シャッフル証明と Bulletproofs R1CS

手法・脆弱性・実装理由

1 シャッフル証明のやり方

役割	操作フロー	使う情報 / 生成物
証明者	1. コミットし $P(X) = \prod_i (x_i - X)$ 構成 2. コミット列をハッシュし乱数 z を取得 3. 乗算ゲート鎖で $P(z)$・$Q(z)$ を評価し「差＝0」制約を作成	入力リスト $\{x_i\}$、Pedersen基底、ハッシュ関数
検証者	1. コミットを読み取り 2. 同じハッシュで **同一の** z を再計算 3. 同じ制約を再構成し証明を検証	公開コミット、ハッシュ関数

> **キー**: 乱数 z はコミット確定後に決まり、Prover は後出しで値を操作できない

具体例（体 $\mathbb{F}_7$、長さ $k = 2$）

1. 入力 $\{2, 5\}$，出力 $\{5, 2\}$
2. 多項式
   $P(X) = (2 - X)(5 - X)$,
   $Q(X) = (5 - X)(2 - X)$
3. ハッシュ→乱数 $z = 3$
4. 評価
   $P(3) = (-1) \cdot 2 \equiv 5$,
   $Q(3) = 2 \cdot (-1) \equiv 5$ → 等しい
   もし出力を $\{5, 6\}$ に改変すると
   $Q'(3) = 2 \cdot (-1) \cdot 3 \equiv 1 \neq 5$ で失敗

誤判定確率 $\leq k/q = 2/7$ （Schwartz–Zippelの補題）

2️⃣ 総和・総積テストの脆弱性と理由

衝突（すり抜け）例

テスト	集合 A	集合 B	両者の値
総和	{1, 4, 5}	{2, 3, 5}	10
総積	{2, 2, 3}	{1, 1, 12}	12

- 総和 $e_1 = \sum x_i$、総積 $e_k = \prod x_i$ は **係数の一部** に過ぎず、攻撃者は同じ値になる異集合を容易に作れる
- よって最悪ケースの誤判定確率＝100 %（攻撃者が仕様を知っていれば必ず通過）

比較：誤判定確率

方法	最悪誤判定確率	攻撃者の後出し改変
総和のみ	1	可能
総積のみ	1	可能
乱数 z 評価	k/q	不可（コミット後に z 決定）

3 なぜ掛け算チェーン？

— R1CS との親和性

R1CS　任意の算術回路を「乗算ゲート＋線形制約」のみで表現する汎用フォーマットについて

1. **R1CS 基本形**
   $(\mathbf{a}\cdot\mathbf{s})\times(\mathbf{b}\cdot\mathbf{s})=(\mathbf{c}\cdot\mathbf{s})$
   → “乗算ゲート + 線形制約” のみで表現
2. **多項式評価と一致**
   連鎖 $(x_1-z)\to(x_2-z)\to\ldots$ は **各段が 1 乗算ゲート**追加変換不要でそのまま制約化できる

補足系

Rank-1 Constraint System（R1CS）

任意の算術回路を「乗算ゲート＋線形制約」のみで表現する汎用フォーマットについて

背景

Bulletproofsを使いたい

→R1CSを使う※

→シャッフル証明をするには掛け算がいい

→ $P(X) = \prod_i (x_i - X)$ でのシャッフル証明をする

回路ごとに「信頼できる複数当事者による鍵生成（Trusted Setup）」が必須だったのがいらない
乗算ゲート列を内積引数により一括処理できる