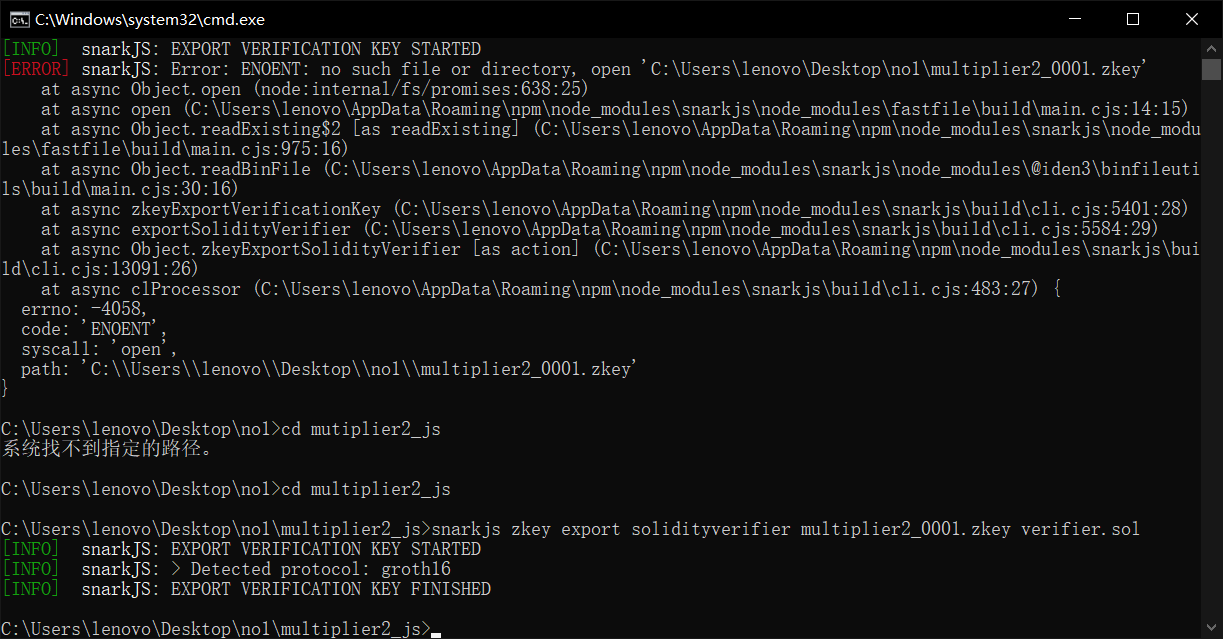
snarkjs powersoftau prepare phase2 pot12\_0001.ptau pot12\_final.ptau -v

snarkjs groth16 setup multiplier2.r1cs pot12\_final.ptau multiplier2\_0000.zkey

snarkjs zkey contribute multiplier2\_0000.zkey multiplier2\_0001.zkey --name="1st Contributor Name" -v

snarkjs zkey export verificationkey multiplier2\_0001.zkey verification\_key.json

snarkjs groth16 prove multiplier2\_0001.zkey witness.wtns proof.json public.json



我想适配sm3电路。我现在正在学习circomlib中与sha256算法有关的代码，目的是模仿这个写出sm3的电路，你认为我的思路对吗，有无可行性。

你的思路是对的，并且是可行的。Circomlib 中的 SHA-256 电路实现了哈希函数的基本逻辑，包括轮函数、位运算、消息扩展等。SM3 和 SHA-256 结构相似，主要区别在于：

1. **不同的消息扩展**：SM3 的消息扩展逻辑不同，需要额外实现 P1 置换等特殊操作。
2. **不同的轮函数**：SM3 使用不同的 T\_j 常数以及 FF\_j、GG\_j 逻辑函数。
3. **不同的初始向量 (IV)**：SM3 的初始哈希值与 SHA-256 不同。
4. **不同的非线性变换**：SM3 采用 P0 和 P1 作为消息扩展和压缩函数中的变换，而 SHA-256 主要依赖于 σ0 和 σ1。

建议的实现步骤：

* **拆解 SHA-256 代码**，理解 Circom 中如何实现哈希函数。
* **找出 SHA-256 和 SM3 的相似点**，复用可以复用的模块（如二进制加法、旋转移位等）。
* **编写 SM3 专有的部分**，如 P0、P1、T\_j、FF\_j、GG\_j，并适配 SHA-256 代码结构。
* **构造 SM3 消息扩展和压缩函数**，确保逻辑符合 SM3 标准。
* **编写测试用例**，对比标准 SM3 计算结果，验证电路的正确性。

Circom 主要的挑战是 **有符号数处理、二进制表示和约束优化**，如果你需要高效的实现，可能需要额外优化。你可以先完成一个基础版，确保功能正确，再逐步优化计算成本。

介绍circom

[Circom 2.0: A Scalable Circuit Compiler-CSDN博客](https://blog.csdn.net/mutourend/article/details/125953309)

[Compiler and Composability in ZKP](https://www.youtube.com/watch?v=zRngElDdUNE)youtube的讲座

学习IDE

[教程: 使用新版 Remix-IDE 开发智能合约DApp本文将使用图文的形式介绍新版本的以太坊智能合约开发工具 Rem - 掘金](https://juejin.cn/post/7046041881913851912)

[导航混音 — Remix - Ethereum IDE 1 文档](https://remix-ide.readthedocs.io/zh-cn/latest/layout.html#default-tools)

[GitHub - iden3/snarkjs: zkSNARK implementation in JavaScript & WASM](https://github.com/iden3/snarkjs)

[GitHub - iden3/circomlib: Library of basic circuits for circom](https://github.com/iden3/circomlib/tree/master)

本地编译以及sm3代码库结构

sm3.circom

├── sm3compression.circom

│ ├── sm3compression\_function.circom

│ ├── tt1.circom

│ │ └── ff.circom

│ ├── tt2.circom

│ │ └── gg.circom

│ ├── ss1.circom

│ ├── ss2.circom

│ ├── p.circom

│ ├── extension.circom

│ ├── rotate.circom

│ ├── xor3.circom

│ └── ../binsum.circom

└── constants.circom

**一些有参考意义的报错问题**

stderr: error[T2000]: Typing error found ┌─ "main.circom":681:26 │ 681 │ for (i=0; i<16; i++) tt1[i] = TT1\_0(); │ ^^^^^^^^^^^^^^^^ Assignee and assigned types do not match. All componentes of an array must be instances of the same template

在Circom中，试图向数组分配不同类型的组件实例，导致类型不匹配。

**确保组件类型一致:**

检查你是如何声明 tt1 的，它必须被初始化为存储相同类型的组件：

component tt1[16];

for (i = 0; i < 16; i++) {

tt1[i] = TT1\_0();

}

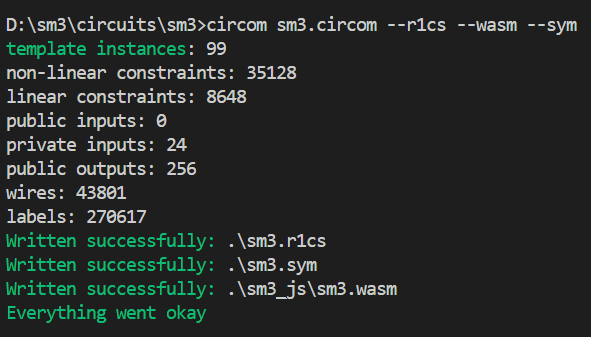
**保持类型一致：**

确保**所有赋值给 tt1 的元素都是相同的组件类型**：

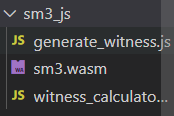
* 如果 TT1\_0() 返回的是一个组件实例，请确认它的类型和声明 tt1 时使用的类型一致。
* 如果你试图混用不同类型的组件（比如 TT1\_0() 和其他组件），Circom 会报错。

1.编译电路

circom sm3.circom --r1cs --wasm --sym

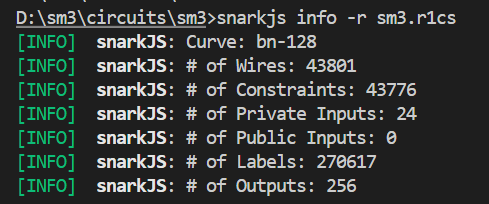


生成了

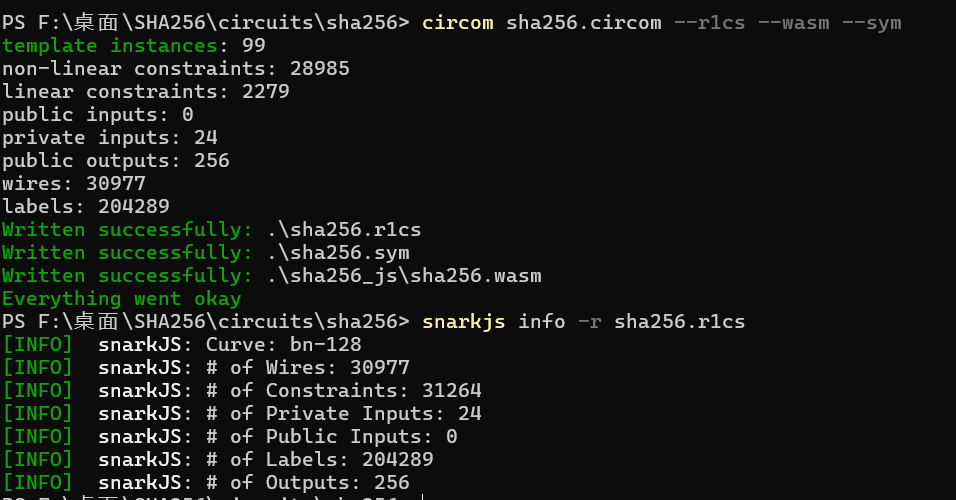


2.显示电路信息

snarkjs info -r circuit.r1cs



\*\*\*这是跟sha256的对比。。



考虑做数据量大的时候，体现出明显的效果，第五章实验评估出效果

最后的证明过程，并行

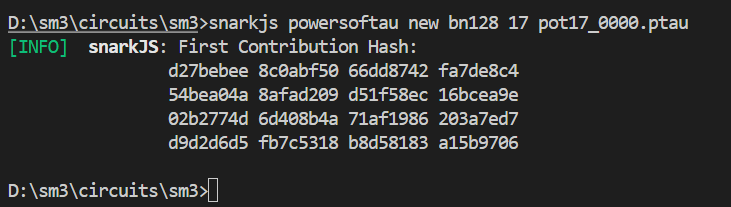
3.生成.ptau文件，可信设置**Start a new powers of tau ceremony**

一个报错：电路规模超出了当前使用的 Powers of Tau 文件（pot12\_0001.ptau）支持的最大约束数

[ERROR] snarkJS: circuit too big for this power of tau ceremony. 90584 > 2\*\*12

3.1

snarkjs powersoftau new bn128 17 pot17\_0000.ptau

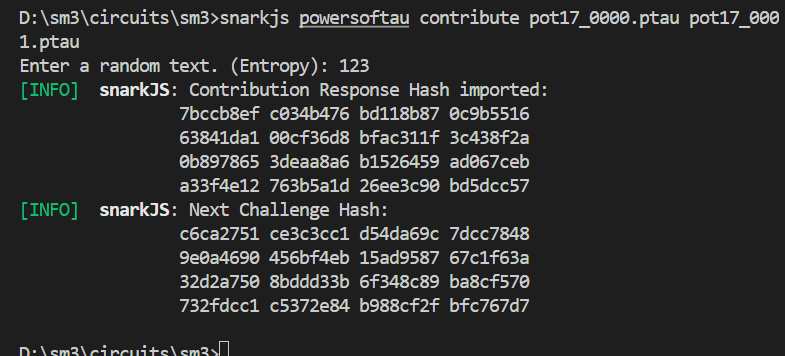


得到

3.2 贡献随机性

贡献随机性

snarkjs powersoftau contribute pot17\_0000.ptau pot17\_0001.ptau

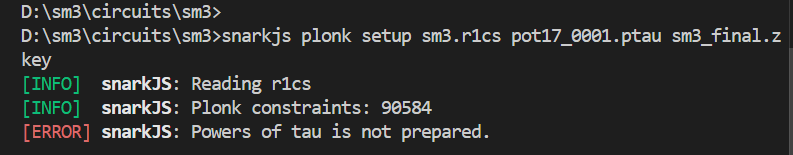


得到

4.为电路进行可信设置

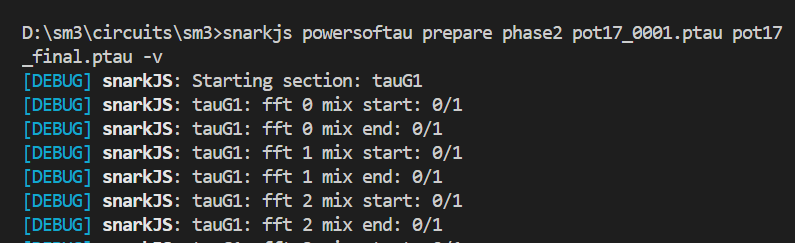
snarkjs plonk setup sm3.r1cs pot17\_0001.ptau sm3\_final.zkey

报错

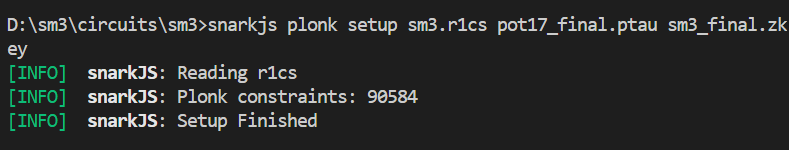




解决：snarkjs powersoftau prepare phase2 pot17\_0001.ptau pot17\_final.ptau -v

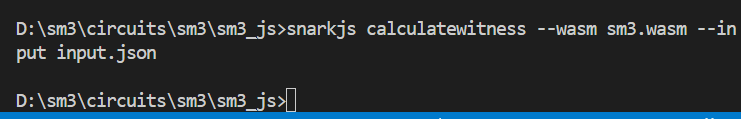


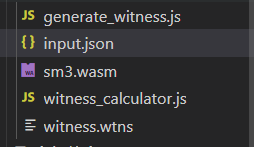
snarkjs plonk setup sm3.r1cs pot17\_final.ptau sm3\_final.zkey



**5. 进入sm3\_js,计算见证**。我们使用在<circuit\_name>\_js目录下创建的Javascript/WASM程序来为我们的输入创建见证（所有导线的值）：

snarkjs calculatewitness --wasm sm3.wasm --input input.json





calculatewitness命令将来自 input.json 的输入发送到 circuit.wasm，后者执行电路，计算（并跟踪）所有中间信号和最终输出。

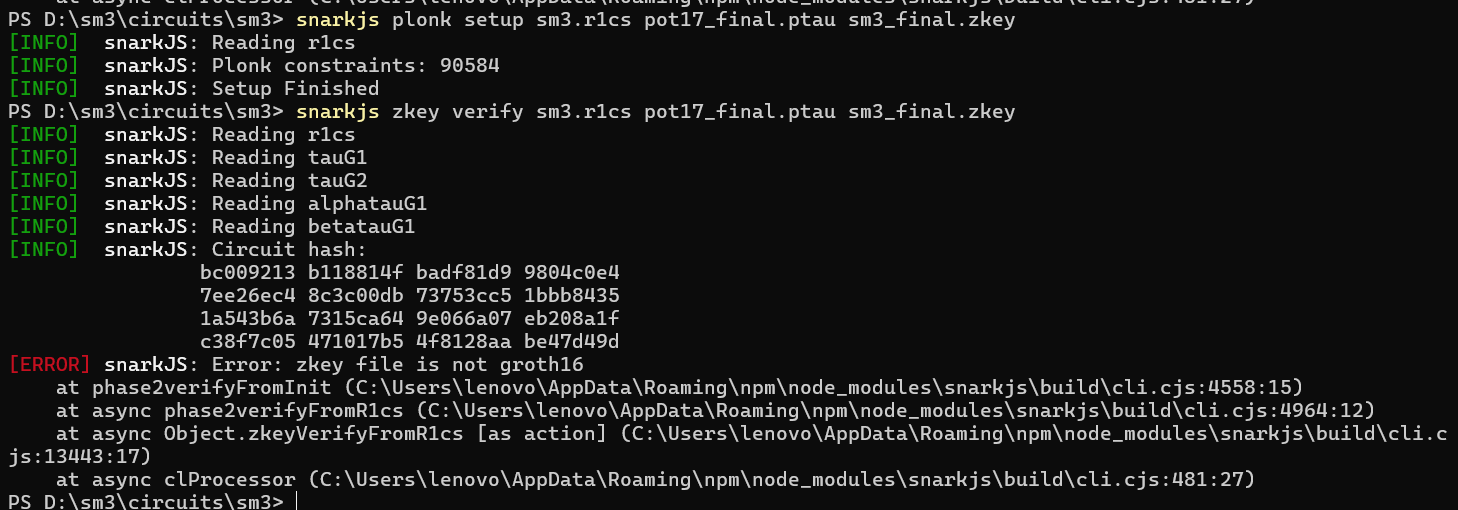
**5.1 想要看witness，可以让snarkjs转换**

snarkjs wtns export json witness.wtns witness.json

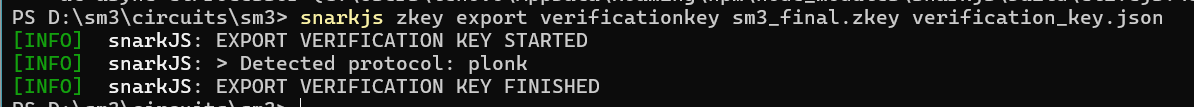
能生成43803个输入中间输出的值，电路中所有信号的值

**6.按照21验证最终zkey**

snarkjs zkey verify sm3.r1cs pot17\_final.ptau sm3\_final.zkey



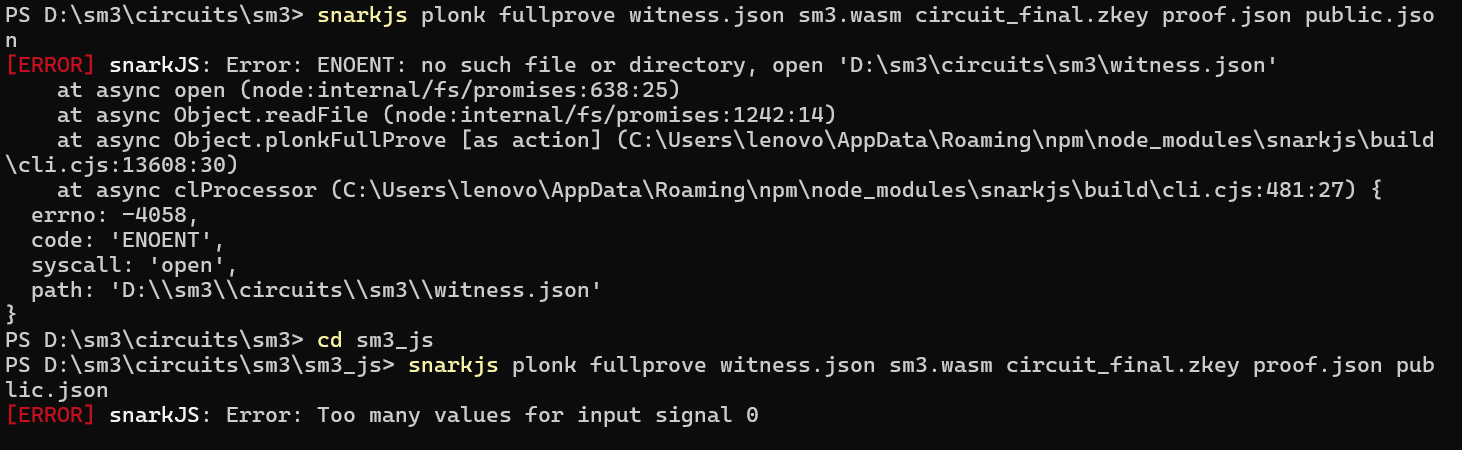
snarkjs zkey export verificationkey sm3\_final.zkey verification\_key.json



7.还可以一步计算witness和证明

snarkjs plonk fullprove witness.json sm3.wasm circuit\_final.zkey proof.json public.json

报错说

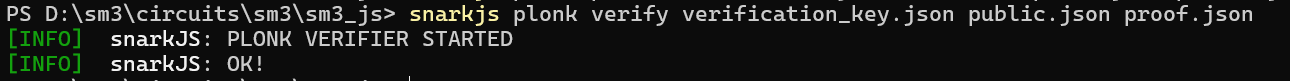


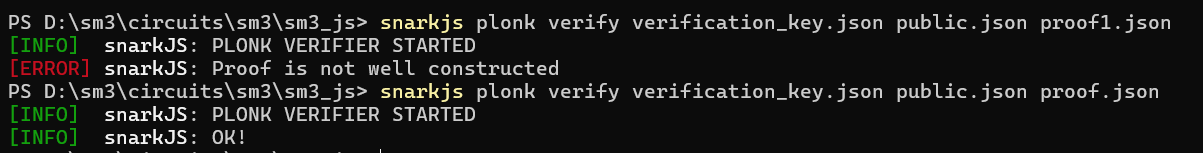
snarkjs plonk prove sm3\_final.zkey witness.wtns proof.json public.json

**生成了证明和public.json**

7.验证密钥

snarkjs plonk verify verification\_key.json public.json proof.json





**关于大小端序在circomlib的SHA256和SM3编写过程的问题**

SHA：

* Function中，定义function sha256compression（hin，inp），与SM3一致

对inp数据块扩展时：

                w[i] +=  inp[i\*32+31-j]<<j;

说明对inp的存储是大端序，inp[31]（高地址）是数字最低位（左移0位）

对w[i]也就是每一轮w这个32位字的存储是十进制形式的。

* Function 中，字寄存器H[8]转换成compression 的 out[256]，与SM3一致

out[i\*32+31-j] = (H[i] >> j) & 1;

out[31]是H[i]的32位数字的最低位，大端序

* H和K字寄存器的初始常量的模板中，SHA用out[0]放低数位，是小端序，与SM3不同

        SHA256中：out[i] <== (c[x] >> i) & 1;

        SM3中：out[31-i] <== (c[x] >> i) & 1;

* 循环移位：

        out[i] <== in[ (i+r)%n ];

对于小端序，这样是循环右移，即每位变成了原来数值和地址都高r位的位置上的数

对于大端序，这样是循环左移，即每位变成了原来数值低r位地址高r位的位置上的数

* Binsum求和

输入的n位字是in[ops][n],它的大小端序的不同导致了每位数值权重的不同

            lin += in[j][k] \* e2;

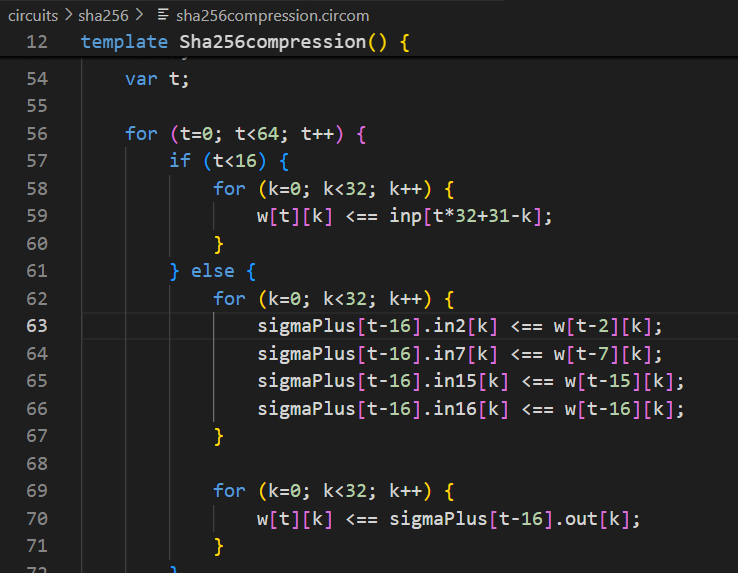
e2作为权重从1开始\*2，这导致小端序k从0开始，是SHA256的用法，但大端序k从n-1开始。

结果转为二进制数组也是同理

        out[k] <-- (lin >> k) & 1;

lin是十进制数字，小端序要把小数位放out[0]低地址，可以一直放完，直到超出界限，取的时候只取0-n，也不会影响lin和lout验证的约束

* 实现inp大端序到w[k]小端序的转化



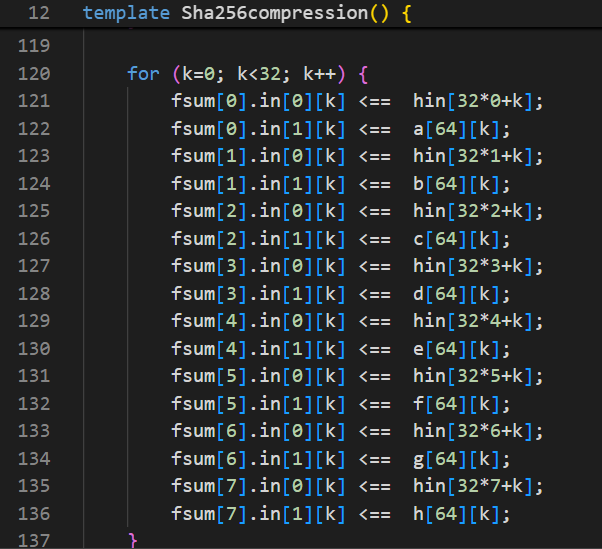
* 总结一下SHA256的端序变化，首先数据输入inp是大端序，也就是符合常识的输入。

底层涉及到的计算工具是小端序，主要是因为binsum，因此导致了Rot和shift函数的诡异

因此涉及到底层逻辑运算和求和函数，他们的输入都是小端序

寄存器abcdefgh的二进制输入输出也是小端序，因为初始输入来自constrant，结果的更新也来自底层计算的out。

中层函数t或者sigma看起来都没问题是因为底层已经统一为小端序，又主要是逻辑运算，单位是每个bit位，对顺序要求不高。例如



最后最高层的检验，是通过

 out[31-k]     === fsum[0].out[k];

这里的交叉实现的，也就是这里把求和函数的out小端序转到了压缩函数out的大端序上。

如下图，下一轮压缩函数的in又由out的大端序转到了小端序

图形用户界面, 文本

AI 生成的内容可能不正确。

 **电路设计问题**

* 国密算法的计算特点是否会导致电路规模过大或计算复杂度过高？如何有效分解和优化算法以适配电路需求？
* 在设计电路时，如何权衡电路复杂度与零知识证明协议的效率和安全性？

 **性能问题**

* 在零知识证明协议中，国密算法的计算效率是否能够满足实际应用需求？
* 如何通过电路设计和参数优化，降低证明生成和验证的时间成本？

 **安全性问题**

* 设计的零知识证明电路是否能够抵御潜在的攻击（如音知识攻击或伪造攻击）？
* 如何保证电路生成的证明具有足够的随机性和隐私性？

 **应用问题**

* 针对特定应用场景（如区块链隐私保护），如何设计一个易于集成且高效的国密算法零知识证明方案？
* 在实际应用中，如何应对多用户或大规模应用场景下的性能瓶颈？

1.与sha256有个对比，虽然extension会复杂一点，增加的开销应该是符合预期的。

2.SM3的优化方法，比如extension向量化，sm3的线性函数有两种等效的表示，在CPU里有一种表示会更高效

3.里面二进制的逻辑运算，查表优化逻辑运算，Plonky。Circom底层库可能不支持。

4.场景merkle树，输入填充，初始值确定下来，简化流程。

5.不同的曲线，对阶有要求，SM3约束写好后可以确定阶需要多大，用SM9没法用快速FFt不满足全国密化

6.综述，常规plonkbulletproofgroth，选一个介绍snark，还有优化方案

7.评估看电路规模，

2025.3.19 细读plonk，sonic等论文，准备中期答辩和论文绪论

[Sonic 简介：具有几乎无需信任的设置的实用 zk-SNARK – Bentham's Gaze](https://www.benthamsgaze.org/2019/02/07/introducing-sonic-a-practical-zk-snark-with-a-nearly-trustless-setup/)

Sonic 不需要每个电路的可信设置，而只需要所有电路的单一设置。

Sonic 的设置永远不必结束，因此可以通过积累更多贡献来持续保护它

与 Groth 等人的设计相比，Sonic 的优势在于 Sonic 使用较小的全局参数集。因此，可以在个人笔记本电脑上实现存储、更新和验证 Sonic 的参数。另一方面，Groth 等人的公共参数的存储、更新和验证成本很高，以至于需要一组分布式计算机来执行这些任务。此外，要将 Groth 等人的 zk-SNARK 与特定应用程序一起使用，某些方必须在全局参数上运行昂贵（但不可信）的派生过程，以获得特定于应用程序的参数。Sonic 不需要任何派生过程。

与 Bulletproofs（由 Bootle 等人引入并由 Bünz 等人改进）相比，Sonic 具有更小的证明大小和更小的验证器计算。Bulletproof 验证程序需要计算，其大小取决于底层应用程序的大小。另一方面，Sonic 有一个固定大小的验证器，适用于任何应用程序。Bulletproofs 确实具有完全无需信任的设置，而 Sonic 仅具有几乎无需信任的设置。此外，Bulletproofs 的安全性基于比 Sonic 更多的标准假设。总体而言，Bulletproofs 更适合较小的应用程序，而 Sonic 更适合较大的应用程序。

它是建立在基于配对的组之上的，并大量使用了Kate等人的恒定大小的多项式承诺方案。它假设证明者和验证者之间进行了两轮交互，然后在随机预言机模型中成为非交互式的。

[STARKs, Part I: Proofs with Polynomials](https://vitalik.eth.limo/general/2017/11/09/starks_part_1.html)暂时没看完

ZK-SNARK 可用于从可验证计算到保护隐私的加密货币的各种用例。它有一个更新、更闪亮的表亲：ZK-STARK。ZK-STARK 的 T 代表“透明”，解决了 ZK-SNARK 的主要弱点之一，即它对“可信设置”的依赖。

它们还带有更简单的加密假设，避免了对椭圆曲线、配对和指数知识假设的需求，而是纯粹依赖哈希和信息论;这也意味着即使面对使用量子计算机的攻击者，它们也是安全的。

然而，这是有代价的：证明的大小从 288 字节增加到几百 KB

[STARKs, Part II: Thank Goodness It's FRI-day](https://vitalik.eth.limo/general/2017/11/22/starks_part_2.html)看不懂思密达。

[了解 PLONK](https://vitalik.eth.limo/general/2019/09/22/plonk.html)看完了，很经典，对累加器解释的简明扼要

[Vitalik Buterin 的网站](https://vitalik.eth.limo/index.html)V神的所有博客

[vnTinyRAM 的继续 zkSNARK 教程 |作者 Mike Hearn |Mike 的博客](https://blog.plan99.net/vntinyram-7b9d5b299097)snark的延伸教程

[零知识证明：图解入门 – 关于密码工程的几点思考](https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/)入门教程

[KZG多项式承诺 | Dankrad Feist](https://dankradfeist.de/ethereum/2021/10/13/kate-polynomial-commitments-mandarin.html)已读，对kate证明的介绍很深入

[PLONK 的工作原理：第 2 部分\_plonk证明-CSDN博客](https://blog.csdn.net/freedomhero/article/details/127813993)

Marlin

[Marlin:Preprocessing zkSNARKs with Universal and Updatable SRS学习笔记\_zk snark marlin-CSDN博客](https://blog.csdn.net/mutourend/article/details/121902741)

手搓plonk

[PLONK：共世非交互式知识的拉格朗日基排列 - SotaZK Labs docs](https://docs.sotazk.org/docs/plonk/)基于v神的二手博客

[Zero-Knowledge Proof for Privacy over Ethereum: Permutations over Lagrange-Bases for Oecumenical Non-Interactive Arguments of Knowledge (PLONK) [step-by-step] (Part 1) | by Agnish Ghosh. | Coinmonks | Medium](https://medium.com/coinmonks/zero-knowledge-proof-for-privacy-over-ethereum-permutations-over-lagrange-bases-for-oecumenical-dc7838e0bc8b)

[PLONK 的工作原理：第 1 部分。PLONK 是最先进的 zk-SNARK... |作者 sCrypt |中等](https://scryptplatform.medium.com/how-plonk-works-part-1-bc8050f4805e)

[PLONK by Hand（第 1 部分：设置）](https://research.metastate.dev/plonk-by-hand-part-1/)

[PLONK by Hand (Part 2: The Proof)](https://research.metastate.dev/plonk-by-hand-part-2-the-proof/)

[Plonk By Hand - HackMD](https://hackmd.io/@chaosma/ByBIElFno)

[(PLONK) [step-by-step] (Part 1) - Google 搜索](https://www.google.com/search?q=(PLONK)+%5Bstep-by-step%5D+(Part+1)&oq=(PLONK)+%5Bstep-by-step%5D+(Part+1)&gs_lcrp=EgRlZGdlKgYIABBFGDkyBggAEEUYOdIBBzc3NGowajGoAgCwAgA&sourceid=chrome&ie=UTF-8)

[零知识证明 - 深入理解PlonK算法 | 登链社区 | 区块链技术社区](https://learnblockchain.cn/article/2180)

In fact, the *alt\_bn128*curve that is *implemented in Ethereum Mainnet* uses this same curve equation and generator, but over a large prime field.

1.与sha256有个对比，虽然extension会复杂一点，增加的开销应该是符合预期的。

2.SM3的优化方法，比如extension向量化，sm3的线性函数有两种等效的表示，在CPU里有一种表示会更高效

3.里面二进制的逻辑运算，查表优化逻辑运算，Plonky。Circom底层库可能不支持。

4.场景merkle树，输入填充初始值确定下来，简化流程。电路应用于merkle树这种特定结构里面，可能会有进一步的优化

5.不同的曲线，对阶有要求，SM3约束写好后可以确定阶需要多大。

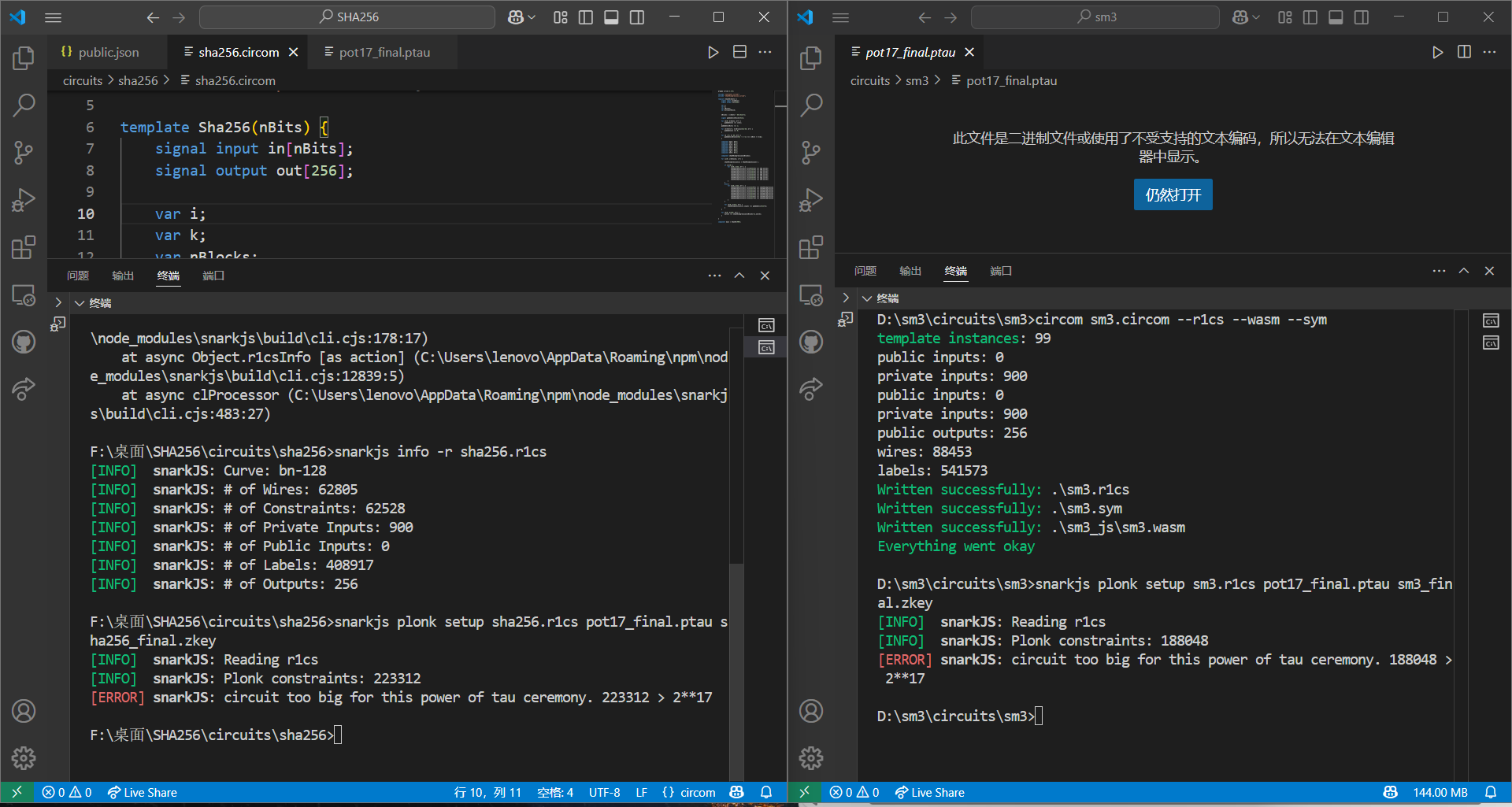
后期拟完成的研究工作：

（1）探索SM3电路约束数量的优化方法，实现电路性能优化。考虑到逻辑运算会增加电路的约束数量，从而影响证明生成的计算成本，设计查表优化逻辑运算，在circom电路中利用自定义约束实现查表。

（2）实现SM3算法的优化。

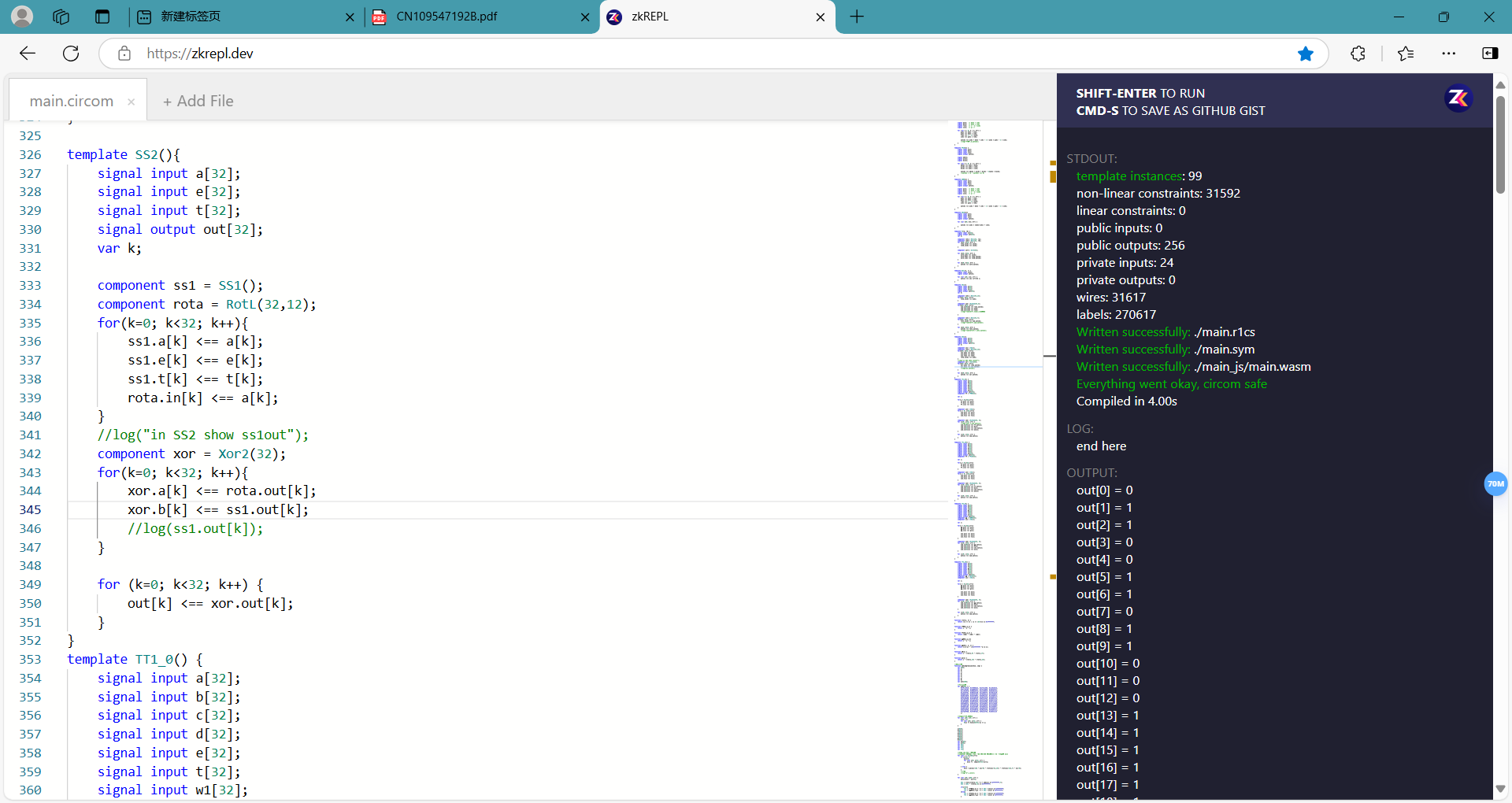
* 探索消息扩展的向量化，在SM3的计算过程中，消息扩展阶段占据了较大的计算量。期望利用向量化技术提高扩展过程的并行计算能力，减少循环依赖，提高整体吞吐率。
* 探索线性函数的高效实现，通过选择更适合硬件架构的表示方式，可以减少计算延迟，提高计算性能。

（3）结合Merkle树的应用场景，在后续构建零知识证明的Merkle路径时，预先确定输入填充的初始值，以此减少动态计算量并简化电路设计。此外，利用优化的哈希路径计算方式，可以进一步降低证明的生成时间。



修改SS2的依赖关系之后的编译结果





差别只是少了label，坏坏坏

Github与circom有关和我做的相似的库

[nullity00/web3-resources: Collection of web3 resources for blockchain enthusiasts. Contains some major alpha on ZKP, Solidity & auditing.](https://github.com/nullity00/web3-resources)

[vocdoni/keccak256-circom: keccak256 hash implementation in Circom](https://github.com/vocdoni/keccak256-circom)

在使用 **Plonk** 零知识证明系统时，选择适当的椭圆曲线至关重要。以下是几个关键因素以及推荐的曲线：

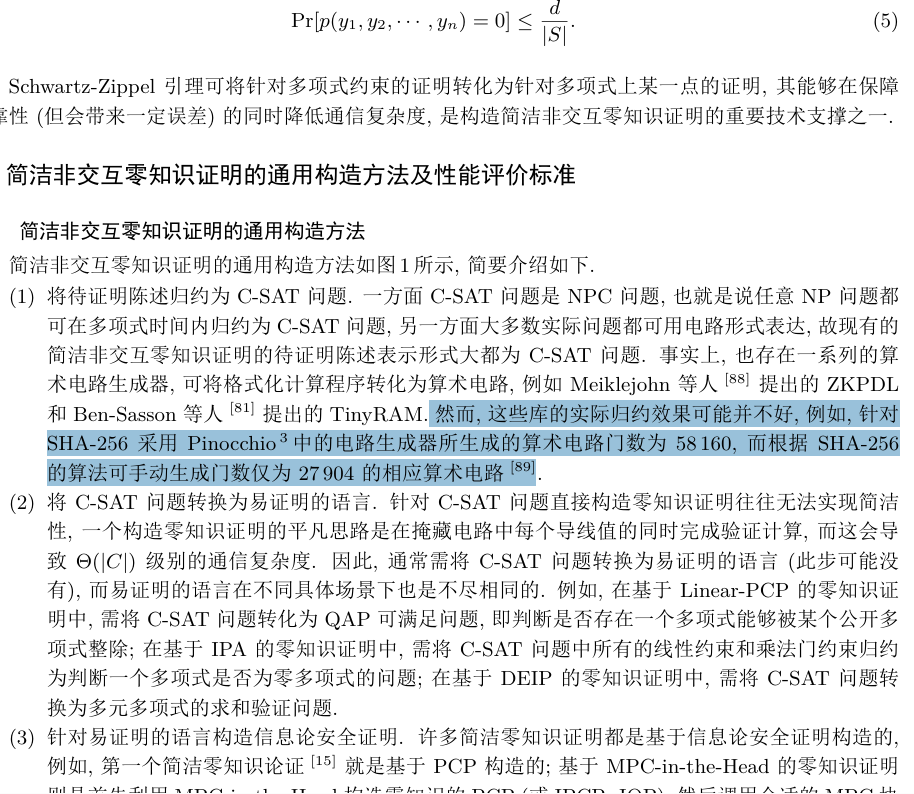
**关键考量因素**

1. **曲线的阶**
   * 需要一个**足够大的质数阶群**来支持安全的多项式承诺方案（如 KZG 承诺）。
   * 由于电路的约束量在 **30,000 到 40,000** 之间，需要支持至少该规模的多项式评估，因此曲线的阶必须大于电路约束数的上界。
2. **Plonk 通常使用的曲线**
   * Plonk 依赖 **KZG（Kate, Zaverucha, Goldberg）承诺方案**，这需要使用 **支持高效配对（pairing）的椭圆曲线**，即：
     + **BN 曲线（Barreto-Naehrig）**
     + **BLS12 曲线（Barreto-Lynn-Scott）**
3. **推荐曲线**
   * **BLS12-381**（最常用）
     + 安全性高，支持 128-bit 安全级别
     + 适用于大多数 ZK 证明系统，包括 Plonk
   * **BN254（BN128）**
     + 被早期的 zk-SNARKs 采用，如 Groth16，但安全性较 BLS12-381 低（约 100-bit 级别）
     + 对于 30,000-40,000 约束的电路，仍然可以使用，但安全性不如 BLS12-381
4. **是否需要阶与约束量匹配**
   * **曲线的阶应至少大于电路的最大约束数**，但通常会选择更大的阶，以确保安全性和未来扩展性。
   * BLS12-381 的阶约为 **r ≈ 2²⁵⁵**，远大于 40,000，完全足够。

**结论**

* **推荐使用 BLS12-381**，因为它提供更高的安全性，且是现代 Plonk 实现的首选曲线。
* **BN254 也可行**，但安全性相对较低，适用于较低安全需求的环境。

你打算在哪个应用场景下使用 Plonk？如果对性能或特定平台有要求，可以进一步优化选择。

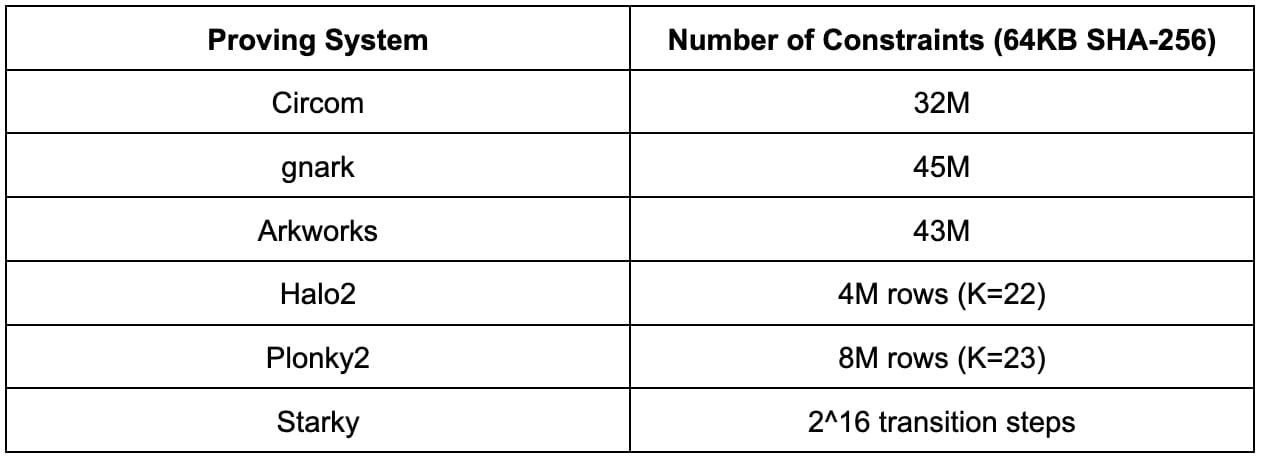


**写论文了**

Word的修改指南

[再发一下论文格式修改-供本科生参考 - 西电睿思灌水专区 - 西电睿思BBS - Powered by Discuz!](https://rs.xidian.edu.cn/forum.php?mod=viewthread&tid=1099090)

[零知识证明开发框架万神殿 | Celer Network发布于币安广场](https://www.binance.com/zh-CN/square/post/263965)



[Merkle Proofs with ZoKrates - ZKPlus](https://zk-plus.github.io/tutorials/basics/merkle-proof#:~:text=Merkle%20proofs%20can%20be%20enriched%20through%20privacy%20guarantees,disclosing%20the%20candidate%20element%20or%20the%20Merkle%20Path.)

[Plonkup 中的哈希函数 |研究 - Anoma](https://anoma.net/research/hash-functions-in-plonkup)

[UltraPlonk: Part 1](https://aztec.slides.com/suyashbagad_aztec/ultraplonk-part-1#/2/0/0)

[zhuanlan.zhihu.com](https://zhuanlan.zhihu.com/p/443323328)

[plonk - Google 学术搜索](https://scholar.google.com/scholar?start=20&q=plonk&hl=zh-CN&as_sdt=0,5)

[Gabizon: Plonk: Permutations over lagrange-bases... - Google 学术搜索](https://scholar.google.com/scholar?start=20&hl=zh-CN&as_sdt=2005&sciodt=0,5&cites=12591581529061923405&scipsc=)

[hackmd.io](https://hackmd.io/@lm3fyIz-TQqOiqATSbtJ7w/HkL8KdgOF)

[Anemoi：一系列支持 ZK 的 AO 哈希函数。](https://anemoi-hash.github.io/#description)

[www.binance.com](https://www.binance.com/ru-UA/square/post/12049983766202)

[SNARKs：从理论到实践：零知识证明系统的发展与应用-CSDN博客](https://blog.csdn.net/mutourend/article/details/136146403)

[a16z：从零知识证明技术发展理解去中心化的进步 - Foresight News](https://foresightnews.pro/article/detail/2350)

[7-Lookup Gate - ZKPunk's ZKPedia](https://learn.zkpunk.pro/plonk-intro-cn/plonk-lookup.html)

[Plonkish - Google 搜索](https://www.google.com/search?q=Plonkish&sca_esv=28d2bf2905880196&sxsrf=AHTn8zpQX_qV1ZDPEucIMUzUrDBh1dOPVg%3A1743740216639&ei=OF3vZ6vcJtmLwbkP9PCD4QE&ved=0ahUKEwir2MXBwr2MAxXZRTABHXT4IBwQ4dUDCBA&uact=5&oq=Plonkish&gs_lp=Egxnd3Mtd2l6LXNlcnAiCFBsb25raXNoMgoQABiwAxjWBBhHMgoQABiwAxjWBBhHMgoQABiwAxjWBBhHMgoQABiwAxjWBBhHMgoQABiwAxjWBBhHSI4PULgOWLgOcAJ4AZABAJgBAKABAKoBALgBA8gBAPgBAZgCAqACCJgDAIgGAZAGBZIHATKgBwCyBwC4BwA&sclient=gws-wiz-serp)

[PLONKish Arithmetization - The halo2 Book](https://zcash.github.io/halo2/concepts/arithmetization.html)

[Arithmetization: R1CS, Plonkish, AIR, and CCS Constraint Systems](https://www.ingonyama.com/ingopedia/arithmzk)

[FPGA时序分析第一节课笔记（FPGA基本构成以及工作原理）\_二路选择器的fpga结构-CSDN博客](https://blog.csdn.net/weixin_58120495/article/details/133708975)

[查找证明 - The halo2 Book](https://trapdoor-tech.github.io/halo2-book-chinese/design/proving-system/lookup.html)

[Polygon zkEVM中的子约束系统\_evm plonk-CSDN博客](https://blog.csdn.net/mutourend/article/details/128205837)

使用查找表的circom电路约束

[Polocolo: A ZK-Friendly Hash Function Based on S-boxes Using Power Residues](https://iacr.org/cryptodb/data/paper.php?pubkey=35025)

传统的哈希函数在零知识证明设置中通常效率低下，导致设计了几个对 ZK 友好的哈希函数。另一方面，查找参数最近已被纳入零知识协议中，允许更有效地处理 ''ZK-unfriendly''作，因此基于查找表的 ZK 友好哈希函数。

[SNARKs：从理论到实践：零知识证明系统的发展与应用-CSDN博客](https://blog.csdn.net/mutourend/article/details/136146403)

Sonic、Plonk和Marlin通过引入通用且可更新的结构化reference字符串，解决了 Groth16 中每个程序的可信设置问题。Marlin 提供了基于 R1CS 的证明系统，是 Aleo 的核心。

Plonk引入了一种新的算术化方案（后来称为 Plonkish），并对copy constraints使用grand-product check。Plonkish 还允许为某些操作引入专门的门，即所谓的定制门。多个项目都有 Plonk 的定制版本，包括 Aztec、ZK-Sync、Polygon ZKEVM、Mina’s Kimchi、Plonky2、Halo 2 和 Scroll 等。可参看博客All you wanted to know about Plonk。

3.5 Lookups (2018/2020)

Gabizon 和 Williamson 在 2020 年引入了plookup，使用grand product check来证明某个值包含在预先计算的值表中。尽管之前在Arya中提出了lookup arguments，但其构造需要确定lookups的multiplicities，效率较低。PlonkUp论文展示了如何将 plookup argument 引入 Plonk。这些lookup arguments的问题在于，它们迫使证明者为整个表支付费用，而与其查找次数无关。这意味着大型表的成本相当大，并且已投入了大量的精力来将证明者的成本降低到其使用的查找数量。

Haböck 引入了LogUp，它使用对数导数将乘积检查转换为倒数之和。LogUp 对于Polygon plonky2 ZKEVM（Beyond Limits: Pushing the Boundaries of ZK-EVM） 的性能至关重要，其需要将整个表拆分为多个 STARK 模块。这些模块必须正确链接，并跨表查找来强化此操作。LogUp-GKR的推出利用GKR协议来提高LogUp的性能。Caulk是第一个prover time与table size呈亚线性关系的lookup argument，其预处理时长为O ( N log ⁡ N ) \mathcal{O}(N\log N)O(NlogN)且storage为O ( N ) \mathcal{O}(N)O(N)，其中N NN为table size。随后出现了其他几个方案，如Baloo、flookup、cq和caulk+。Lasso提出了多项改进，避免在表具有给定结构时对其进行commit。此外，Lasso 的证明者只为查找操作访问的表条目付费。Jolt利用 Lasso 通过查找来证明虚拟机的执行情况。

3.6 Spartan (2019)

Spartan为使用 R1CS 描述的电路提供了 IOP，利用多变量多项式和sumcheck协议的属性。使用合适的多项式承诺方案，它会产生带有线性时长prover的透明 SNARK。

3.7 HyperPlonk (2022)

HyperPlonk基于使用多变量多项式的 Plonk 思想而构建。它不依赖于商来检查约束的执行情况，而是依赖于sumcheck协议。它还支持high degree约束，而不会损害证明者的运行时间。由于它依赖于多变量多项式，因此无需执行 FFT，且证明者的运行时间与电路尺寸呈线性关系。HyperPlonk 引入了适合较小域的新permutation IOP 和基于sumcheck的batch opening协议，这减少了prover的工作量、证明大小和验证者的时间。

[您想知道的有关 Plonk 的所有信息](https://blog.lambdaclass.com/all-you-wanted-to-know-about-plonk/)

[我们对零知识证明历史的高度主观看法](https://blog.lambdaclass.com/our-highly-subjective-view-on-the-history-of-zero-knowledge-proofs/)

写综述参考

[ZK Protocols using SNARKs](https://www.ingonyama.com/ingopedia/protocolssnark)

[零知识证明：zk-SNARKs的历史、进展与应用-CSDN博客](https://blog.csdn.net/qq_41781104/article/details/136517483)

写circom参考

[CIRCOM - Polygon Knowledge Layer](https://docs.polygon.technology/zkEVM/concepts/circom-intro-brief/#circom-as-a-dsl)

[零知识证明开发框架万神殿 | Celer Network发布于币安广场](https://www.binance.com/zh-CN/square/post/263965)

[零知识证明编码入门 - 知乎](https://zhuanlan.zhihu.com/p/617975690)

[zkProver 中的 CIRCOM - 多边形知识层](https://docs.polygon.technology/zkEVM/architecture/zkprover/stark-recursion/circom-in-zkprover/)（更偏stark的递归）

 **分布电路 chip（spread chip）**：用于将一个数的偶数/奇数位扩展开来，以便和一些特殊的查表、约束等操作结合。

 **电路上下文（Context）**：在零知识电路中注册变量、创建约束。

 **Gate 和 QuantumCell**：用于构建加法、乘法、乘加等门操作。

 **AssignedValue / Witness**：是电路中实际分配的值（电路 witness）

Spread.rs

/// 将一个 dense 编码的值转换为 spread 编码的值

pub fn spread(

) -> Result<AssignedValue<F>, Error>

Limb的数量是由16位/lookupbits决定

|  |  |
| --- | --- |
| 将输入拆为多个 limb | 为了便于查表、节省约束 |

|  |  |
| --- | --- |
| 为每个 limb 分配 witness | 进入电路处理 |

|  |  |
| --- | --- |
| 重组 limb 验证是否和原始一致 | 确保电路逻辑正确 |

|  |  |
| --- | --- |
| 为每个 limb 做 spread 编码 | 插入空位，准备查表或组合操作 |

|  |  |
| --- | --- |
| 将多个 spread\_limb 合并 | 得到最终的 spread 表达 |

/// 不进行检查地将 spread 编码的值拆成 even（偶数位）和 odd（奇数位）两个值

pub fn decompose\_even\_and\_odd\_unchecked(

) -> Result<(AssignedValue<F>, AssignedValue<F>), Error> {

Ok((even\_assigned, odd\_assigned))

    /// 将 AssignedValue 形式的比特数组转换成字段元素（位权求和）

    fn bits\_to\_fe(

) -> Result<AssignedValue<F>, Error> {

Ok(sum)

    /// 对单个 limb 做 spread 编码（在每个比特之间插入 0），最后输出的还是字段表示

fn spread\_limb(

) -> Result<AssignedValue<F>, Error> {

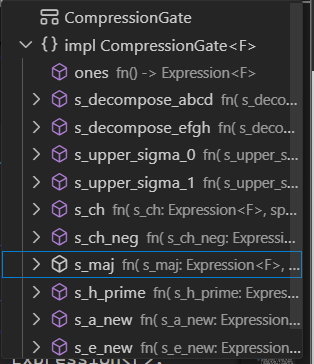
        Ok(assigned\_spread)

**Compression**

Compression\_gates.rs:

pub struct CompressionGate<F: FieldExt>(PhantomData<F>);

impl<F: FieldExt> CompressionGate<F> {



Compression\_util.rs:

Halo2的相关链接

[plookup多项式协议详解-CSDN博客](https://blog.csdn.net/mutourend/article/details/114603638)PLOOKUP代码解析

时间：

Sm3 512：

PS D:\sm3\circuits\sm3> Measure-Command { circom sm3.circom --r1cs --wasm --sym }

Days : 0

Hours : 0

Minutes : 0

Seconds : 4

Milliseconds : 497

Ticks : 44971056

TotalDays : 5.20498333333333E-05

TotalHours : 0.001249196

TotalMinutes : 0.07495176

TotalSeconds : 4.4971056

TotalMilliseconds : 4497.1056

Sm3 24

PS D:\sm3\circuits\sm3> Measure-Command { circom sm3.circom --r1cs --wasm --sym }

Days : 0

Hours : 0

Minutes : 0

Seconds : 2

Milliseconds : 911

Ticks : 29116806

TotalDays : 3.37000069444444E-05

TotalHours : 0.000808800166666667

TotalMinutes : 0.04852801

TotalSeconds : 2.9116806

TotalMilliseconds : 2911.6806

Sha256 512

PS F:\桌面\SHA256\circuits\sha256> Measure-Command { circom sha256.circom --r1cs --wasm --sym }

Days : 0

Hours : 0

Minutes : 0

Seconds : 3

Milliseconds : 810

Ticks : 38105121

TotalDays : 4.41031493055556E-05

TotalHours : 0.00105847558333333

TotalMinutes : 0.063508535

TotalSeconds : 3.8105121

TotalMilliseconds : 3810.5121

24

PS F:\桌面\SHA256\circuits\sha256> Measure-Command { circom sha256.circom --r1cs --wasm --sym }

Days : 0

Hours : 0

Minutes : 0

Seconds : 2

Milliseconds : 390

Ticks : 23901330

TotalDays : 2.76635763888889E-05

TotalHours : 0.000663925833333333

TotalMinutes : 0.03983555

TotalSeconds : 2.390133

TotalMilliseconds : 2390.133