Core Program Week 3

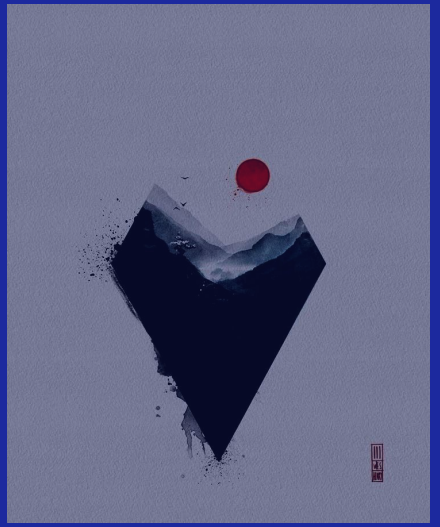
コミットメントスキーム
・PLONK

ZK Tokyo

目次

- ゼロ知識証明とは（recap）
- ゼロ知識証明のつくりかた
- ゼロ知識性はどこからくるのか: コミットメントスキーム
- さまざまな証明スキーム: PLONK
- 演習問題

ゼロ知識証明とは（ recap）



ゼロ知識証明とは (recap)

ゼロ知識証明　=　1）証明　+　2）ゼロ知識性

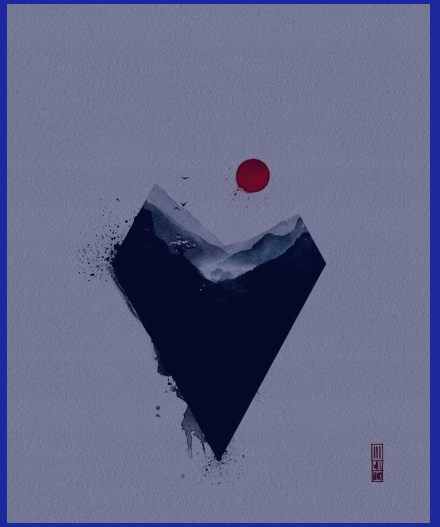
1. 証明
   - さまざまな証明種別（e.g., 社会的証明、**数学的証明**）
   - 証明が満たすべき性質
     - **完全性 (completeness)**: 主張が真であれば、常に証明可能である（ i.e., 証明不可能 => 偽）
     - **健全性 (soundness)**: 証明可能であれば、主張は真である（ i.e., 偽 => 証明不可能）
   - 証明のその他の性質
     - 証明を生成することは難しく、証明を検証することは簡単（ e.g., NP問題）
     - 証明者（Prover）と検証者（Verifier）間のやりとりとして計算問題を捉えると、
       さまざまな計算複雑性クラスが定義できる（ e.g. IP、MIP、PCP、IOP）

ゼロ知識証明とは (recap)

ゼロ知識証明　=　1）証明　+　2）ゼロ知識性

2. ゼロ知識性
   - 定義: 証明プロセスにおいて検証者が、**証明された主張が真であること**以外、何の情報も得られないこと（e.g., 所属証明）
   - さまざまなゼロ知識性の定義: シミュレーターアルゴリズムの定義
     - 完全ゼロ知識性 (perfect zero-knowledge)
     - 統計的ゼロ知識性 (statistical zero-knowledge)
     - **計算的ゼロ知識性 (computational zero-knowledge)**
   - ゼロ知識性の定義におけるさまざまな仮定（e.g., 検証者の正直性）

ゼロ知識証明のつくりかた



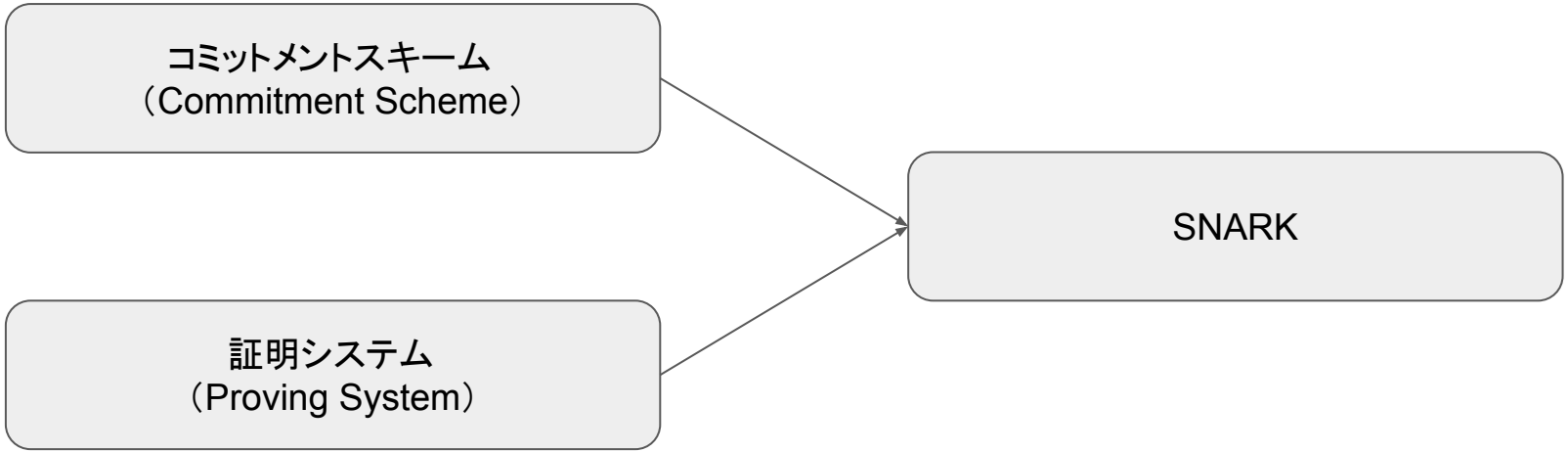
ゼロ知識証明のつくりかた

SNARK (Succinct Non-interactive ARgument of Knowledge)

- 簡潔性 (succinctness)
  - 証明サイズが小さいこと: **O(1)**
  - 証明の検証時間が短いこと: **O(logN)**
- 非対話性 (non-interactiveness)
  - 証明者が生成した証明が、**誰でも検証可能（ publicly-verifiable）**であること
- 知識の証明 (argument of knowledge)
  - 単に主張が真であることの証明ではなく、**主張が真となるような知識を証明者が知っていること** の証明であること（e.g., 数独）

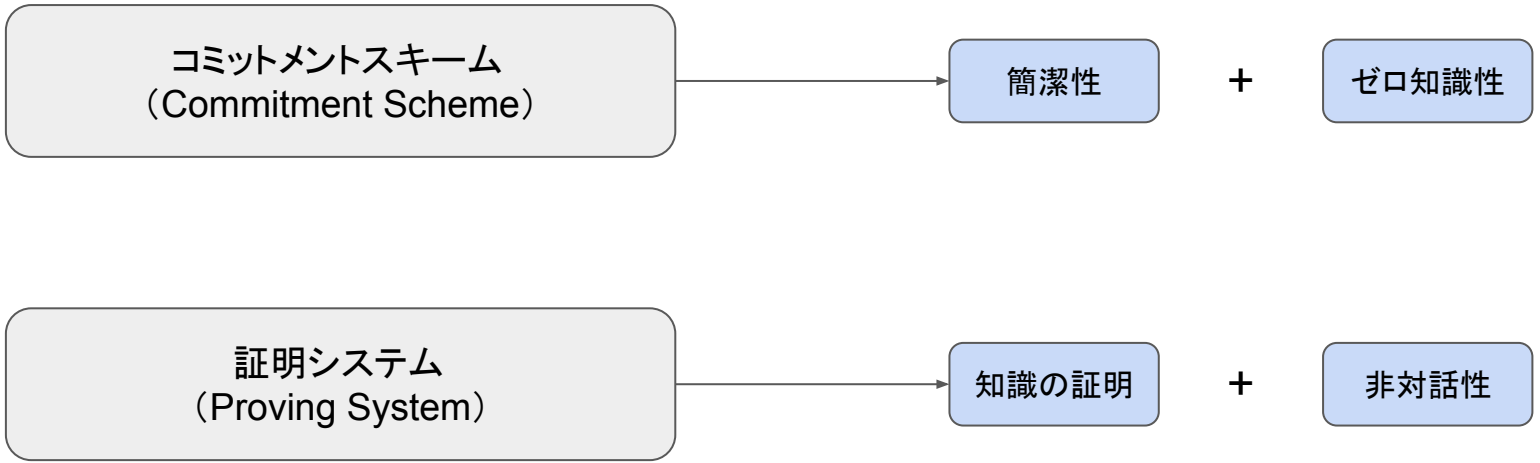
ゼロ知識証明のつくりかた: zkSNARK

zkSNARK = コミットメントスキーム + 証明システム

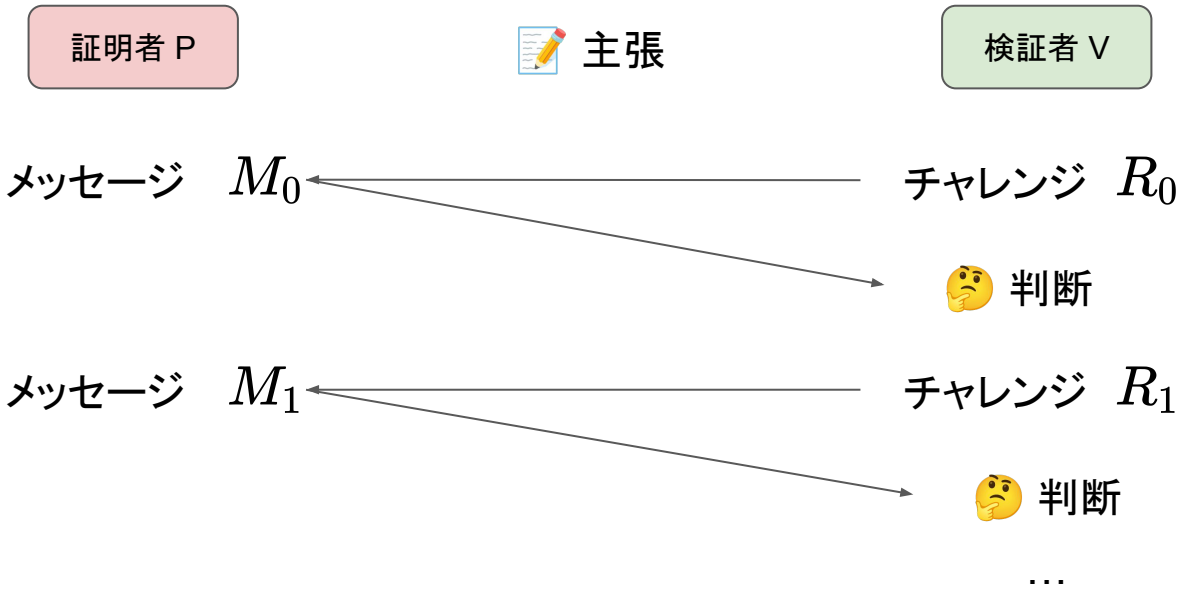


ゼロ知識証明のつくりかた: zkSNARK

zkSNARK = コミットメントスキーム + 証明システム

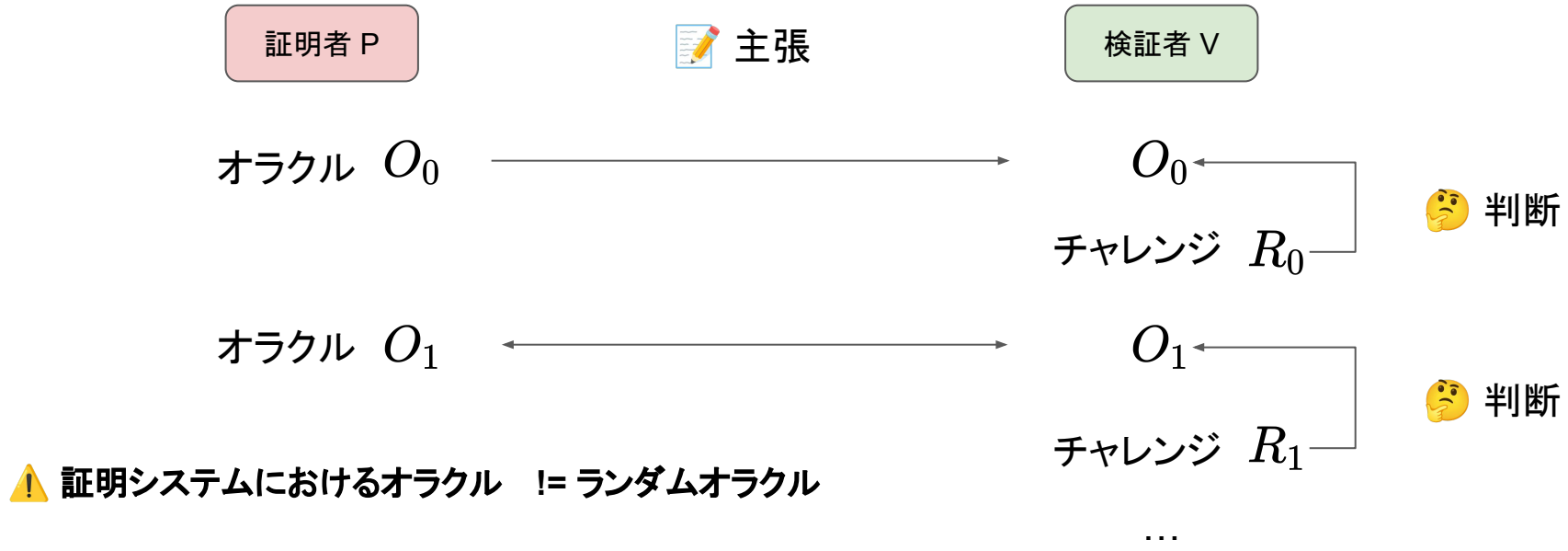


ゼロ知識証明のつくりかた: 証明システム



対話的証明 (Interactive Proofs)・1985

ゼロ知識証明のつくりかた: 証明システム

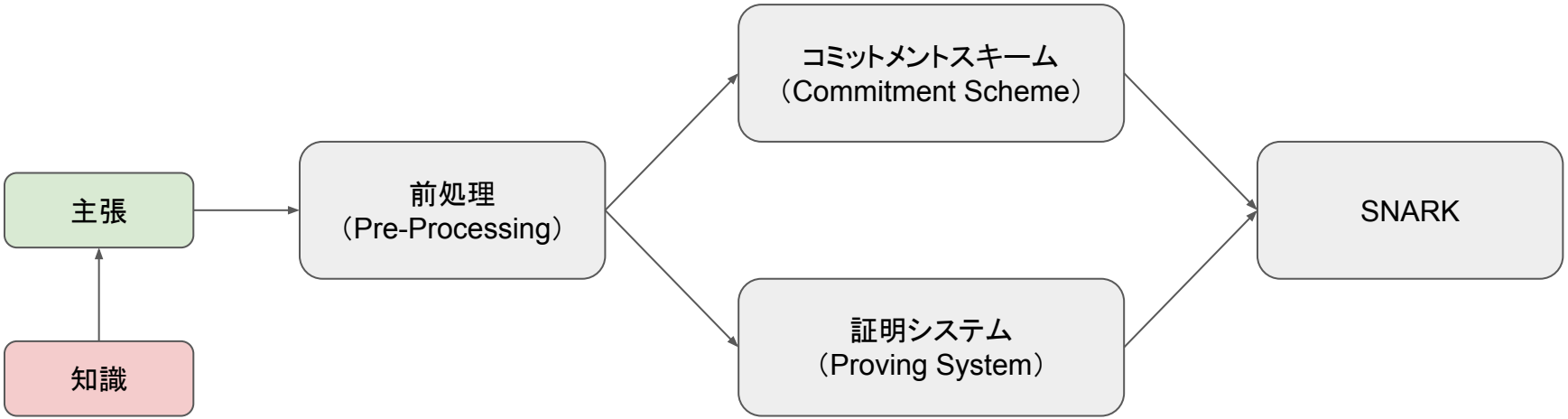


⚠️ **証明システムにおけるオラクル　!= ランダムオラクル**

対話的オラクル証明 (Interactive Oracle Proofs)・2016

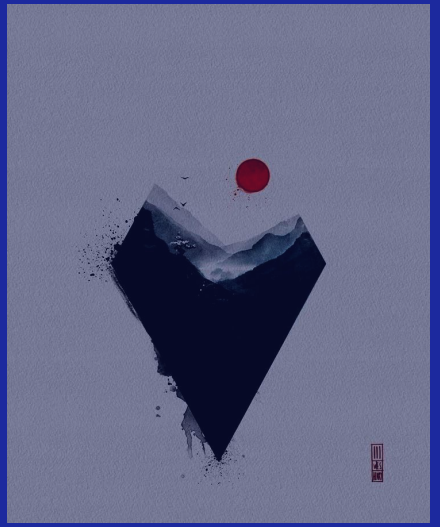
ゼロ知識証明のつくりかた: zkSNARK

zkSNARK = コミットメントスキーム + 証明システム



ZK Tokyo

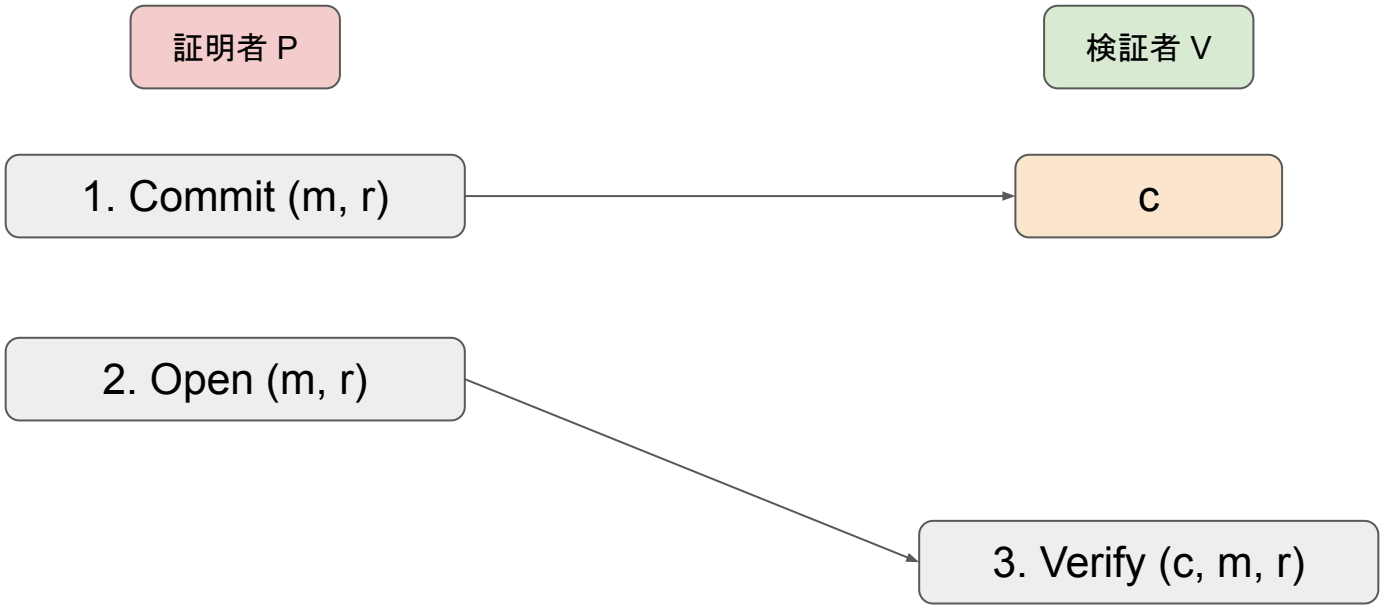
ゼロ知識性はどこからくるのか：コミットメントスキーム



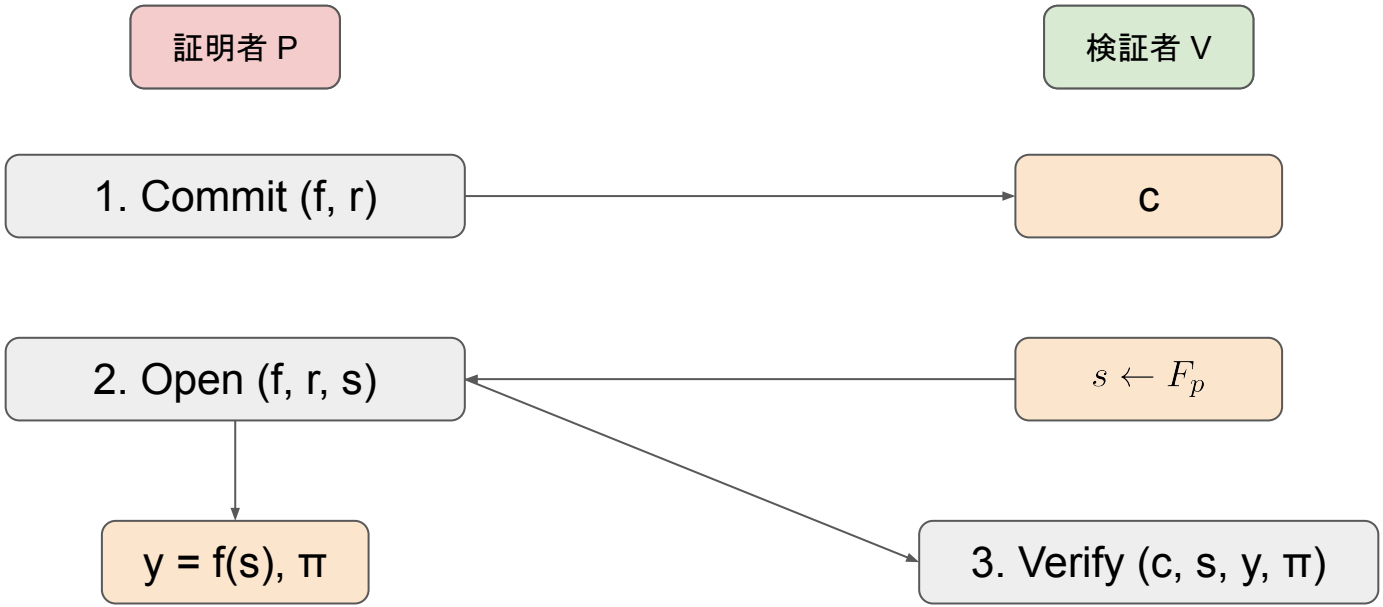
コミットメントスキーム（Commitment Scheme）

- さまざまなコミットメントスキーム
  - 単一の値に対するコミットメント : ビットコミットメント、ハッシュコミットメント …
  - 複数の値に対するコミットメント : ベクトルコミットメント、**多項式コミットメント** …
- コミットメントスキームが満たすべき性質
  - **束縛性 (binding)**: 一度コミットするとコミットした値は変更できない
  - **秘匿性 (hiding)**: コミットメントからは元の値は割り出せない
- ブロックチェーン領域においてコミットメントスキームが満たすべき性質
  - コミットメントのサイズが小さいこと（ e.g., 群の1要素）
  - **効率的な評価検証** が可能であること

コミットメントスキームの流れ: イメージ



多項式コミットメントスキームの流れ



KZG（Kate）コミットメントスキーム

- 多項式コミットメントスキーム（PCS）のひとつ
  - 多項式インタラクティブオラクルプルーフ（PIOP）と組み合わせることができる（i.e., PLONK）
  - 生成されるコミットメントサイズが多項式の次数に依らず一定 O(1)
- コミット前に信頼されたセットアップ（Trusted Setup）が必要
  - セキュリティ向上のため、MPC 経由で生成することが一般的（e.g., Powers-of-Tau）
- 基本的な流れ
  1. Setup: MPC セッションは、2~4 のステップで必要となるパブリックパラメーターを計算する
  2. Commit: 証明者は、多項式に対してコミットする
  3. Open: 証明者は、検証者によって選ばれたランダムな値で元の多項式を実行し、その証明を生成する
  4. Verify: 検証者は、3で生成された評価証明の正しさを検証する

1. セットアップ（Trusted Setup）

A. パブリックパラメーターの選出

セットアップの最終的な目的 **SRS (Structured Reference String)** の算出
→ SRS の算出に必要な値や数学的構造を固定する

$$pp = \begin{cases} p & \longrightarrow \text{有限体の位数となる十分に大きい素数} \\ \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T & \longrightarrow \text{2つの楕円曲線上の加法群・1つの乗法群} \\ g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2 & \longrightarrow \text{楕円曲線群の生成元} \\ d \in \mathbb{N}_p & \longrightarrow \text{コミットされる多項式の最大次数} \end{cases}$$

1. セットアップ（Trusted Setup）

A. パブリックパラメーターの選出

KZG はコミットメントの検証時に**楕円曲線ペアリング**を用いるため、
パラメーターとして選出される3つの群は、**効率的に計算できる**
ペアリングが存在するような楕円曲線の組み合わせ を選ぶ必要がある

$$e(\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T)$$

KZGを含めた暗号理論では、主に楕円曲線ペアリングの**双線形性（bilinearity）**
の性質を検証等の計算で活かすことが多い

$$e(a \cdot x, b \cdot y) = e(x, y)^{a \cdot b}$$

$$\forall x \in \mathbb{G}_1, \forall y \in \mathbb{G}_2$$

1. セットアップ（Trusted Setup）

B. SRS (Structured Reference String) の計算 → 実際のセレモニー例

A で設定した値を元にSRS を算出するために、
まずは有限体からランダムに1つの要素を選ぶ
→ この値は**有毒廃棄物 (toxic waste)** と呼ばれ、**誰にも知られてはいけない**

$$\tau \leftarrow \mathbb{F}_p$$

次にこの値を用いて、SRS を以下のように算出する
→ SRS は、証明者と検証者に公開され、これ以降のステップで利用される

$$SRS = \{\tau \cdot g_1, \tau^2 \cdot g_1, \tau^3 \cdot g_1, \ldots, \tau^d \cdot g_1, g_2, \tau \cdot g_2\}$$

2. 多項式へのコミット（commit）

証明者は、コミット対象となる**一変量多項式（univariate polynomial）**を用意する
→ 多項式の最大次数は、**セットアップで定義された最大次数以下** である必要がある

$$f(X) := \Sigma_{i=0}^{d} c_i X^i$$
$$= c_0 + c_1 \cdot X + c_2 \cdot X^2 + c_3 \cdot X^3 + \ldots + c_d \cdot X^d$$

多項式に対するコミットメントは↓のように計算される
→ コミットメント Com_f は、楕円曲線群の生成元 g_1 に、多項式 f を τ で評価した結果を掛けた値になる（i.e., Com_f は、**楕円曲線群の1つの要素**に集約される）

$$Com_f := f(\tau) \cdot g_1$$

2. 多項式へのコミット（commit）

ここで証明者は τ を知らないため、前ページの計算はできないようにみえるが、
コミットメント Com_f を展開してみると..

$$
\begin{aligned}
Com_f &:= f(\tau) \cdot g_1 \\
&= \Sigma_{i=0}^{d} c_i (\tau^i \cdot g_1) \\
&= c_0 + c_1 \cdot (\tau \cdot g_1) + c_2 \cdot (\tau^2 \cdot g_1) + c_3 \cdot (\tau^3 \cdot g_1) + \ldots + c_d \cdot (\tau^d \cdot g_1)
\end{aligned}
$$

↑となり、計算に必要となる τ は元の多項式から、
$\tau^i \cdot g_1$ は、セットアップで生成されたSRS から取得できるものであることがわかる

ここで、KZG のセキュリティは↓から成り立つと捉えられる
→ 楕円曲線上の離散対数問題（Elliptic Curve Discrete Logarithm Problem）
→ 指数を知っていることの仮説（Knowledge of Exponent Assumption）

3. 多項式の評価と評価証明の生成（open / evaluate）

🔺 **コミットメントスキーム（CS）!= 暗号化スキーム**

→ CS の目的 = コミット生成者が**コミット対象の値を実際に保有していること** の検証

→ 秘匿性を持たない CS も有用（i.e.., 値の暗号化も復号も行ってはいない）

✅ 多項式コミットメントスキーム（PCS）では↑を、コミット対象の多項式を**ランダムに選ばれたポイントで評価** することで効率的に検証する

→ コミット生成者は、多項式自体は送りたくない（i.e., 複雑性、秘匿性）

→ コミット生成者は、

1）多項式のコミットメントを生成した後に

2）多項式をランダムなポイントで評価（実行）し

3）**2の結果と1で生成したコミットメントの整合性が取れていること** を証明する

ことで、コミット対象の多項式（の知識）を保有していることを検証可能にする

3. 多項式の評価と評価証明の生成（open / evaluate）

A. 多項式の評価ポイントの選出

コミットメント検証者は、コミットメント生成者に対して
多項式をどのポイントで評価するべきかを**ランダムに決定**し送付する
→ 検証者の役割を**暗号学的ハッシュ関数（CHF）**で置き換えることで
非対話的な KZG を構築することができる（i.e., **フィアット・シャミア変換** ）

$$a \leftarrow \mathbb{F}_p$$

→ ここで、次数が d 以下の2つの異なる多項式を f と g とすると、f と g は、**最大でも d 個のポイントでのみ評価が一致** する（Schwartz-Zippelの補題）
→ $f(a) =$ であることの証明 = 高い確率 ( )で $\frac{d}{p}$ を保有している
→ $f(a) = b$ であることをf を明らかにしない形で**どのように証明するか** ？

3. 多項式の評価と評価証明の生成（open / evaluate）

B. 商多項式の計算

ここで、 $f(a)$ であることを↓の多項式に置き換えて考えると

$$f(X) - f(a)$$

↑は、a を根（root）に持つ、つまり X = a のときに 0 となることがわかり、
これは数式的には↓のように表現できる

$$f(X) - f(a) = (X - a) \cdot q(X)$$

→ q(X) は**商多項式（quotient polynomial）**と呼ばれ、次数 d-1 の多項式となる

$$q(X) = \frac{f(X) - f(a)}{X - a}$$

3. 多項式の評価と評価証明の生成（open / evaluate）

C. 評価証明（i.e., 商多項式に対するコミットメント）の生成

コミットメント生成者は、B で生成した多項式 q(X) の情報を
検証者に送付することで間接的に $f(a)$ であることが証明できる
→ 素の q(X) は送りたくない（i.e., 検証可能性、秘匿性）
→ q(X) に対する**別のコミットメント**を生成する

$$Com_q = q(\tau) \cdot g_1 = \Sigma_{i=0}^{d} c_i (\tau^i \cdot g_1)$$

→ Com_f と同様に、対象の多項式 q(X) と SRS から計算できる

最終的に、コミットメント生成者は検証者に対して、 $f(a)$ と Com_q を送付する
→ **どのように検証できるか？**

4. コミットメントの検証（verify）

🪄 ゴール: コミット生成者が**コミット対象の値を実際に保有していること** の検証

✅ 前提: 検証者は、$SRS \cdot a \cdot f(a)$　Com_q のみを保有

検証者が検証すべき等式は↓

$$f(\tau) =?(\tau - a) \cdot q(\tau) + f(a)$$

→ しかし検証者も　τを知らないため、このままの検証はできない
→ 両辺に生成元　g_1を掛けることで**楕円曲線群の要素の比較問題** へと変換する

$$f(\tau) \cdot g_1 =?\{(\tau - a) \cdot q(\tau) + f(a)\} \cdot g_1$$

$$Com_f =?(\tau - a) \cdot Com_q + f(a) \cdot g_1$$

4. コミットメントの検証（verify）

🪄 ゴール: コミット生成者が**コミット対象の値を実際に保有していること** の検証

✅ 前提: 検証者は、SRS・a・$f(a)$　　のみを保有　　Com_q

$$Com_f \stackrel{?}{=} (\tau - a) \cdot Com_q + f(a) \cdot g_1$$

↑も τを含んでいるため、直接計算はできない
→ **楕円曲線ペアリング** を用いて　　の要素の比較問題に再変換（G_1 p.18）

$$\begin{aligned} e(Com_f, g_2) &= e(Com_q, (\tau - a) \cdot g_2) \cdot e(f(a) \cdot g_1, g_2) \\ &= e(Com_q, \tau \cdot g_2 - a \cdot g_2) \cdot e(f(a) \cdot g_1, g_2) \end{aligned}$$

→ ↑の等式であれば、検証者は**計算に必要な値を全て保有している**
→ なぜ↑は正しいといえるのか？（i.e., 等式は成り立つのか）

4. コミットメントの検証（verify）

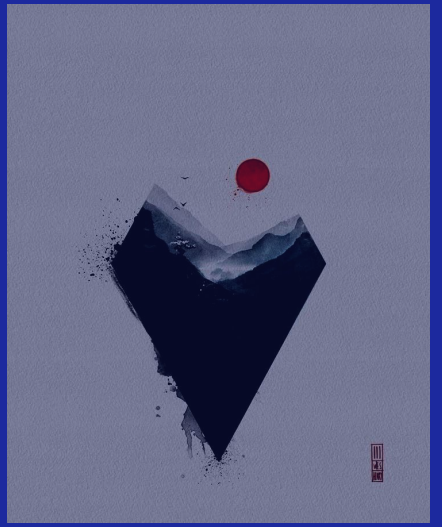
$$e(Com_f, g_2) = e(Com_q, \tau \cdot g_2 - a \cdot g_2) \cdot e(f(a) \cdot g_1, g_2)$$

❓↑はそもそもなにを検証していて、なぜ正しいといえるのか？
→ 右辺（RHS）から展開していくと、最終的に左辺（LHS）となることがわかる

$$
\begin{aligned}
RHS &= e(g_1, g_2)^{q(\tau)\cdot(\tau-a)} \cdot e(g_1, g_2)^{f(a)} && \longleftarrow \text{ペアリングの双線形性を利用} \\
&= e(g_1, g_2)^{q(\tau)\cdot(\tau-a)+f(a)} \\
&= e(g_1, g_2)^{\frac{f(\tau)-f(a)}{\tau-a}\cdot(\tau-a)+f(a)} && \longleftarrow f(\tau) =? (\tau - a) \cdot q(\tau) + f(a) \\
&= e(g_1, g_2)^{f(\tau)} && \text{を適用（p.26）} \\
&= e(f(\tau) \cdot g_1, g_2) && \longleftarrow \text{ペアリングの双線形性を利用} \\
&= LHS
\end{aligned}
$$

さまざまな証明スキーム：

PLONK



さまざまな証明スキーム

- ゼロ知識証明の生成 & 検証・実装方法はさまざま
  - 証明プロトコル : e.g.) PLONK, Groth16
  - 実装フレームワーク・ライブラリー
    - 回路（サーキット）実装 : e.g.) circom, noir, halo2
    - プロトコル実装 : e.g.) icicle-snark
    - 証明生成 & 検証実装 : e.g.) snarkJS, gnark, plonky2

- ZKエンジニアが実装すべきもの
  - サーキット実装
  - 証明生成 & 検証フレームワークの組み込み
    - どこで証明を生成するか : e.g.) フロントエンドの TypeScript 環境
    - どこに証明を格納するか : e.g.) IPFS (InterPlanetary File System)
    - どこで証明を検証するか : e.g.) スマートコントラクト

PLONK

- **PLONK** := **P**ermutations over **L**agrange-basis for **O**ecumenial **N**oninteractive arguments of **K**nowledge (paper: https://eprint.iacr.org/2019/953.pdf)
  - ↑のオリジナルモデル以外にも数多くの PLONK亜種が存在する

- PLONK の性質
  - **複数の多項式間の関係性** の証明
  - 多項式コミットメントを用いたゼロ知識性の担保
  - 利点
    - 汎用的かつ更新可能なセットアップ
    - 高い拡張性: e.g.) カスタムゲート、ルックアップテーブル、畳み込み
  - 欠点
    - 証明とキーのサイズ

PLONK: メンタルイメージ

🪄 ゴール（recap）: **ある主張が真となる知識を保有していることの証明**

→ 主張 ≈ 関数（i.e., 数式化）≈ 算術回路
→ 知識 ≈ 入力値 ≈ ウィットネス（witness）
→ ゼロ知識証明 ≈ 知識の秘匿化 ≈ 秘匿性のあるコミットメントスキームの利用

⚠️ コミットメントスキームで証明するもの　　ゼロ知識証明で証明するもの

→ CS　で証明するもの = （コミットされた）多項式を知っていること
→ ZKPで証明するもの = 多項式が表現（encode）している内容の一貫性
→ **CSを用いるとゼロ知識性は担保できるが、CS単体ではゼロ知識証明ではない**

PLONK: メンタルイメージ

1. PLONKにおける証明したい主張(+ 知識)の表現形態

   → **複数の一変量多項式(univariate polynomial)**として表現
   (PLONKの亜種は、多重線形多項式など別の多項式を用いることもある)
   → Q. なぜ多項式を用いるのか？多項式のなにが有用なのか？
   → A. 多項式は**大量のデータ(+ データ間の関連性)を単一の数学的構造** として表せる

2. PLONKで証明・検証するもの

   → 証明者が、自らがコミットした多項式を**実際に保有している**こと
   → 証明者が提示する多項式が、**主張とそれに対する入力値の組み合わせ** を
   正確に表現していること

PLONK: メンタルイメージ

1. 表現形態

どのようにして
主張(+知識)を
表現するか

2. 秘匿方法

どのようにして
秘匿性(ゼロ知識性)を
担保するか

3. 証明・検証方法

どのようにして
検証可能な証明を
生成するか

PLONK: メンタルイメージ

1. 表現形態

複数の一変量多項式

2. 秘匿方法

KZGコミットメント

3. 証明・検証方法

PIOP
（+ Fiat-Shamir Trans.）

PLONK: メンタルイメージ

1. 表現形態

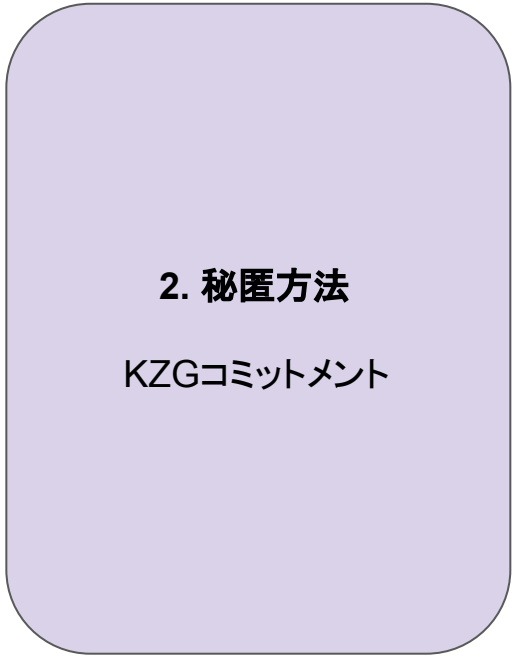
複数の一変量多項式

- 主張の構造を、+ と × を演算子としてもつ
  算術回路（arithmetic circuit）として表現
  → あらゆる計算は + と × の組み合わせで表現できる
  → fan-in = 2・fan-out = ∞

- **回路自体の構造**を多項式として表現
  → ゲート毎の計算の制約を表す多項式
  → ゲート間の繋げ方の制約を表す多項式
  → 回路へのインプットに関係なく事前に生成できる

- **回路へのインプット**を多項式として表現
  → 証明者がそれぞれ生成する多項式

PLONK: メンタルイメージ



- セットアップでSRSを生成する（p.20）
  → 固定された次数以上の多項式は扱えないという
  コンテキストで捉えると、**あらゆる証明に対して**
  汎用的なセットアップは存在しない

- 1で生成した**各多項式に対して**、
  KZGコミットメントを生成
  → 水面下では商多項式も生成する必要あり（ p.25）

- 証明の検証時には、**各多項式の評価証明の**
  検証も合わせて行う
  → この検証がないと知識の証明にならない

PLONK: メンタルイメージ

3. 証明・検証方法

PIOP
（+ Fiat-Shamir Trans.）

- 検証プロセスにおいて、各多項式は
  ランダムに選ばれたポイント で評価される
  → Fiat-Shamir 変換によって非対話的に変換できる

- 回路のインプット・アウトプットの整合性は、
  KZGの評価証明によって検証される
  → **ゼロテスト**（i.e., 多項式がゼロであるかのテスト）など
  他の手法でも検証自体は可能

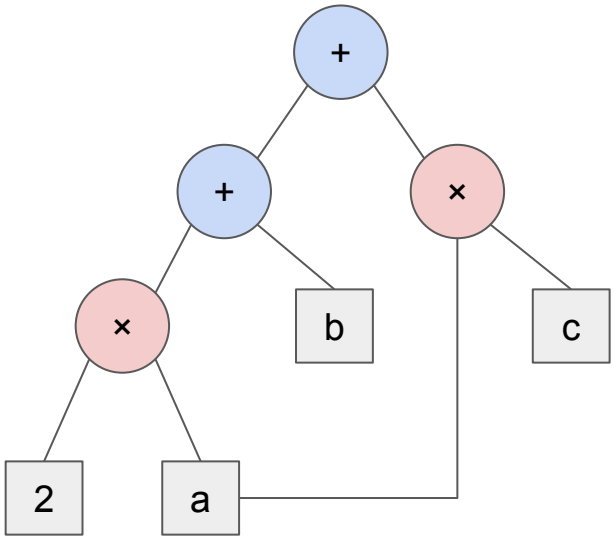
- 回路自体を表現した多項式の検証も必要
  → ゲート毎の制約（gate constraints）の整合性は、
  ゼロテストによって検証される
  → ゲート間の制約（copy constraints）の整合性は、
  多項式の積を用いた順列チェック によって検証される

PLONKの仕組み #1-A: 主張 => 関数 => 算術回路

- あるロジックがあり、それを抽象的に関数として表すとのようになるとする

$$f(a, b, c) = (2 \cdot a + b) + a \cdot c$$

→ これをPLONKが扱いやすい算術回路に
変換すると→のようになる



- 証明者は、この関数(i.e., 算術回路)を
ある秘密の入力値 a, b, c で実行すると、
ある秘密の出力値 y となるという主張を
PLONKベースのSNARKを用いて
証明したい
→ e.g.) $y = f(a, b, c) = f(1, 2, 3) = 4 + 3 = 7$

PLONKの仕組み #1-B: 算術回路 => ベクトル

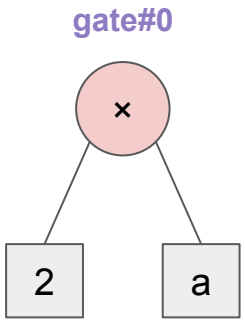
- 算術回路をベクトル(i.e., データの羅列)として捉えるために
  PLONKでは、ゲート毎の演算を↓の等式に変換する

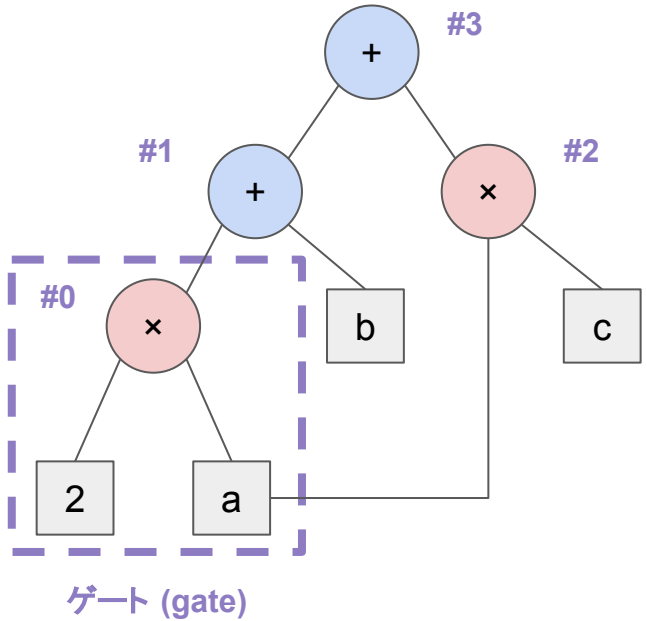
$$q_L \cdot l + q_R \cdot r + q_M \cdot l \cdot r + q_O \cdot o + q_C = 0$$

q_L: 演算における左(1つ目)の入力値の係数
q_R: 演算における右(2つ目)の入力値の係数
q_M: 演算子が掛け算の場合は 1、足し算の場合は 0
q_O: 演算の出力値の係数
q_C: 演算における定数

→ e.g.) ゲート#0

$$\begin{cases} q_L = q_R = q_C = 0 \\ q_M = 1 \\ q_O = -1 \end{cases}$$





PLONKの仕組み #1-B: 算術回路 => ベクトル

- p.41の算術回路における全てのゲートに対して、同様の等式に変換すると、となる

	q_L	q_R	q_M	q_O	q_C
gate#0	0	0	1	-1	0
gate#1	1	1	0	-1	0
gate#2	0	0	1	-1	0
gate#3	1	1	0	-1	0

→ 各列（e.g., q_L 紫に色付けされた列）をそれぞれベクトルとして捉えると、
元の算術回路は、↑の**5つのベクトルで表せる** ことがわかる

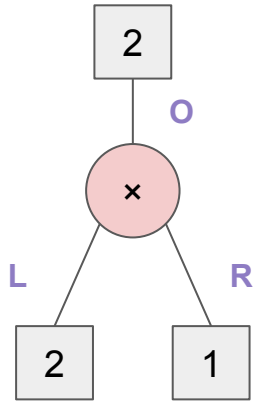
PLONKの仕組み #1-B: 算術回路 => ベクトル

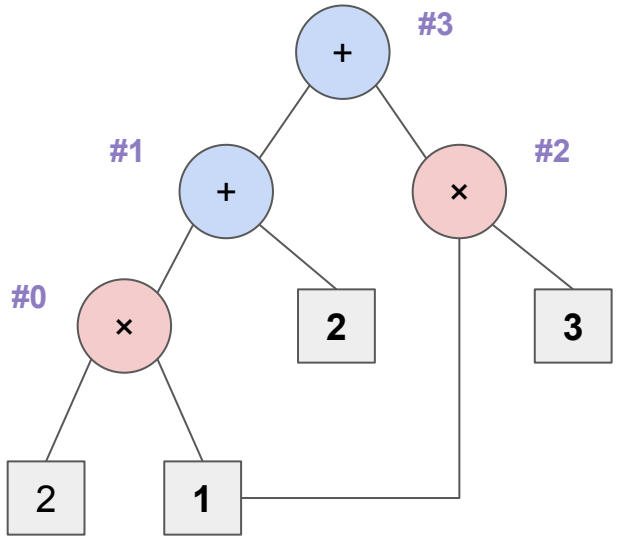
- 次に、p.41の算術回路に $(a, b, c) = (2, 1, 3)$ を代入した際の
  各ゲートにおける入力値・出力値 をベクトルとして表現する

L ゲートの左（1つ目）の入力値
R ゲートの右（2つ目）の入力値
O ゲートの出力値

→ e.g.) ゲート#0:

$$\begin{cases} L = 2 \\ R = 1 \\ O = 2 \end{cases}$$





PLONKの仕組み #1-B: 算術回路 => ベクトル

- p.43の算術回路における全てのゲートに対して、入出力値を計算すると↓となる

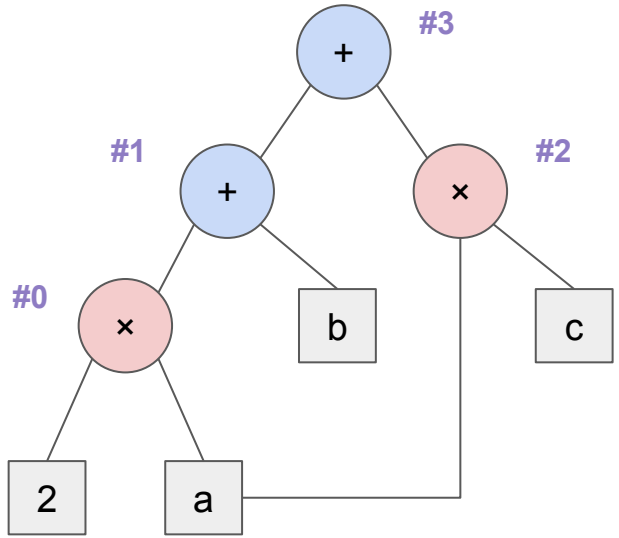
	L	R	O
gate#0	2	1	2
gate#1	2	2	4
gate#2	1	3	3
gate#3	4	3	7

→ 各列（e.g., L 紫に色付けされた列）をそれぞれベクトルとして捉えると、
算術回路の入出力値は、↑の**3つのベクトルで表せる** ことがわかる

PLONKの仕組み #1-B: 算術回路 => ベクトル

- p.42とp.44で、**ゲート毎の演算とその入出力値** はベクトルとして表現できたが、
  ゲート間の繋がり（wiring）は表現できていない
  → e.g.) ゲート#0の出力値は、ゲート#1の左の入力値
  → ゲート間の繋がりを表現するためには、↓のように
  すべてのゲートにおける入出力に対するグローバルな
  （i.e., ゲートを超えた）インデクシング が必要

	L	R	O
gate#0	2 (i=0)	1 (i=1)	2 (i=2)
gate#1	2 (i=3)	2 (i=4)	4 (i=5)
gate#2	1 (i=6)	3 (i=7)	3 (i=8)
gate#3	4 (i=9)	3 (i=10)	7 (i=11)



PLONKの仕組み #1-B: 算術回路 => ベクトル

- p.44のグローバルインデックスを用いて、どの値同士が繋がっているのかを
  インデックスのマッピングとして表すと↓となる

	σ_L	σ_R	σ_O
gate#0	0	6	3
gate#1	2	4	9
gate#2	1	7	10
gate#3	5	8	11

→ PLONKでは、**1の原始k乗根（primitive k-th root of unity）** $\{1, \omega, \omega^2, \omega^3, .を., \omega^{k-1}\}$
インデックスとして用いるため、このままではグローバルインデックスとして作用しない

PLONKの仕組み #1-B: 算術回路 => ベクトル

- （後ほど詳説するコピー制約の検証のため）グローバルインデックスは $\omega^i \cdot k_j$（i.e., ゲート#iのj番目のインプット）として表される

	σ_L	σ_R	σ_O
gate#0	$\omega^0 \cdot k_0$	$\omega^2 \cdot k_0$	$\omega^1 \cdot k_0$
gate#1	$\omega^0 \cdot k_2$	$\omega^1 \cdot k_1$	$\omega^3 \cdot k_0$
gate#2	$\omega^0 \cdot k_1$	$\omega^2 \cdot k_1$	$\omega^3 \cdot k_1$
gate#3	$\omega^1 \cdot k_2$	$\omega^2 \cdot k_2$	$\omega^3 \cdot k_2$

→ 各列（e.g., σ_R 紫に色付けされた列）をそれぞれベクトルとして捉えると、
算術回路のゲート間の繋がりは、↑の**3つのベクトルで表せる** ことがわかる

PLONKの仕組み #1-C: ベクトル => 多項式

- p.41, p.43, p.45の11個のベクトル（i.e., 配列）をそれぞれ多項式として表したい
  → ベクトルを多項式に変換する方法には主に↓の3種類がある

  1. As Coefficients: ベクトルの要素を多項式の **係数** として捉える

     $P(X) := \Sigma_{i=0}^{k} v_i X^i$

  2. As Evaluation Points: ベクトルの要素を多項式の **評価値** として捉える

     $\{(0, v_0), (1, v_1), (2, v_2), \ldots, (k, v_k)\}$

  3. As Polynomial Roots: ベクトルの要素を多項式の **根** として捉える

     $P(X) := \Sigma_{i=0}^{k} (X - v_i)$

- PLONKでは、まずは2のようにベクトルの要素を多項式の評価値として捉える
  → e.g.）ベクトル　を評価値として捉えた多項式　は↓の4点を通る

  $\{(0, 0), (1, 1), (2, 0), (3, 1)\}$

PLONKの仕組み #1-C: ベクトル => 多項式

- $\{(0,0),(1,1),(2,0)\}$の3点を通る多項式 $f(x)=x^2-2x$ は、**多項式補間（polynomial interpolation）**と呼ばれる手法によって一意に定めることができる
  → **異なる k 個の点を通る多項式は、最大次数が k-1 以下の多項式に常に一意に定まる** ことが知られている（代数学の基本定理 / Fundamental Theorem of Algebra）

- さまざまな多項式補間アルゴリズムがあるなかで、PLONKではほとんどの場合 **ラグランジュ補間** や**逆高速フーリエ変換**と呼ばれるアルゴリズムが用いられる
  → ラグランジュ補間 (Lagrange Interpolation)
  → 高速フーリエ変換（Fast Fourier Transform・FFT）

- PLONKでは、多項式を係数型（p.48のオプション1）と評価型（p.48のオプション2）の**両方で表現する必要**があるため、その変換にFFT/IFFTを用いる
  → **1の原始k乗根（primitive k-th root of unity）** $\{1,\omega,\omega^2,\omega^3,..,\omega^{k-1}\}$ を多項式の評価ドメインとすることで、FFT/IFFTの最適化を図る

PLONKの仕組み #1-C: ベクトル => 多項式

- 評価ドメインを1の原始k乗根とすると、p.42, p.44, p.47の11個のベクトルに対応する多項式はそれぞれ↓の条件を満たすような多項式となる

$$selector\ polynomials := \begin{cases} Q_L = \{(1,0),(\omega,1),(\omega^2,0),(\omega^3,1)\}\text{の4点を通る多項式} \\ Q_R = \{(1,0),(\omega,1),(\omega^2,0),(\omega^3,1)\}\text{の4点を通る多項式} \\ Q_M = \{(1,1),(\omega,0),(\omega^2,1),(\omega^3,0)\}\text{の4点を通る多項式} \\ Q_O = \{(1,-1),(\omega,-1),(\omega^2,-1),(\omega^3,-1)\}\text{の4点を通る多項式} \\ Q_C = \{(1,0),(\omega,0),(\omega^2,0),(\omega^3,0)\}\text{の4点を通る多項式} \end{cases}$$

← ゲート制約を表す多項式

$$witness\ polynomials := \begin{cases} L = \{(1,2),(\omega,2),(\omega^2,1),(\omega^3,4)\}\text{の4点を通る多項式} \\ R = \{(1,1),(\omega,2),(\omega^2,3),(\omega^3,3)\}\text{の4点を通る多項式} \\ O = \{(1,2),(\omega,4),(\omega^2,3),(\omega^3,7)\}\text{の4点を通る多項式} \end{cases}$$

← ゲート入出力を表す多項式

$$wiring\ polynomials := \begin{cases} \sigma_L = \{(1,\omega^0\cdot k_0),(\omega,\omega^0\cdot k_2),(\omega^2,\omega^0\cdot k_1),(\omega^3,\omega^1\cdot k_2)\}\text{の4点を通る多項式} \\ \sigma_R = \{(1,\omega^2\cdot k_0),(\omega,\omega^1\cdot k_1),(\omega^2,\omega^2\cdot k_1),(\omega^3,\omega^2\cdot k_2)\}\text{の4点を通る多項式} \\ \sigma_O = \{(1,\omega^1\cdot k_0),(\omega,\omega^3\cdot k_0),(\omega^2,\omega^3\cdot k_1),(\omega^3,\omega^3\cdot k_2)\}\text{の4点を通る多項式} \end{cases}$$

← ゲート間の繋がりを表す多項式

PLONKの仕組み #2: 多項式コミットメント（KZG）

- PLONKでは、p.48にある**11個の多項式**に対してKZGコミットメントを適用する

  1. ゲート毎の演算を表す5個のセレクター多項式
  2. ゲート間の繋がりを表す3個のワイヤリング多項式
  3. 各ゲートの入出力の値を表す3個のウィットネス多項式

  → ↑以外の証明やその検証に必要となる副次的な多項式（ e.g., 商多項式）に対しても適宜KZGコミットメントを適用

- PLONKのセットアップ ≈ KZGのセットアップ
  → ⚠️ 対応可能な多項式の最大次数 d は、証明するサーキットサイズ（i.e., サーキットの保有するゲート数） $|C|$ よりも大きい必要がある（p.18）
  → ⚠️ KZGコミットメントを生成するためには、**係数表現された多項式が必要** であるため、FFT/IFFTで適宜多項式の変換を行う必要がある（p.21）

PLONKの仕組み #3-A: インプット制約の証明と検証

- 算術回路のインプット制約（i.e., パブリックインプット・アウトプット）の一貫性は、**KZGコミットメントスキームの評価証明**（p.26）を用いて証明することができる

ナイーブな実装：複数のKZG評価証明の送付 & 検証
1. <証明者> パブリックなインプット・アウトプットのインデックスで
   対象のウィットネス多項式を評価（ e.g., $L(\omega^0) = 2$, $O(\omega^8) = 3$ ）
2. <証明者> それぞれの評価証明を生成して検証者に送付（ p.24）
3. <検証者> それぞれの評価値が「**パブリックインプットであること + その評価表明**」を検証

→ 🤔 パブリックインプット・アウトプットの数だけ評価証明（とペアリング）の計算が必要
→ ❗ KZGのバッチ証明（batch proofs）の考え方を適用すると、
たった1つのKZG評価証明の検証 で済む

PLONKの仕組み #3-B: ゲート制約の証明と検証

- Recap: PLONKにおけるゲート制約の表現（p.42）

$$q_L \cdot l + q_R \cdot r + q_M \cdot l \cdot r + q_O \cdot o + q_C = 0$$

→ セレクター多項式とウィットネス多項式を用いて全てのゲート制約を表現すると ...

$$Q_L(X) \cdot L(X) + Q_R(X) \cdot R(X) + Q_M(X) \cdot L(X) \cdot R(X) + Q_O(X) \cdot O(X) + Q_C(X) = 0$$

- 0 多項式の**ゼロテスト（zero testing）**: 多項式 f(X) がゼロであるかどうかの検証

→ f(X) = 0 であれば、多項式を **根の形式** で表現することができる（p.48）

→ $Z(X) := \Pi(X - a)$ とすると　　$f(X) = 0 \Leftrightarrow f(X) = q(X) \cdot Z(X)$ (p.25)

→ PLONKの評価ドメインは、 $\Omega = \{1, \omega, \omega^2, \omega^3, \ldots, \omega^{n-1}\}$ なので

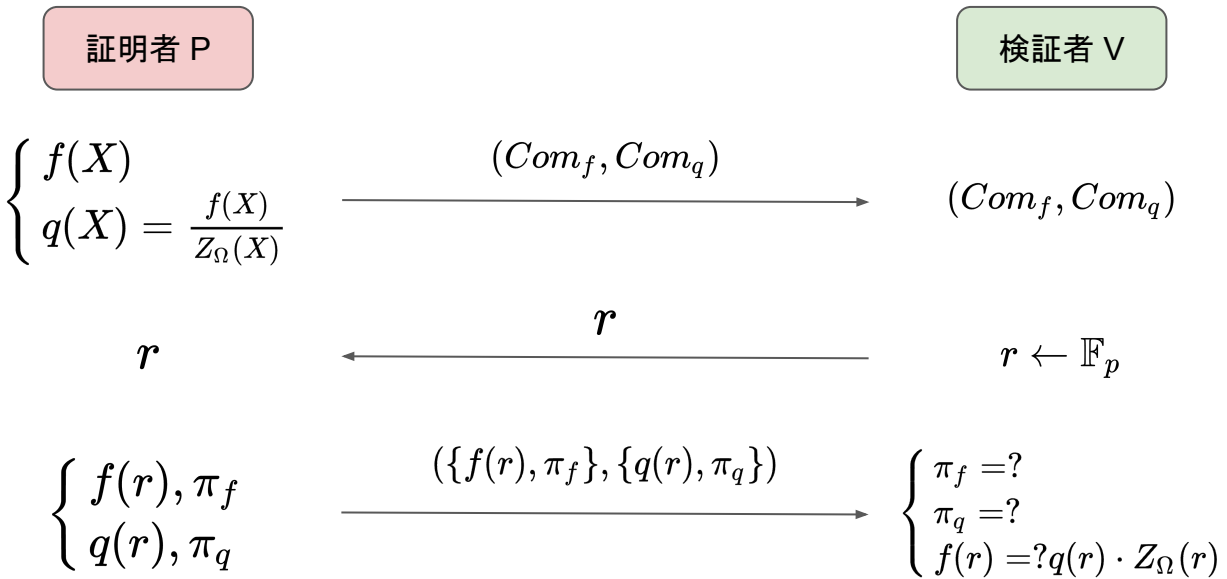
$Z_\Omega(X) := \Pi_{a \in \Omega}(X - a)$ というシンプルな計算となる

↑

消滅多項式（vanishing polynomial）と呼ばれる

PLONKの仕組み #3-B: ゲート制約の証明と検証

- KZGコミットメントスキームと組み合わせると、ゲート制約は↓の流れで検証される
  → $f(X) = Q_L(X) \cdot L(X) + Q_R(X) \cdot R(X) + Q_M(X) \cdot L(X) \cdot R(X) + Q_O(X) \cdot O(X) + Q_C(X)$ として考える



PLONKの仕組み #3-C: コピー制約の証明と検証

- Recap: ワイヤリング多項式 = グローバルインデックスのマッピング（p.47）

	σ_L	σ_R	σ_O
gate#0	$\omega^0 \cdot k_0$	$\omega^2 \cdot k_0$	$\omega^1 \cdot k_0$
gate#1	$\omega^0 \cdot k_2$	$\omega^1 \cdot k_1$	$\omega^3 \cdot k_0$
gate#2	$\omega^0 \cdot k_1$	$\omega^2 \cdot k_1$	$\omega^3 \cdot k_1$
gate#3	$\omega^1 \cdot k_2$	$\omega^2 \cdot k_2$	$\omega^3 \cdot k_2$

→ 🤔 証明者によって生成されたワイヤリング多項式の正しさをどのように証明・検証するか？
→ ❗ ウィットネス多項式の元々の（素の）インデックスにおける評価値と、
ワイヤリング多項式によってマッピングされたインデックスにおける**評価値が順列（並び替え）であること**を示せば良い

PLONKの仕組み #3-C: コピー制約の証明と検証

- ✨ ゴール: マッピング前後のウィットネス多項式の評価値が順列であることの証明

	$L(\omega^i)$	$R(\omega^i)$	$O(\omega^i)$
gate#0	2 (i=0)	1 (i=1)	2 (i=2)
gate#1	2 (i=3)	2 (i=4)	4 (i=5)
gate#2	1 (i=6)	3 (i=7)	3 (i=8)
gate#3	4 (i=9)	3 (i=10)	7 (i=11)

$L(\omega^{\sigma_L(\omega^i)})$	$R(\omega^{\sigma_R(\omega^i)})$	$O(\omega^{\sigma_O(\omega^i)})$
??	??	??
??	??	??
??	??	??
??	??	??

→ ❓ワイヤリング多項式の評価値は**グローバルインデックス** である一方で、
ウィットネス多項式の評価ドメインはゲート毎の**ローカルインデックス** であることを考慮する

PLONKの仕組み #3-C: コピー制約の証明と検証

- ✨ ゴール: マッピング前後のウィットネス多項式の評価値が順列であることの証明

  - ワイヤリング多項式 = **スワップしたインデックス同士の評価値は同値**
    → それぞれの因数分解は同じになる（i.e., 全体の積が同値になる）
    → インデックスのミスマッチにだけ留意する

  - ❓証明アイデア: **並び替え前後の多項式の差分がゼロになること** をゼロテストする

    $$f(X) - g(\sigma(y)) = 0$$

    → 考え方・ロジック自体は問題ない
    → 並び替え後のウィットネス多項式の評価の時間計算が $O(n^2)$ となり、
    SNARKではなくなってしまうため NG

PLONKの仕組み #3-C: コピー制約の証明と検証

- ✨ ゴール: マッピング前後のウィットネス多項式の評価値が順列であることの証明
  - 証明方法: 並び替え前後の多項式の **大積（grand product）**の比率が1:1になることを証明する

$$
\begin{aligned}
\frac{f(X)}{g(X)} &= \Pi_{i=0}^{k-1} \Pi_{W \in L,R,O} \frac{(W(\omega^i) + \beta \cdot \omega^i + \gamma)}{(W(\sigma_W(\omega^i)) + \beta \cdot \sigma_W(\omega^i) + \gamma)} \\
&= \Pi_{i=0}^{k-1} \frac{(L(\omega^i) + \beta \cdot \omega^i + \gamma) \cdot (R(\omega^i) + \beta \cdot (\omega^i \cdot k_1) + \gamma) \cdot (O(\omega^i) + \beta \cdot (\omega^i \cdot k_2) + \gamma)}{(L(\sigma_L(\omega^i)) + \beta \cdot \sigma_L(\omega^i) + \gamma) \cdot (R(\sigma_R(\omega^i)) + \beta \cdot \sigma_R(\omega^i) + \gamma) \cdot (O(\sigma_O(\omega^i)) + \beta \cdot \sigma_O(\omega^i) + \gamma)} \\
&= 1
\end{aligned}
$$

→ 証明が「たまたま」通ってしまうことを防ぐため（ i.e., インデックスと評価値の束縛とインデックスの衝突を防ぐため）に 2つの体の要素 β と γ が検証者によって選出され、各項に適用される

→ **どのようにして ↑の等式を証明するか？**

PLONKの仕組み #3-C: コピー制約の証明と検証

$$
\begin{aligned}
\frac{f(X)}{g(X)} &= \Pi_{i=0}^{k-1} \Pi_{W \in L,R,O} \frac{(W(\omega^i) + \beta \cdot \omega^i + \gamma)}{(W(\sigma_W(\omega^i)) + \beta \cdot \sigma_W(\omega^i) + \gamma)} \\
&= \Pi_{i=0}^{k-1} \frac{(L(\omega^i) + \beta \cdot \omega^i + \gamma) \cdot (R(\omega^i) + \beta \cdot (\omega^i \cdot k_1) + \gamma) \cdot (O(\omega^i) + \beta \cdot (\omega^i \cdot k_2) + \gamma)}{(L(\sigma_L(\omega^i)) + \beta \cdot \sigma_L(\omega^i) + \gamma) \cdot (R(\sigma_R(\omega^i)) + \beta \cdot \sigma_R(\omega^i) + \gamma) \cdot (O(\sigma_O(\omega^i)) + \beta \cdot \sigma_O(\omega^i) + \gamma)} \\
&= 1
\end{aligned}
$$

- 単純に全体の積をチェックするだけでは不十分（ i.e., 違う掛け合わせが同じ積になることもある）
  → ↑を再帰的な制約に変換するような多項式 p(X) をみつけ、p を使ってゼロテストを行う
  → p は累積的に $\frac{f(X)}{g(X)}$を計算するような多項式（**積算器 / accumulator** とも呼ばれる）

$$
p(X) = \begin{cases} p(\omega^0) = 1 \\ p(\omega^{i+1}) = p(\omega^i) \cdot \frac{f(X_i)}{g(X_i)} \quad \forall i \in \{0, 1, 2, \ldots, k-1\} \end{cases}
$$

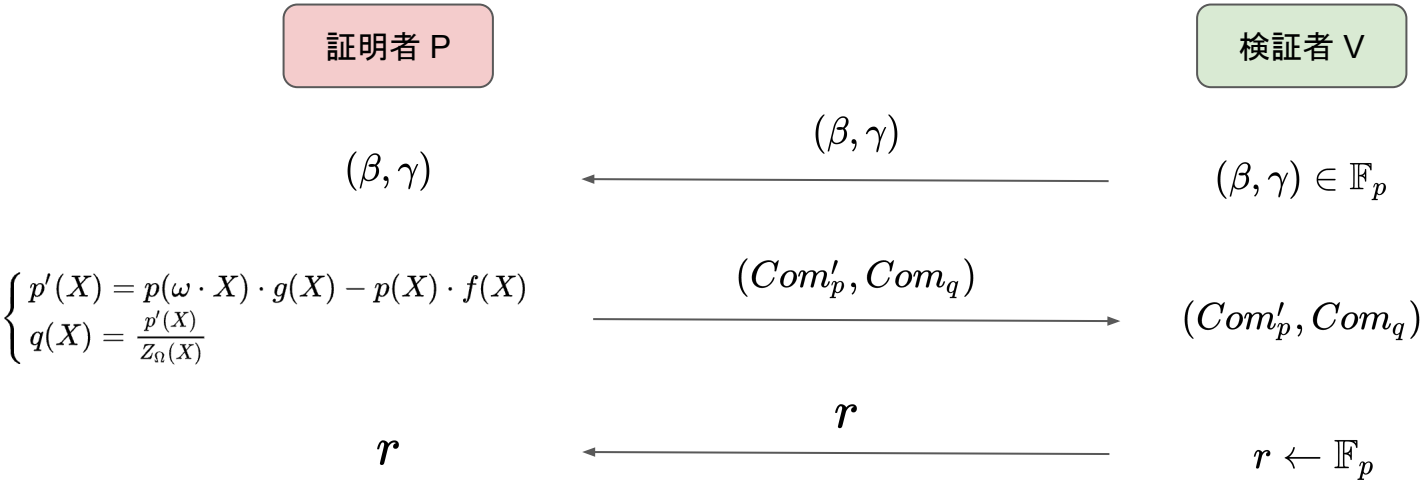
  → ここで、ゼロテストを行う多項式 p' は↓となる

$$
p'(X) = p(\omega \cdot X) \cdot g(X) - p(X) \cdot f(X) = 0
$$

PLONKの仕組み #3-C: コピー制約の証明と検証

- KZGコミットメントスキームと組み合わせると、コピー制約は↓の流れで検証される

→ $\frac{f(X)}{g(X)} = \Pi_{i=0}^{k-1} \Pi_{W \in L,R,O} \frac{(W(\omega^i)+\beta \cdot \omega^i+\gamma)}{(W(\sigma_W(\omega^i))+\beta \cdot \sigma_W(\omega^i)+\gamma)}$ として考える



PLONKの仕組み #3-C: コピー制約の証明と検証

- KZGコミットメントスキームと組み合わせると、コピー制約は↓の流れで検証される

→ $\frac{f(X)}{g(X)} = \Pi_{i=0}^{k-1} \Pi_{W \in L,R,O} \frac{(W(\omega^i)+\beta\cdot\omega^i+\gamma)}{(W(\sigma_W(\omega^i))+\beta\cdot\sigma_W(\omega^i)+\gamma)}$ として考える

証明者 P

検証者 V

$$\begin{cases} f(r), \pi_f \\ g(r), \pi_g \\ q(r), \pi_q \\ p'(r), \pi_{p_1} \\ p'(r\cdot\omega), \pi_{p_2} \end{cases} \xrightarrow{(f(r), \pi_f, g(r), \pi_g, q(r), \pi_q, p'(r), \pi_{p_1}, p'(r\cdot\omega), \pi_{p_2})} \begin{cases} \pi_f =? \\ \pi_g =? \\ \pi_q =? \\ \pi_{p_1} =? \\ \pi_{p_2} =? \\ p'(r\cdot\omega)\cdot g(r) =? \; p'(r)\cdot f(r) \\ p'(r) =? \; q(r)\cdot Z_\Omega(r) \end{cases}$$

PLONKの仕組み: まとめ

- PLONKは一変量多項式を中心としたゼロ知識証明スキーム
  - SNARKスキームではあるが、インタラクティブな形で実装することも可能

- 主張 + 知識を算術回路 → ベクトルに変換したのち、最終的に複数の多項式として表現

- 多項式はベクトルから1の原始k乗根をドメインとした評価値型として補間されるが、
  FFT/IFFTで効率的に係数型にも変換できるため、KZGコミットメントが適用できる

- 証明者は、回路の構造（ゲート毎＆ ゲート間）と回路へのインプットに対して、
  複数の多項式の関係性を構築し、それを証明することで元の主張の証明を行う

- 検証者は、ランダムな評価ポイントにおける多項式の関係性とKZGの評価証明を
  検証することで、効率的に証明を検証することができる

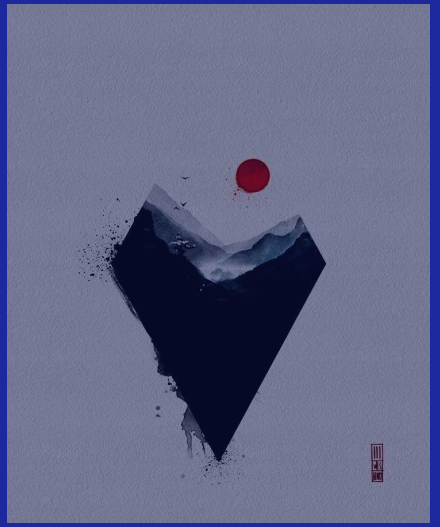
PLONK: その他

- PLONKは非常に柔軟な設計が可能なプロトコルであるため、実装もさまざま
  - このスライドではオリジナル(2019)に近い根本的な考え方・仕組みを紹介
  - 理論・実装の両面からの最適化手法も数多くあり、より実用的な実装ではそれらを取り入れている場合も多い(e.g., カスタムゲート)

- リンク等
  - より理解を深めたい場合はp.66 の参考資料がオススメ
  - PLONKに対応しているプロトコル・ライブラリ一覧

ZK Tokyo

演習問題



演習問題

1) SNARK とは … ARgument of Knowledge（p.7）を指しますが、
なぜ「証明」であるのに“argument” と表現しているのでしょうか？
”proof” と “argument” の違いを調べて説明してみましょう。
2) p.21のKZGにおいて、コミットメントに**秘匿性（i.e., ゼロ知識性）**を与えるには
どうすれば良いでしょうか？具体的な構築方法を調べて説明してみましょう。
3) **f(a, b) = 5 * (a * b - a) + 2 * b** を PLONK が適用できる形（i.e., 複数の一変量多項式）に
途中経過をわかりやすく明示しながら変換してみましょう。
4) p.52におけるパブリックインプット・アウトプットの証明を
たった1つのKZG評価証明で行うにはどうすれば良いか、調べて説明してみましょう。
→ ヒント: ゼロテスト（p.53~p.54）を用いる。

参考資料

- Commitment Schemes
  - ZKDocs by Trail of Bits
  - KZG Polynomial Commitments by Dankrad Feist
- PLONK
  - PlonK (the original paper)
  - Plonkbook: A Handbook for Poly-IOP Gadgets
  - The Plonk SNARK by Dan Boneh
  - Understanding PLONK by Vitalik Buterin
  - All you wanted to know about Plonk by LambdaClass

You nailed it! 🔥