数学之美2022季

零知识证明-深入浅出zkSnark

深圳市元宇元宙智能科技有限公司

零知识证明概述

zkSNARKop

zkSNARK

零知识证明分类和历史

•1985 年，零知识证明Zero-Knowledge Proof - 由 S.Goldwasser、 S.Micali 及 C.Rackoff 首次提出。
•2010年，Groth实现了首个基于椭圆曲线双线性映射全能的，常数大小的非交互式零知识证明协议。后来这个协议经过不断优化，最终成为区块链著名的零知识证明协议SNARKs。
•2013年，Pinocchio协议实现了分钟级别证明，毫秒级别验证，证明大小不到300字节，将零知识证明从理论带到了应用。后来Zcash使用的SNARKs正是基于Pinocchio的改进版。
•2014 年，名为Zerocash的加密货币则使用了一种特殊的零知识证明工具zk-SNARKs （ Zero-Knowledge Succinct Non-interactive Arguments of Knowledge ）实现了对交易金额、交易双方的完全隐藏，更注重于隐私，以及对交易透明的可控性。
•2017 年， Zerocash 团队提出将 zk-SNARKs 与智能合约相互结合的方案，使交易能在众目睽睽下隐身，打造保护隐私的智能合约。

	zk-Snark	Zk-Stark	Bbulletproofs
介绍	简明非交互零知识证明	简洁透明的零知识证明	
关键特性	生成的证明小 验证速度快	没有可信设置过程 生成证明时间快 抗量子攻击	没有可信设置过程
项目应用	zcash coda	StarkDEX	Monero

	Trusted Set-Up	Speed (Verifier + Prover)	Proof Size	Quantun Resistant
Zk-Snark	Yes	Middle	Smallest	No
Zk-Stark	No	Fastest	Biggest	Yes
Bulletproofs	No	Slowest	Middle	No

zkSNARKs 算法

zk-SNARK 是 zero-knowledge succint non-interactive arguments of knowledge 的简称，其中：

- Succint (简洁性)：与实际计算的长度相比，生成的零知识证据消息很小
- Non-interactive (非交互性)：几乎没有任何交互。对于 zk-SNARK 算法来说，通常有一个构建阶段，构建阶段完成之后，证明者 (prover) 只需向验证者 (verifier) 发送一个消息即可。而且，SNARK 通常还有一个被称作是 "公开验证者" 的特性，意味着任何人无需任何交互即可验证零知识证据，这对区块链是至关重要的
- Arguments (争议性)：验证者只能抵抗计算能力有限的证明者的攻击。具有足够计算能力的证明者可以创建伪造的零知识证据以欺骗验证者 (注意刚提到的足够的计算能力，它足可以打破公钥加密，所以现阶段可不必担心)。这也通常被称为 "计算完好性 (computational soundness)"，而不是 "完美完好性 (perfect soundness) "
- of Knowledge：对于一个证明者来说，在不知晓特定证明 (witness) 的前提下，构建一个有效的零知识证据是不可能的

同态隐藏+多项式构成了zkSNARK的基础

zkSNARK

zkSNARKop

QAP

同态隐藏

椭圆曲线有限域加密

双线性配对

多项式

R1CS

算术电路

d-KCA

zkSnark本质上构造了一个多项式双盲验证通用机制

双盲验证要满足两个条件：

Prover blind evaluate：

Prover得到的输入，不能有verifier的任何知识。

实现方案：

DH密钥交换机制，利用了幂函数乘法同态隐藏特性

fiat-shamir变换

椭圆曲线加密r*G代替r，利用加法同态隐藏特性，这是zkSnark的方案

Verifier blind verify：

Verifier需要不知道P的知识前提下，还能验证知识的正确性。

实现方案：

零知识ZKP

Schnorr签名

ECDSA签名

zkSNARK涉及到了d-KCA、双线性配对、随机数偏置

为什么是多项式 1/2

多项式可以提供额外的好处：
Succint.

Verifier只需要验证多项式运算的结果，而无需知道庞大的多项式系数。可以大大压缩证明所需要的数据量。

加密界对此有多年的研究，如PCP、LPCP

为什么是多项式 2/2

任何计算问题，都可以转换成多项式问题。
zkSNARK通过：
Step1：把计算问题翻译成算术电路
Step2：把算术电路转换成r1cs
Step3：把r1cs转成QAP问题

其中，step1和step2非常直观，符合直觉
QAP问题是最近10多年加密学术的突破
zkSNARK正是有了这个基础，才可以应用在任意的计算问题上

零知识证明概述

zkSNARKop

zkSNARK

zkSnarkOP setup

Succinct Non-Interactive Argument of Knowledge of Polynomial

- Setup
  - sample random values s, α
  - calculate encryptions g^{α} and $\left\{g^{s^i}\right\}_{i\in[d]}, \left\{g^{\alpha s^i}\right\}_{i\in\{0,\ldots,d\}}$
  - proving key: $\left(\left\{g^{s^i}\right\}_{i\in[d]}, \left\{g^{\alpha s^i}\right\}_{i\in\{0,\ldots,d\}}\right)$
  - verification key: $\left(g^{\alpha}, g^{t(s)}\right)$

Alpha：d-KCA

zkSnarkOP prove & verify

Prove

- assign coefficients $\{c_i\}_{i\in\{0,\ldots,d\}}$ (i.e., knowledge), $p(x) = c_d x^d + \cdots + c_1 x^1 + c_0 x^0$
- calculate polynomial $h(x) = \frac{p(x)}{t(x)}$
- evaluate encrypted polynomials $g^{p(s)}$ and $g^{h(s)}$ using $\left\{g^{s^i}\right\}_{i\in[d]}$
- evaluate encrypted shifted polynomial $g^{\alpha p(s)}$ using $\left\{g^{\alpha s^i}\right\}_{i\in\{0,\ldots,d\}}$
- sample random δ
- set the randomized proof $\pi = \left(g^{\delta p(s)}, g^{\delta h(s)}, g^{\delta \alpha p(s)}\right)$

Delta：偏置，防止泄露多项式知识

- Verification
  - parse proof π as $\left(g^{p}, g^{h}, g^{p'}\right)$
  - check polynomial restriction $e\left(g^{p'}, g\right) = e\left(g^{p}, g^{\alpha}\right)$
  - check polynomial cofactors $e\left(g^{p}, g\right) = e\left(g^{t(s)}, g^{h}\right)$

zkSnarkOP 双盲验证特性分析

达到的特性：

Prover有**多项式的系数**知识，这个多项式有d阶，且能够被t(x)整除

Verifier要能够在不知道这个知识的前提下，验证prover确实有这个知识。

Prover可能通过以下几种手段欺骗verifier:

1、返回一个任意的加密值E(P(s))，宣称这就是满足要求的

2、返回一个加密值E(t(s)*fakeh(s))，宣称这就是满足要求的

Verifier可能通过判断prover知道的知识不是V构造的，来获取有限的信息

应对措施

$$e(g^p, g) = e\left(g^{t(s)}, g^h\right)$$

Verifier检查p(s)=t(s)*h

$$e\left(g^{p'}, g\right) = e\left(g^p, g^\alpha\right)$$

d-KCA，将详细解释

$$\left(g^{\delta p(s)}, g^{\delta h(s)}, g^{\delta \alpha p(s)}\right)$$

Prover返回一个偏置混淆真实信息

zkSnarkOP d-KCA限制多项式阶

- 假设G是一个有限循环群，g是它的一个生成元
- 选取同态隐藏函数$E(x) = x \cdot g$
- Bob随机选择一个α和一个s，把生成的α对发送给Allice：

  $(a_0, b_0) = (E(1), \alpha \cdot E(1))$

  $(a_1, b_1) = (E(s), \alpha \cdot E(s))$

  …… ……

  $(a_d, b_d) = (E(s^d), \alpha \cdot E(s^d))$

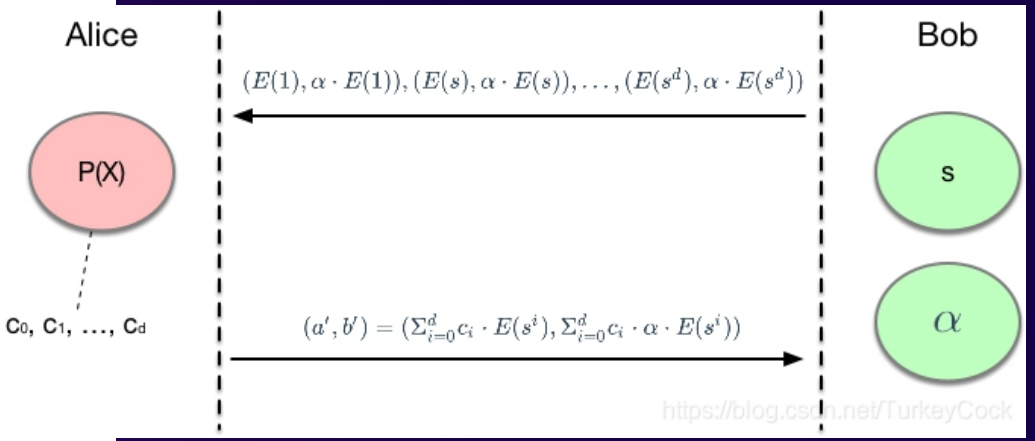
- Alice需要保守的秘密是$P(X)$的系数：$P(X) = c_0 + c_1 \cdot X + \ldots + c_d \cdot X^d$
- Alice计算新的α对：

  $a' = P(s) \cdot g = c_0 \cdot g + c_1 \cdot s \cdot g + \ldots + c_d \cdot s^d \cdot g = c_0 \cdot a_0 + c_1 \cdot a_1 + \ldots + c_d \cdot a_d = \Sigma_{i=0}^{d} c_i \cdot a_i$

  $b' = \alpha \cdot a' = \Sigma_{i=0}^{d} c_i \cdot \alpha \cdot a_i = \Sigma_{i=0}^{d} c_i \cdot b_i$

  然后把(a', b')发送给Bob

- Bob验证(a', b')是否是α对，如果是的话就接受该回复



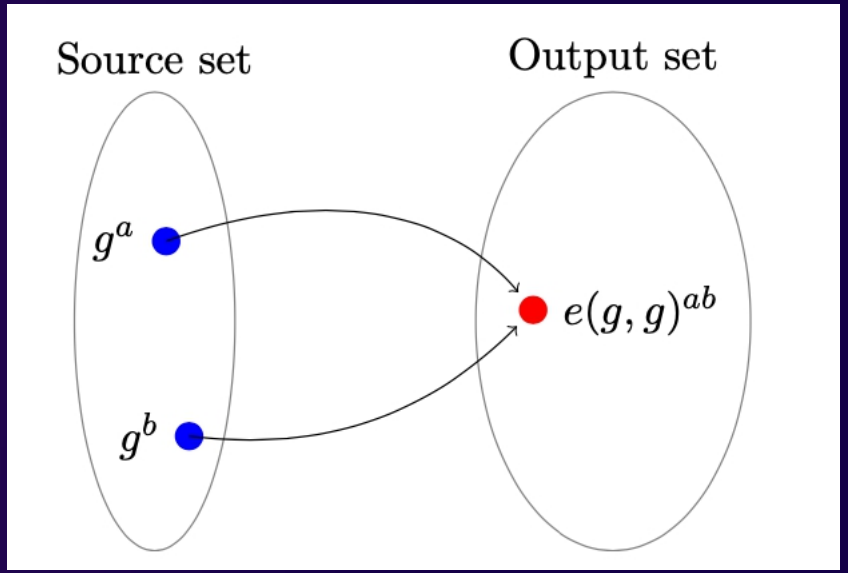
Prover因为不知道alpha，他只能执行一个d-阶的多项式，才能得到alpha*E(P(s))，因此alpha的作用在于限制了Prover使用的多项式阶，Prover就不可以使用P(x)*任意一个多项式来欺骗V

zkSnarkOP 双线性配对的必要性

Verifier需要对加密值E(h(s)), E(t(s))相乘的操作，如果E(x)=x*G，椭圆曲线加密，加密值是不能直接相乘的。因此引入pairing，

$$e(g^a, g^b) = \mathbf{g}^{ab}$$

注意的是，定义等式两边g都是代表有限椭圆曲线群的生成元，并不一定是同一个群。



zkSnarkOP难以验证复杂的计算问题

回顾P(x)=T(x)*H(x)，其中T(x)是公开信息，假设有d阶，Prover可以任意构造一个fakeH(x)，其阶=P的阶-d，Verifier得到了一对(a, b)

$$a' = P(s) \cdot g = c_0 \cdot g + c_1 \cdot s \cdot g + \ldots + c_d \cdot s^d \cdot g = c_0 \cdot a_0 + c_1 \cdot a_1 + \ldots + c_d \cdot a_d = \Sigma_{i=0}^{d} c_i \cdot a_i$$

$$b' = \alpha \cdot a' = \Sigma_{i=0}^{d} c_i \cdot \alpha \cdot a_i = \Sigma_{i=0}^{d} c_i \cdot b_i$$

这个新的(a, b) 都能被E(T(S))整除，并且是alpha对，而zkSNARKop是无法判断这个情况的。

要证明的多项式，必须要强制执行某种运算，而这种运算能被巧妙的证明。
即所谓的enforcing operation

如果只验证，$p(x) = l(x) \times r(x) - o(x)$
存在两个问题
1、隐藏了operation，prover可能构造任意一个被t(x)整除的多项式，但不一定满足上面的结构
2、在证明阶段，prover是不能使用加密结果的乘法的，即使使用了双线性配对的技巧，也只能用一次，即如果Prover使用了双线性配对的加密乘法，Verifier就不能再用

零知识证明概述

zkSNARKop

zkSNARK

zkSnark的实现步骤

A) 编码成一个多项式问题

把需要验证的程序编写成一个多项式方程：t(x) h(x) = w(x) v(x)，当且仅当程序的计算结果正确时这个等式才成立。证明者需要说服验证者这个等式成立。

B) 简单随机抽样

验证者会选择一个私密评估点 s 来将多项式乘法和验证多项式函数相等的问题简化成简单乘法和验证等式 t(s)h(s) = w(s)v(s) 的问题。

这样做不但可以减小证明量，还可以大量的减少验证所需的时间。

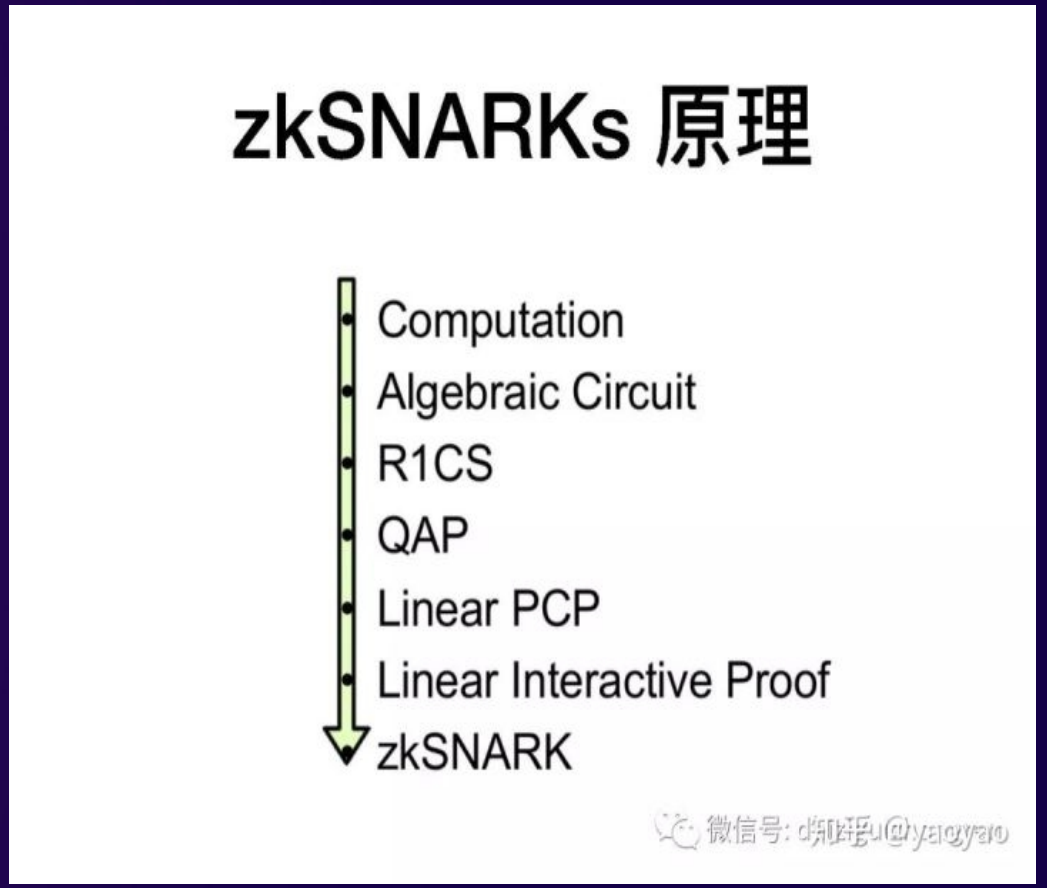
C) 同态（Homomorphic）编码 / 加密

我们使用一个拥有一些同态属性的（并不需要完全同态，因为完全同态目前尚不实用）编码 / 加密函数 E。这个函数允许证明者在不知道 s 的情况下计算 E(t(s)), E(h(s)), E(w(s)), E(v(s))，她只知道 E(s) 和一些其他有用的加密信息。

D) 零知识

证明者通过乘以一个数来替换 E(t(s)), E(h(s)), E(w(s)), E(v(s)) 的值，这样验证者就可以在不知道真实的编码值的情况下验证他们正确的**结构**了。

有一个粗糙的想法是这样的，因为校验 t(s)h(s) = w(s)v(s) 和校验 t(s)h(s) k = w(s)v(s) k（对于一个不等于 0 的私密的随机数 k 来说）几乎是完全一样的，而不同的地方在于如果你只接收到了 (t(s)h(s) k) 和 (w(s)v(s) k) 那么从中获取到 t(s)h(s) 或者 w(s)v(s) 的值就几乎是不可能了。



从通用计算问题到多项式

Algorithm 1 Operation depends on an input

```
function calc(w, a, b)
    if w then
        return a × b
    else
        return a + b
    end if
end function
```

Step1: 用多项式方程描述

$$w \times (a \times b) + (1 - w) \times (a + b) = v$$

Step2: 每个operation约束为一个乘法
转换为多个operation组

$$\begin{aligned}
1: &\quad 1 \cdot a \times 1 \cdot b = 1 \cdot m \\
2: &\quad 1 \cdot w \times 1 \cdot m + -1 \cdot a + -1 \cdot b = 1 \cdot v + -1 \cdot a + -1 \cdot b \\
3: &\quad 1 \cdot w \times 1 \cdot w = 1 \cdot w
\end{aligned}$$

成（span）表达式

$$\begin{aligned}
L(x) &= a \cdot l_a(x) + w \cdot l_w(x) \\
R(x) &= m \cdot r_m(x) + a \cdot r_a(x) + b \cdot r_b(x) + w \cdot r_w(x) \\
O(x) &= m \cdot o_m(x) + v \cdot o_v(x) + a \cdot o_a(x) + b \cdot o_b(x) + w \cdot o_w(x)
\end{aligned}$$

Step4: 计算“基”多项式

$$\begin{aligned}
l_a(x) &= \frac{1}{2}x^2 - \frac{5}{2}x + 3; & r_m(x) &= -x^2 + 4x - 3; & o_m(x) &= \frac{1}{2}x^2 - \frac{5}{2}x + 3; \\
l_w(x) &= -\frac{1}{2}x^2 + \frac{5}{2}x - 2; & r_a(x) &= x^2 - 4x + 3; & o_v(x) &= -x^2 + 4x - 3; \\
& & r_b(x) &= \frac{3}{2}x^2 - \frac{13}{2}x + 6; & o_a(x) &= x^2 - 4x + 3; \\
& & r_w(x) &= \frac{1}{2}x^2 - \frac{3}{2}x + 1; & o_b(x) &= x^2 - 4x + 3; \\
& & & & o_w(x) &= -\frac{1}{2}x^2 - \frac{3}{2}x + 1;
\end{aligned}$$

Step5: 计算多项式L,R,

$$\begin{aligned}
&l_a(1) = 1;\ l_a(2) = 0;\ l_a(3) = 0;\ r_m(1) = 0;\ r_m(2) = 1;\ r_m(3) = 0;\ o_m(1) = 1;\ o_m(2) = 0;\ o_m(3) = 0; \\
&l_w(1) = 0;\ l_w(2) = 1;\ l_w(3) = 1;\ r_a(1) = 0;\ r_a(2) = -1;\ r_a(3) = 0;\ o_v(1) = 0;\ o_v(2) = 1;\ o_v(3) = 0; \\
&r_b(1) = 1;\ r_b(2) = -1;\ r_b(3) = 0;\ o_a(1) = 0;\ o_a(2) = -1;\ o_a(3) = 0; \\
&r_w(1) = 0;\ r_w(2) = 0;\ r_w(3) = 1;\ o_b(1) = 0;\ o_b(2) = -1;\ o_b(3) = 0; \\
&o_w(1) = 0;\ o_w(2) = 0;\ o_w(3) = 1;
\end{aligned}$$

$$L(x) = 3 \cdot l_a(x) + 1 \cdot l_w(x) = x^2 - 5x + 7$$

$$R(x) = 6 \cdot r_m(x) + 3 \cdot r_a(x) + 2 \cdot r_b(x) + 1 \cdot r_w(x) = \frac{1}{2}x^2 - 2\frac{1}{2}x + 4$$

$$O(x) = 6 \cdot o_m(x) + 6 \cdot o_v(x) + 3 \cdot o_a(x) + 2 \cdot o_b(x) + 1 \cdot o_w(x) = 2\frac{1}{2}x^2 - 12\frac{1}{2}x + 16$$

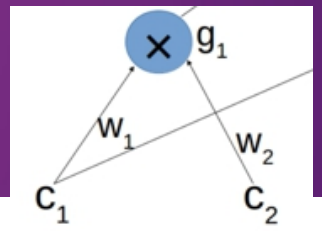
$$h(x) = \frac{L(x) \times R(x) - O(x)}{t(x)} = \frac{\frac{1}{2}x^4 - 5x^3 + \frac{35}{2}x^2 - 25x + 12}{(x-1)(x-2)(x-3)} = \frac{1}{2}x - 2$$

算术电路和“基”多项式

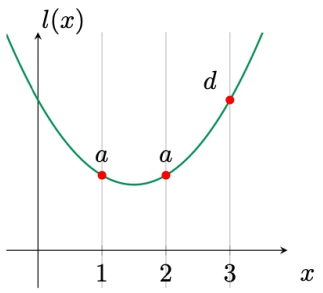
left operand **operator** right operand = output

单个的计算问题都可以抽象如上。

$$l(x) \textbf{ operator } r(x) = o(x)$$



$$\begin{aligned} a \times b &= r_1 \\ a \times c &= r_2 \\ d \times c &= r_3 \end{aligned}$$



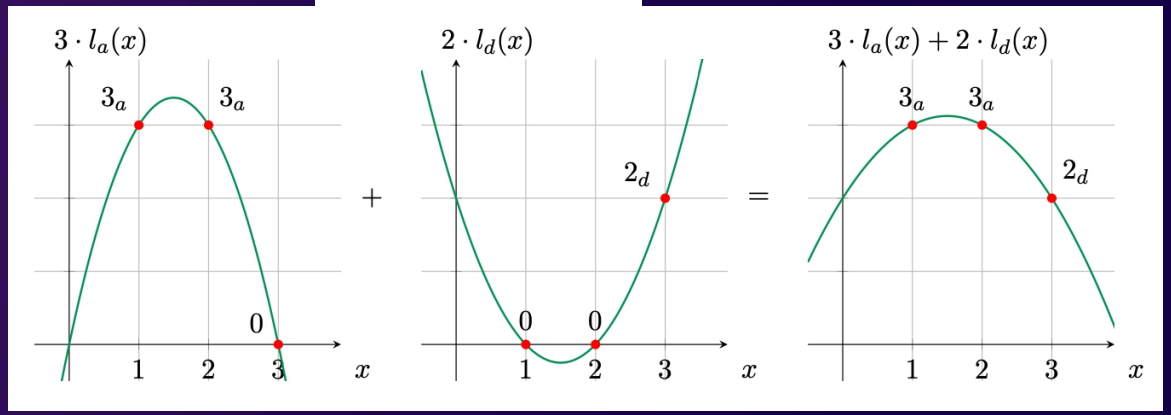
基多项式la(x)的特点：

- *在x=1,2,3,...,d 的时候，la(x)=0或者1*
- *所有的+-*/都可以用乘法表达，一个操作*对应一个门
- *每个操作由约束一个乘法，由两个基多项式相乘*

$$l(x) \times r(x) - o(x) = t(x)h(x)$$

$$L(x) = a\,l_a(x) + d\,l_d(x),$$

$a = 3$ and $d = 2$:



知识：从一个多项式的系数到一组“基”多项式的系数

- Setup 构造“基”多项式
  - construct *variable polynomials* for left operand $\{l_i(x)\}_{i\in\{1,\dots,n\}}$ such that for all operations $j \in \{1,\dots,d\}$ they evaluate to corresponding coefficients, i.e., $l_i(j) = c_{\mathrm{L},i,j}$, and similarly for right operand and output
  - sample random s, α
  - calculate $t(x) = (x-1)(x-2)\dots(x-d)$ and its evaluation $g^{t(s)}$
  - compute proving key: $\left(\left\{g^{s^k}\right\}_{k\in[d]}, \left\{g^{l_i(s)}, g^{r_i(s)}, g^{o_i(s)}, g^{\alpha l_i(s)}, g^{\alpha r_i(s)}, g^{\alpha o_i(s)}\right\}_{i\in\{1,\dots,n\}}\right)$
  - compute verification key: $\left(g^{t(s)}, g^{\alpha}\right)$

- Proving 知识为基多项式的系数vi
  - compute function $f(*)$ and therefore corresponding variables values $\{v_i\}_{i\in\{1,\dots,n\}}$
  - calculate $h(x) = \frac{L(x)\times R(x) - O(x)}{t(x)}$, where $L(x) = \sum_{i=1}^{n} v_i \cdot l_i(x)$, and similarly $R(x), O(x)$
  - assign variable values and *sum up* to get operand polynomials:
    $$g^{L(s)} = \left(g^{l_1(s)}\right)^{v_1} \cdots \left(g^{l_n(s)}\right)^{v_n}, \quad g^{R(s)} = \prod_{i=1}^{n}\left(g^{r_i(s)}\right)^{v_i}, \quad g^{O(s)} = \prod_{i=1}^{n}\left(g^{o_i(s)}\right)^{v_i}$$
  - assign variable values to the shifted polynomials:
    $$g^{\alpha L(s)} = \prod_{i=1}^{n}\left(g^{\alpha l_i(s)}\right)^{v_i}, \quad g^{\alpha R(s)} = \prod_{i=1}^{n}\left(g^{\alpha r_i(s)}\right)^{v_i}, \quad g^{\alpha O(s)} = \prod_{i=1}^{n}\left(g^{\alpha o_i(s)}\right)^{v_i}$$
  - calculate encrypted evaluation $g^{h(s)}$ using provided powers of s: $\left\{g^{s^k}\right\}_{k\in[d]}$
  - set proof: $\left(g^{L(s)}, g^{R(s)}, g^{O(s)}, g^{\alpha L(s)}, g^{\alpha R(s)}, g^{\alpha O(s)}, g^{h(s)}\right)$

- Verification
  - parse proof as $\left(g^{L}, g^{R}, g^{O}, g^{L'}, g^{R'}, g^{O'}, g^{h}\right)$
  - variable polynomials restriction check:
    $e(g^L, g^\alpha) = e(g^{L'}, g), \quad e(g^R, g^\alpha) = e(g^{R'}, g), \quad e(g^O, g^\alpha) = e(g^{O'}, g)$
  - valid operations check:
    $e(g^L, g^R) = e(g^t, g^h) \cdot e(g^O, g)$

保证知识在等式两边的一致性

为了保证variable consistency polynomials, 即L(x), R(x), O(x)所用到的基多项式系数，又称variables, 也即是我们要证明的“知识”，不被篡改，用前面d-KCA工具可以保证

- Setup
  - ... sample random $\beta_l, \beta_r, \beta_o$
  - calculate, encrypt and add to the proving key the *variable consistency polynomials*: $\left\{g^{\beta_l l_i(s)+\beta_r r_i(s)+\beta_o o_i(s)}\right\}_{i\in\{1,\ldots,n\}}$
  - encrypt β-s and add to the verification key: $\left(g^{\beta_l}, g^{\beta_r}, g^{\beta_o}\right)$

- Proving
  - ... assign variable values to the *variable consistency polynomials*: $g^{z_i(s)} = \left(g^{\beta_l l_i(s)+\beta_r r_i(s)+\beta_o o_i(s)}\right)^{v_i}$ for $i \in \{1, \ldots, n\}$
  - add assigned polynomials in encrypted space: $g^{Z(s)} = \prod_{i=1}^{n} g^{z_i(s)} = g^{\beta_l L(s)+\beta_r R(s)+\beta_o O(s)}$
  - add to the proof: $g^{Z(s)}$
- Verification
  - ... check the consistency between provided *operand polynomials* and the “checksum” polynomial: $e\left(g^L, g^{\beta_l}\right) \cdot e\left(g^R, g^{\beta_r}\right) \cdot e\left(g^O, g^{\beta_o}\right) = e\left(g^Z, g\right)$ which is equivalent to: $e\left(g, g\right)^{\beta_l L+\beta_r R+\beta_o O} = e\left(g, g\right)^Z$

延展性问题Malleability

前述方案存在两个延展性问题，根源是一致的：
1、不同操作里，variable polynomial的系数可能设置不同的值
2、L，R，O的variable polynomial的系数可能设置不同的值

根本原因在于前面解决方案里，prover拿到的是明文beta，而相比我们已经掌握的d-KCA，prover是对加密的alpha-shift进行操作。

- Setup
  - ...sample random $\beta, \gamma, \rho_l, \rho_r$ and set $\rho_o = \rho_l \cdot \rho_r$
  - set generators $g_l = g^{\rho_l}, g_r = g^{\rho_r}, g_o = g^{\rho_o}$
  - set proving key: $\left(\left\{g^{s^k}\right\}_{k\in[d]}, \left\{g_l^{l_i(s)}, g_r^{r_i(s)}, g_o^{o_i(s)}, g_l^{\alpha_l l_i(s)}, g_r^{\alpha_r r_i(s)}, g_o^{\alpha_o o_i(s)}, g_l^{\beta l_i(s)} \cdot g_r^{\beta r_i(s)} \cdot g_o^{\beta o_i(s)}\right\}\right)$
  - set verification key: $\left(g_o^{t(s)}, g^{\alpha_l}, g^{\alpha_r}, g^{\alpha_o}, g^{\beta\gamma}, g^{\gamma}\right)$

- Proving
  - ...assign variable values

    $$g^{Z(s)} = \prod_{i=1}^{n} \left(g_l^{\beta l_i(s)} \cdot g_r^{\beta r_i(s)} \cdot g_o^{\beta o_i(s)}\right)^{v_i}$$

- Verification
  - ...variable polynomials restriction check:

    $e\left(g_l^{L'}, g\right) = e\left(g_l^{L}, g^{\alpha_l}\right)$, and similarly for g_r^R, g_o^O
  - variable values consistency check:

    $e\left(g_l^{L} \cdot g_r^{R} \cdot g_o^{O}, g^{\beta\gamma}\right) = e\left(g^{Z}, g^{\gamma}\right)$
  - valid operations check:

    $e\left(g_l^{L} \cdot g_r^{R}\right) = e\left(g_o^{t}, g^{h}\right) e\left(g_o^{O}, g\right) \Rightarrow$

    $e(g,g)^{\rho_l \rho_r LR} = e(g,g)^{\rho_l \rho_r th + \rho_l \rho_r O}$

zkSNARK-setup构造多项式

选择椭圆曲线有限域、双线性配对

- select a generator g and a cryptographic pairing e
- for a function $f(u) = y$ with n total variables of which m are input/output variables, convert into the polynomial form[27] $\left(\{l_i(x), r_i(x), o_i(x)\}_{i\in\{0,\dots,n\}}, t(x)\right)$ of degree d (equal to the number of operations) and size $n+1$

R1CS->QAP

- sample random $s, \rho_l, \rho_r, \alpha_l, \alpha_r, \alpha_o, \beta, \gamma$

s:完全随机　　Alpha：d-KCA

- set $\rho_o = \rho_l \cdot \rho_r$ and the operand generators $g_l = g^{\rho_l}, g_r = g^{\rho_r}, g_o = g^{\rho_o}$

双线性配对

- set the proving key:

$$\left(\left\{g^{s^k}\right\}_{k\in[d]}, \left\{g_l^{l_i(s)}, g_r^{r_i(s)}, g_o^{o_i(s)}\right\}_{i\in\{0,\dots,n\}}, \right.$$
$$\left\{g_l^{\alpha_l l_i(s)}, g_r^{\alpha_r r_i(s)}, g_o^{\alpha_o o_i(s)}, g_l^{\beta l_i(s)} g_r^{\beta r_i(s)} g_o^{\beta o_i(s)}\right\}_{i\in\{m+1,\dots,n\}},$$
$$\left. g_l^{t(s)}, g_r^{t(s)}, g_o^{t(s)}, g_l^{\alpha_l t(s)}, g_r^{\alpha_r t(s)}, g_o^{\alpha_o t(s)}, g_l^{\beta t(s)}, g_r^{\beta t(s)}, g_o^{\beta t(s)} \right)$$

- set the verification key:

$$\left(g^1, g_o^{t(s)}, \left\{g_l^{l_i(s)}, g_r^{r_i(s)}, g_o^{o_i(s)}\right\}_{i\in\{0,\dots,m\}}, g^{\alpha_l}, g^{\alpha_r}, g^{\alpha_o}, g^{\gamma}, g^{\beta\gamma}\right)$$

zkSNARK-prove

- for the input u, execute the computation of $f(u)$ obtaining values $\{v_i\}_{i\in\{m+1,\dots,n\}}$ for all the itermediary variables
- assign all values to the unencrypted variable polynomials $L(x) = l_0(x) + \sum_{i=1}^{n} v_i \cdot l_i(x)$ and similarly $R(x), O(x)$
- sample random δ_l, δ_r and δ_o
- find $h(x) = \dfrac{L(x)R(x) - O(x)}{t(x)} + \delta_r L(x) + \delta_l R(x) + \delta_l \delta_r t(x) - \delta_o$
- assign the prover's variable values to the encrypted variable polynomials and apply *zero-knowledge* δ-shift $g_l^{L_p(s)} = \left(g_l^{t(s)}\right)^{\delta_l} \cdot \prod_{i=m+1}^{n} \left(g_l^{l_i(s)}\right)^{v_i}$ and similarly $g_r^{R_p(s)}$, $g_o^{O_p(s)}$
- assign its α-shifted pairs $g_l^{L'_p(s)} = \left(g_l^{\alpha_l t(s)}\right)^{\delta_l} \cdot \prod_{i=m+1}^{n} \left(g_l^{\alpha_l l_i(s)}\right)^{v_i}$ and similarly $g_r^{R'_p(s)}$, $g_o^{O'_p(s)}$
- assign the variable values consistency polynomials

  $$g^{Z(s)} = \left(g_l^{\beta t(s)}\right)^{\delta_l} \left(g_r^{\beta t(s)}\right)^{\delta_r} \left(g_o^{\beta t(s)}\right)^{\delta_o} \cdot \prod_{i=m+1}^{n} \left(g_l^{\beta l_i(s)} g_r^{\beta r_i(s)} g_o^{\beta o_i(s)}\right)^{v_i}$$

- compute the proof $\left(g_l^{L_p(s)}, g_r^{R_p(s)}, g_o^{O_p(s)}, g^{h(s)}, g_l^{L'_p(s)}, g_r^{R'_p(s)}, g_o^{O'_p(s)}, g^{Z(s)}\right)$

zkSNARK-verify

- parse a provided proof as $\left(g_l^{L_p}, g_r^{R_p}, g_o^{O_p}, g^h, g_l^{L'_p}, g_r^{R'_p}, g_o^{O'_p}, g^Z\right)$
- assign input/output values to verifier's encrypted polynomials and add to 1:
  $g_l^{L_v(s)} = g_l^{l_0(s)} \cdot \prod_{i=1}^{m} \left(g_l^{l_i(s)}\right)^{v_i}$ and similarly for $g_r^{R_v(s)}$ and $g_o^{O_v(s)}$
- variable polynomials restriction check :
  $e\left(g_l^{L_p}, g^{\alpha_l}\right) = e\left(g_l^{L'_p}, g\right)$ and similarly for $g_r^{R_p}$ and $g_o^{O_p}$
- variable values consistency check:
  $e\left(g_l^{L_p} g_r^{R_p} g_o^{O_p}, g^{\beta\gamma}\right) = e\left(g^Z, g^\gamma\right)$

Alpha : d-KCA

- valid operations check:
  $e\left(g_l^{L_p} g_l^{L_v(s)}, g_r^{R_p} g_r^{R_v(s)}\right) = e\left(g_o^{t(s)}, g^h\right) \cdot e\left(g_o^{O_p} g_o^{O_v(s)}, g\right)$

Filecion- PoREP setup工作

算法：

1、在setup阶段，利用了zkSnark的Prove，计算π_{SEAL}
2、R是通过<私钥, D|计算出来的merkle承诺，对D实现了压缩
3、原始数据D，通过zkSnark，实现了压缩，因为只保留π_{SEAL}
4、w向量，即witness，merkle承诺需要提供的路径

注意1：PoREP的setup，不同于zkSnark的setup仅仅构造基多项式和ProveKey, VerifyKey，而是将原始数据通过Merkle承诺压缩，提交到链上的智能合约。（注：precommit2完成的工作）
注意2：不同上下文的key 含义

Filecoin PoRep protocol

Setup

- INPUTS:
  - prover key pair $(\mathsf{pk}_{\mathcal{P}}, \mathsf{sk}_{\mathcal{P}})$
  - prover SEAL key $\mathsf{pk}_{\mathsf{SEAL}}$
  - data $\mathcal{D}$
- OUTPUTS: replica $\mathcal{R}$, Merkle root rt of $\mathcal{R}$, proof π_{SEAL}

1) Compute $h_{\mathcal{D}} := \mathsf{CRH}(\mathcal{D})$
2) Compute $\mathcal{R} := \mathsf{Seal}^{\tau}(\mathcal{D}, \mathsf{sk}_{\mathcal{P}})$
3) Compute $\mathsf{rt} := \mathsf{MerkleCRH}(\mathcal{R})$
4) Set $\vec{x} := (\mathsf{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathsf{rt})$
5) Set $\vec{w} := (\mathsf{sk}_{\mathcal{P}}, \mathcal{D})$
6) Compute $\pi_{\mathsf{SEAL}} := \mathsf{SCIP.Prove}(\mathsf{pk}_{\mathsf{SEAL}}, \vec{x}, \vec{w})$
7) Output $\mathcal{R}$, rt, π_{SEAL}

Filecion- PoREP prove工作

算法：

1、在prove阶段，利用了zkSnark的Prove，计算π_{POS}
2、R保留在本地，对D实现了压缩
3、计算一个随机节点到Root的路径path
4、w向量，即witness，merkle承诺需要提供的路径

注：commit完成的工作

Prove

- INPUTS:
  - prover *Proof-of-Storage* key $\mathsf{pk}_{\mathsf{POS}}$
  - replica $\mathcal{R}$
  - random challenge c
- OUTPUTS: a proof π_{POS}

1) Compute $\mathsf{rt} := \mathsf{MerkleCRH}(\mathcal{R})$
2) Compute $\mathsf{path} :=$ Merkle path from rt to leaf $\mathcal{R}_c$
3) Set $\vec{x} := (\mathsf{rt}, c)$
4) Set $\vec{w} := (\mathsf{path}, \mathcal{R}_c)$
5) Compute $\pi_{\mathsf{POS}} := \mathsf{SCIP.Prove}(\mathsf{pk}_{\mathsf{POS}}, \vec{x}, \vec{w})$
6) Output π_{POS}

zkSNARK的proving key

Filecion- PoREP verify工作

算法：

1、在verify阶段，利用了zkSnark的verify，验证π_{SEAL} π_{POS}
2、链上合约里只保留hD, rt, pkp, vkseal, vkpos

注：智能合约完成的工作

Verify

- INPUTS:
  - prover public key, $\mathsf{pk}_{\mathcal{P}}$
  - verifier SEAL and POS keys $\mathsf{vk}_{\mathsf{SEAL}}$, $\mathsf{vk}_{\mathsf{POS}}$
  - hash of data $\mathcal{D}$, $h_{\mathcal{D}}$
  - Merkle root of replica $\mathcal{R}$, rt
  - random challenge, c
  - tuple of proofs, $(\pi_{\mathsf{SEAL}}, \pi_{\mathsf{POS}})$
- OUTPUTS: bit b, equals 1 if proofs are valid

1) Set $\vec{x_1} := (\mathsf{pk}_{\mathcal{P}}, h_{\mathcal{D}}, \mathsf{rt})$
2) Compute $b_1 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{SEAL}}, \vec{x_1}, \pi_{\mathsf{SEAL}})$
3) Set $\vec{x_2} := (\mathsf{rt}, c)$
4) Compute $b_2 := \mathsf{SCIP.Verify}(\mathsf{vk}_{\mathsf{POS}}, \vec{x_2}, \pi_{\mathsf{POS}})$
5) Output $b_1 \wedge b_2$

椭圆曲线Key

zkSNARK的verify key

Filecion- PoREP 代码

TBD

谢 谢 观 看