zk-SNARK について
～ ゼロ知識で, 簡潔で, 非対話による 変数に関する知識証明 ～

清家大嗣

2020 年 5 月 20 日

1 zk-SNARK について

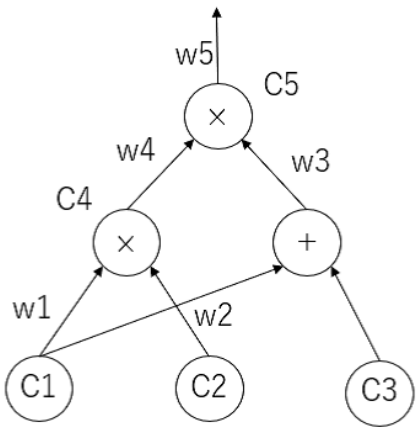

図 1: QAP (Quadratic Arithmetic Program) の一例

zk-SNARK [1, 2, 3, 4, 5, 6] を用いると, どのようなことが可能になるだろうか. zk-SNARK は zero knowledge-Succinct Non-interactive Argument of Knowledge の略であり, ゼロ知識で, 簡潔に非対話的に実変数に関する知識を持つことを証明可能にする. 最初に, 実変数 (Argument) について理解する. そのために, 図 1 にある二次算術プログラム (QAP: Quadratic Arithmetic Program) を説明する. QAP における, 変数は入力, 中間出力, 最終出力である. 図 1 では, 入力 $c1, c2, c3$ を用いて積や和などの二項演算子による計算 (途中の積により得られる中間出力を $c4$ とする) を経て最終出力 $c5$ が計算され, $c1, c2, c3, c4, c5$ が変数となる. zk-SNARK を用いると, これら変数情報を一切明らかにすることなく, 自身が何らかの入力 $c1, c2, c3$, 対応した中間出力 $c4$, 最終出力 $c5$ に関する知識を証明できる (この際, $c1, c2, c3, c4, c5$ の情報を部分的に公開, 検証者が計算することも可能 (後述)).

また, zk-SNARK の Succinct (簡潔) 特性により, 入力に対して得られた計算が正しいことをその計算をせずに証明できる. この性質から, 「zk-SNARK による証明時間 < その計算を実行する

のにかかる時間」 ケースや 「zk-SNARK の証明に必要なデータ量 (論文 [2] の結論では, QAP のサイズに関わらず, 288 Byte)」 < 「検証に必要な入出力データを集める時間」 が成立する場合に, ブロックチェーンにとって有効な手法になりうると考えられる. ブロックチェーンの合意形成時の計算検証にかかるコストが下がり, ブロックチェーンのスケーラビリティを改善できる (この手法はクラウドリソースを用いた計算結果の信頼性を担保するためにも用いられる) からである. また, 出力変数を固定して, その出力の因数分解を知っていることや, ハッシュの原像を明らかにせず, その知識を持つことのみの証明も可能である (自身がお金を利用する際に必要なシークレットを持つことを, シークレットを公開せずに証明できる).

1.1 zk-SNARK をどのように実現するか

zk-SNARK は図 1 のような QAP を変数 x の多項式表現に変形する. まず, 積をベースとした演算を複数回繰り返した結果, 出力 $c5$ が得られると考える. つまり, 図 1 の QAP は次の 2 式の等号が正しいと検証されれば, 正しく実行されたことになる.

$$c_4 = c_1 * c_2 \tag{1}$$

$$c_5 = c_4 * (c_1 + c_3) \tag{2}$$

これら式の総数を d とおく (この例の場合 $d = 2$ である). そして, 次のような行列 $L_{\mathrm{mat}}, R_{\mathrm{mat}}, O_{\mathrm{mat}}$ を考える.

$$L_{\mathrm{mat}} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad R_{\mathrm{mat}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad O_{\mathrm{mat}} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{3}$$

各列は式番号 1, 2 (d 個ある場合, $1, 2, ..., d$) と対応しており, 各行は変数 $c1, c2, ..., c5$ と対応している. L の各列は, 該当式番号の右辺の二項演算子 $*$ の左側にどれだけ変数 $c1, c2, ..., c5$ が含まれているかを表している (変数が含まれている場合, その行にフラグ 1 を立てている). R の各列は, 該当式番号の右辺の二項演算子 $*$ の右側にどれだけ変数 $c1, c2, ..., c5$ が含まれているかを表す. O の各列は, 該当式番号の左辺にどれだけ変数 $c1, c2, ..., c5$ が含まれているかを表す.

(3) の行列 $L_{\mathrm{mat}}, R_{\mathrm{mat}}, O_{\mathrm{mat}}$ から各式について検証可能な多項式 $P(x)$ を生成する. $P(x)$ は $x = 1$ の時, 式番号 (1) を検証し, $x = 2$ の場合は式番号 (2) を検証できるように設計する. (3) 式の各行列を行方向に見て, 例えば行列 O の 4 行目に着目すると (1) 式において c_4 は左辺に出現し, (2) 式では出現しない. つまり, 多項式 $P(x)$ において, $x = 1$ の時に c_4 が出現し, $x = 2$ の時は c_4 が消滅するように c_4 の係数として $o_4(x)$ を定義する[1]. (3) 式に従う QAP の場合, 行成分が (1, 0) の場合 $o_i(x) = 2 - x$, (0, 1) の場合は $o_i(x) = x - 1$ とできる. また, どの式にも出現しない (行成分が (0, 0) となっている) 場合, $o_i(x) = (2 - x)(x - 1)$ であり, どの式にも出現する (行成分が (1, 1) となっている) 場合 $o_i(x) = (2 - x) + (x - 1) = 1$ である. 他の行列 L, R の各行に対しても同様に考えて $l_i(x), r_i(x)$ を計算すると $P(x)$ は (4) 式のように表現できる.

$$P(x) = \left\{\sum_{i=1}^{5} c_i \cdot l_i(x)\right\}\left\{\sum_{i=1}^{5} c_i \cdot r_i(x)\right\} - \sum_{i=1}^{5} c_i \cdot o_i(x) \tag{4}$$

計算すると分かるが, $P(1) = 0$ の場合は (1) 式が得られ, $P(2) = 0$ の場合は (2) 式が得られる.

[1] このような $o_4(x)$ はラグランジュ補完 (補足 A) により定義できる. 当然素数 p を法とする有限体の世界においても, 拡張ユークリッドの互除法を用いれば ラグランジュ補完により得られた $o_4(x)$ の分母を 1 にすることは可能である.

1.2 (4) 式を用いた知識の証明

(4) 式は式番号に該当する x を代入する (例えば $x=1$ や $x=2$) と 0 になる. つまり, 多項式 $T(x)=(x-1)(x-2)$ で割り切れる (4) 式で表現 (つまり, $l_i(x), r_i(x), o_i(x)$ の一次結合で計算) される $P(x)$ を知っていることと, (1, 2) 式を満たす $c_1, c_2, ..., c_5$ を知っていることは同値である (これは, $P(x)$ が $T(x)$ で割り切れるので $T(1)=P(1)=T(2)=P(2)=0$ から, (4) 式が (1, 2) 式と同値となるためである). 従って, (5) 式となる $H(x)$ と $c1, c2, ..., c5$ を提示できれば, 自分が図 1 の QAP を正しく再現する $c1, c2, ..., c5$ を知っていたことの証明になる.

$$\begin{aligned} P(x)=T(x)H(x) &= \left\{\sum_{i=1}^{5} c_i \cdot l_i(x)\right\}\left\{\sum_{i=1}^{5} c_i \cdot r_i(x)\right\} - \sum_{i=1}^{5} c_i \cdot o_i(x) \\ &\equiv L(x) \cdot R(x) - O(x) \end{aligned} \tag{5}$$

以下, この提示を実行するエンティティを証明者 (Prover) と呼び, (5) 式が正しく成立していることを確認するエンティティを検証者 (Verifier) と呼ぶ.

1.3 節 1.2 の知識の証明を簡潔 (Succinct) にする

証明者から検証者に与えられた $H(x)$ と $c1, c2, ..., c5$ から, (5) 式の成立を確認するにはどうすればよいか. (5) 式の両辺の x に関する係数を比較するのはコストが高い (d 次の多項式の積の展開には $O(d^2)$ の積演算が必要).

この計算量を減らすために, 最高次数が $2d$ (これは (5) 式の右辺の最大次数の上限である) の異なる関数 $f(x), g(x)$ が素数 p を法とした有限体 F_p 上で高々$2d$ 個の点でしか交わらないことを用いる. 言い換えれば, $(x', f(x')) = (x', g(x'))$ となる x' は高々 $2d$ 個しかない (例えば, 二次方程式の解は高々 2 個) ことを利用する. つまり, 適当な値 s と適当な関数 $H'(x)$ を (5) 式に代入して右辺, 左辺の値が偶然一致する確率は $2d/p$ で抑えられ, p が十分に大きければその確率は殆ど無視できる.

従って, 検証者が証明者に秘密の値 s を送信して, 証明者は次式を満たす $L(s)$, $R(s)$, $O(s)$, $H(s)$ を検証者に返すことで (5) 式を満たす $H(x)$ を知っていると示せるようにする.

$$\begin{aligned} P(s)=T(s)H(s) &= \left\{\sum_{i=1}^{5} c_i \cdot l_i(s)\right\}\left\{\sum_{i=1}^{5} c_i \cdot r_i(s)\right\} - \sum_{i=1}^{5} c_i \cdot o_i(s) \\ &\equiv L(s) \cdot R(s) - O(s) \end{aligned} \tag{6}$$

すなわち, (6) 式が成立することを検証者が確認すればよい. 証明者が s から計算した値 $L=L(s), R=R(s), O=O(s), H=H(s)$ を検証者は受け取って, $LR-O=T(s)H$ を確認すればよいという事になる. しかし, $T(s)$ は s の値を知る証明者も容易に計算可能なため, $LR-O=T(s)H$ となるような L, R, O, H を適当に作成される可能性がある. 従って, 検証者は s を直接証明者に送信せずに, $H(x)$ と $c1, c2, ..., c5$ から $L(s), R(s), O(s), H(s)$ を計算できるように暗号化した上で送信する必要がある.

そこで, 検証者が生成した値 s を秘密とするため, 準同型暗号化 (Homomorphic Encryption) を施し, 証明者に送信する. 例えば, 加法に関する準同型暗号を可能とする楕円曲線暗号を用いる. 暗号化した値 $E(s)$ を生成元 (generator) G (or g) を用いて次式のように定義できる[2].

$$E(s) = sG \pmod{p} \equiv g^s \tag{7}$$

[2] 一般的な暗号の論文では, 楕円曲線上の点の加法は乗法表記される.

上式に加えて, 1 回の乗法を可能とするペアリング (paring) について説明する. これは, (7) 式による暗号化だけでは暗号化した値同士の積の計算ができないためである[3]. ペアリングは 2 つに分類され, 対称ペアリングと非対称ペアリングが存在する. 非対称ペアリングでは, 楕円曲線上の互いに異なる巡回群に含まれる 2 点 g_1, g_2 を用いて, 楕円曲線上の 2 点から有限体 F_p への写像となるペアリング関数 e を定義できる ([4] の P. 85, 非対称ペアリング).

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab} \equiv \mathrm{g}^{ab} \tag{8}$$

(8) 式の ab は秘匿されていることに注意する [4]. (8) 式は, (9) 式で表されるペアリングの双線形 (bilinear) と呼ばれる性質 ([4] の第 7 章 (P. 84) を参考) から示すことができる[5].

$$e(g_{p_1} + g_{p_2}, g_q) = e(g_{p_1}, g_q) \cdot e(g_{p_2}, g_q) \tag{9}$$

(8) 式から, ペアリング実行後の暗号化された状態での加法演算が実行可能なことも理解される. 例えば, zk-SNARK で用いる (10) 式が成立する.

$$e(g_1^a, g_2^b) \cdot e(g_1^c, g_2^d) = e(g_1, g_2)^{ab} \cdot e(g_1, g_2)^{cd} = \mathrm{g}^{ab} \cdot \mathrm{g}^{cd} = \mathrm{g}^{ab+cd} \tag{10}$$

(7, 8, 9) 式のような計算が成立するペアリング方式として Weil-paring ([4] の第 18 章参照) がある. しかし, 非対称ペアリングは必要なパラメータ数 (楕円曲線上の互いに異なる巡回群を持つ 2 点 (g_1, g_2)) が多く, パラメータが 1 つでよい (楕円曲線上の 1 点 g) 対称ペアリングの方が扱いやすい. 本稿で紹介する zk-SNARK でも, 同一巡回群に含まれる楕円曲線上の生成元 g をベースとした準同型暗号の加法検証を多用する方が見通しが良い. このため, (8) 式から対称ペアリング関数 e' を定義する. 楕円曲線のある点 g に対し, 巡回群 <g> に含まれない g' への線形写像を $\phi(g)$ とする. ϕ は distortion 写像と呼ばれ, 非対称ペアリングから対称ペアリング e' を次式により作成可能である.

$$e'(g^a, g^b) \equiv e(g^a, \phi(g^b)) = e(g^a, \{\phi(g)\}^b) = e(g^a, g'^b) = e(g, g')^{ab} \tag{11}$$

特に $e'(g^a, g^b) = e'(g^b, g^a)$ となるので対称ペアリングと呼ばれる. (10) 式と (11) 式を用いて, 対称ペアリングに対しても (12) 式の成立が確認できる.

$$e'(g^a, g^b) \cdot e'(g^c, g^d) = e(g^a, g'^b) \cdot e(g^c, g'^d) = e(g, g')^{ab+cd} \tag{12}$$

(7, 8, 11, 12) 式を用いて (楕円曲線暗号と準同型暗号, 及び対称ペアリング), 検証者と証明者は次の手続きを行っていく (ここで, 対称ペアリング関数を改めて e と定義する).

- Verifier: 秘密の値 s をランダムに生成する. その後, $g^{l_i(s)}, g^{r_i(s)}, g^{o_i(s)}$ を $i = 1, 2, ..., n$ について計算し, g^{s^j} を $j = 0, 1, 2, ..., d$ に関して計算して Prover に送信する.
- Prover: $g^{l_i(s)}, g^{r_i(s)}, g^{o_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ から, $g^L = g^{L(s)}$, $g^R = g^{R(s)}$, $g^O = g^{O(s)}$, $g^H = g^{H(s)}$ ($H(x)$ は高々 d 次式であるため計算可能) を計算して, Verifier に送信する
- Verifier: $e(g^L, g^R) = e(g^H, g^{T(s)}) \cdot e(g^O, g)$ となるか確認する

この方法は $T(s)$ を Prover が計算できないため, 一見うまくいくように思われる. しかし, g^{s^j} から $g^{T(s)}$ が計算できるため, 最終式 $e(g^L, g^R) = e(g^H, g^{T(s)}) \cdot e(g^O, g)$ を満たす g^L, g^R, g^H, g^O を適当に, 無数に計算可能である (例えば, $g^L = g^5$, $g^R = \{g^{T(s)}\}^2$, $g^H = g^6$, $g^O = \{g^{T(s)}\}^4$ とでき

[3] 一方, 既知の値 c と, 暗号化された値 g^s から, $g^{cs} = \{g^s\}^c$ のように計算できる. このことを zk-SNARK では存分に利用している.

[4] $e(g_1, g_2) = \mathrm{g}$ は有限体上の値で既知ではあるが, ペアリングした後の値 A から $A = \mathrm{g}^{ab}$ となる ab を求めるのは離散対数問題 (DLP: Discrete logarithm problem) を解くことになり困難である. ペアリングとは 楕円離散対数問題 (ECDLP: Elliptic curve discrete logarithm problem) を DLP に置き換える手法でもある.

[5] (9) 式に $g_{p_1} = g_{p_2} = g_1$, $g_q = g_2$ を代入すれば, $e(2g_1, g_2) = e(g_1, g_2) \cdot e(g_1, g_2) = e(g_1, g_2)^2$ となる. これを帰納的に繰り返せば (8) 式が得られる.

る). また, $H(s)$ 計算用の g^{s^j} を使って, 勝手な QAP に対応した $L(s)$, $R(s)$, $O(s)$ を作る可能性もある[6] ($L(x), R(x), O(x)$ は高々 d 次の式である). これら問題に対処するために, まずは α-shift と呼ばれる手法について説明する.

1.3.1 多項式 $L(s), R(s), O(s)$ を制約する α-shift

1.3 節の問題を解決する α-shift について説明する. 1.3 節のアルゴリズムに加えて, Verifier は新たに α と呼ばれる秘密の値を生成し, $g^{\alpha l_i(s)}$ $(i = 1, 2, ..., n)$ を Prover に送信する ($r_i(s), o_i(s)$ についても同様に送信する). $g^{\alpha H(s)}$ は計算する必要がないため, $g^{\alpha s^j}$ $(j = 0, 1, 2, ..., d)$ は送らない (この手続きが重要で, 仮に $g^{\alpha s^j}$ が送信されると, これらの暗号化した値から証明者は新しく任意の QAP に従う $g^{\alpha L} = g^{\alpha L(s)}, g^{\alpha R} = g^{\alpha R(s)}, g^{\alpha O} = g^{\alpha O(s)}$ を計算できてしまう). これらから, 証明者は検証者に次式が成立すると主張できる.

$$e(g^L, g^\alpha) = e(g^{\alpha L}, g), \tag{13}$$

$$e(g^R, g^\alpha) = e(g^{\alpha R}, g), \tag{14}$$

$$e(g^O, g^\alpha) = e(g^{\alpha O}, g). \tag{15}$$

ここで, α は証明者にとって未知である ($g^{L(s)}$ と $g^{\alpha L(s)}$ などから α を導くことは ECDLP を解いたことになる) ため, (13, 14, 15) 式を満たす $g^{\alpha L}$ は $g^{\alpha l_i(s)}$ $(i = 1, 2, ..., n)$ の線型結合により表現されることになる. つまり, α シフトした $g^{\alpha l_i(s)}$, $g^{\alpha r_i(s)}$, $g^{\alpha o_i(s)}$ によって送信可能な多項式が制約される. 次に, ここまでの手続きをまとめる.

- Verifier: 秘密の値 s, α をランダムに生成する. その後, $g^{l_i(s)}$, $g^{r_i(s)}$, $g^{o_i(s)}$, $g^{\alpha l_i(s)}$, $g^{\alpha r_i(s)}$, $g^{\alpha o_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ を計算して Prover に送信する.
- Prover: $g^{l_i(s)}$, $g^{r_i(s)}$, $g^{o_i(s)}$, $g^{\alpha l_i(s)}$, $g^{\alpha r_i(s)}$, $g^{\alpha o_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ から, $g^L = g^{L(s)}$, $g^R = g^{R(s)}$, $g^O = g^{O(s)}$, $g^{\alpha L} = g^{\alpha L(s)}$, $g^{\alpha R} = g^{\alpha R(s)}$, $g^{\alpha O} = g^{\alpha O(s)}$, $g^H = g^{H(s)}$ ($H(x)$ は高々 d 次の式であるから計算可能) を計算して, Verifier に送信する
- Verifier: $e(g^L, g^R) = e(g^H, g^{T(s)}) \cdot e(g^O, g)$ となるか確認する. また,
  $e(g^L, g^\alpha) = e(g^{\alpha L}, g)$,
  $e(g^R, g^\alpha) = e(g^{\alpha R}, g)$,
  $e(g^O, g^\alpha) = e(g^{\alpha O}, g)$ が成立するか確認する.

α が証明者にとって, 非公開情報であることが重要である. また, 後の zk-SNARK が上手く働くことの重要な要素として, 検証者が α について知らなくても, g^α を知っていれば検証可能なことについても着目しておきたい.

ここで, α を各 $L(s), R(s), O(s)$ に対して異なるように取ることも可能である. こうすることで, $g^{\alpha_l s^j}$ $(j = 0, 1, ..., d)$ は $g^{\alpha l_i(x)}$ $(i = 1, ..., n)$ からしか計算できなくなり ($g^{\alpha r_i(x)}, g^{\alpha r_i(x)}$ との線形結合で表されるケースを無くす), セキュリティを高めることができる. これより, 前の手続きを次のように更新することができる.

[6] 例えば, $f(x) = ax^3 + b \cdot x + c$ について $g^{f(s)} = \{g^{s^3}\}^a \cdot \{g^{s^1}\}^b \cdot \{g^{s^0}\}^c = g^{as^3 + b \cdot s + c}$ のように計算できる.

- Verifier: 秘密の値 $s, \alpha_l, \alpha_r, \alpha_o$ をランダムに生成する. その後, $g^{l_i(s)}$, $g^{r_i(s)}$, $g^{o_i(s)}$, $g^{\alpha_l l_i(s)}$, $g^{\alpha_r r_i(s)}$, $g^{\alpha_o o_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ を計算して Prover に送信する.

- Prover: $g^{l_i(s)}$, $g^{r_i(s)}$, $g^{o_i(s)}$, $g^{\alpha_l l_i(s)}$, $g^{\alpha_r r_i(s)}$, $g^{\alpha_o o_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ から, $g^L = g^{L(s)}$, $g^R = g^{R(s)}$, $g^O = g^{O(s)}$, $g^{\alpha_l L} = g^{\alpha_l L(s)}$, $g^{\alpha_r R} = g^{\alpha_r R(s)}$, $g^{\alpha_o O} = g^{\alpha_o O(s)}$, $g^H = g^{H(s)}$ ($H(x)$ は高々 d 次の式であるから計算可能) を計算して, Verifier に送信する

- Verifier: $e(g^L, g^R) = e(g^H, g^{T(s)}) \cdot e(g^O, g)$ となるか確認する. また,
  $e(g^L, g^{\alpha_l}) = e(g^{\alpha_l L}, g)$,
  $e(g^R, g^{\alpha_r}) = e(g^{\alpha_r R}, g)$,
  $e(g^O, g^{\alpha_o}) = e(g^{\alpha_o O}, g)$ が成立するか確認する.

1.3.2 多項式 $L(x), R(x), O(x)$ 計算時に代入される c_i $(i = 1, ..., n)$ を同じにする

$g^{l_i(s)}$ $(i = 1, 2, ..., n)$ から $g^{L(s)}$ を計算するのに $c1, c2, ..., c5$ を用いる ($g^{r_i(s)}$, $g^{o_i(s)}$ も同様). しかし, この c_i が各 $L(s), R(s), O(s)$ に同じ値として代入されているか確認する検証はまだ行っていない. このため, 秘密の値 $\beta_l, \beta_r, \beta_o$ を用いて $g^{z_i(s)} = g^{\beta_l l_i(s) + \beta_r r_i(s) + \beta_o o_i(s))}$ を計算し証明者に送信する. この暗号化された値を c_i により乗法して, $i = 1, 2, ..., n$ について和を取った $g^Z = g^{Z(s)} = \prod_{i=1}^n \{g^{z_i(s)}\}^{c_i}$ を検証者に返す. このようにすると次式が成立する.

$$e(g^L, g^{\beta_l}) \cdot e(g^R, g^{\beta_r}) \cdot e(g^O, g^{\beta_o}) = e(g^Z, g) \tag{16}$$

ここで, 秘密の値 $\beta_l, \beta_r, \beta_o$ を $\beta_l = \beta_l' \cdot s^{d+1}$, $\beta_r = \beta_r' \cdot s^{2(d+1)}$, $\beta_o = \beta_o' \cdot s^{3(d+1)}$ と定義して乱数生成する. こうすることで, $z_i(s) = \beta_l' s^{d+1} l_i(s) + \beta_r' s^{2(d+1)} r_i(s) + \beta_o' s^{3(d+1)} o_i(s))$ の各項は互いに独立のように扱うことが可能となる. つまり, $z_i(s)$ の線型結合で表される $Z(s)$ を用いて, $L(s), R(s), O(s)$ がユニークに決定可能ということが理解される.

しかし, $g^{\beta_l}, g^{\beta_r}, g^{\beta_o}$ は検証者側で公開されているため, $z_i(s)$ が変更される可能性がある. 例えば, $g^{z_i'(s)} \equiv g^{z_i(s)} \cdot g^{\beta_l} = g^{\beta_l (l_i(s)+1) + \beta_r r_i(s) + \beta_o o_i(s)}$ と変更できる. このことを防ぐために, 新たな乱数 γ を用いて $g^{\beta_l}, g^{\beta_r}, g^{\beta_o}$ を秘匿する. 方法は簡単で (16) 式を (17) 式のように変更するだけである.

$$e(g^L, g^{\beta_l \gamma}) \cdot e(g^R, g^{\beta_r \gamma}) \cdot e(g^O, g^{\beta_o \gamma}) = e(g^Z, g^\gamma) \tag{17}$$

ここまでの手続きをまとめると, 次のようになる (これは [2] の論文の Protocol 1 に該当する).

- Verifier: 秘密の値 $s, \alpha, \beta_l, \beta_r, \beta_o$ をランダムに生成する. その後, $g^{l_i(s)}$, $g^{r_i(s)}$, $g^{o_i(s)}$, $g^{\alpha l_i(s)}$, $g^{\alpha r_i(s)}$, $g^{\alpha o_i(s)}$, $g^{z_i(s)} = g^{\beta_l l_i(s) + \beta_r r_i(s) + \beta_o o_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ を計算して Prover に送信する.

- Prover: $g^{l_i(s)}$, $g^{r_i(s)}$, $g^{o_i(s)}$, $g^{\alpha l_i(s)}$, $g^{\alpha r_i(s)}$, $g^{\alpha o_i(s)}$, $g^{z_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ から, $g^L = g^{L(s)}$, $g^R = g^{R(s)}$, $g^O = g^{O(s)}$, $g^{\alpha L} = g^{\alpha L(s)}$, $g^{\alpha R} = g^{\alpha R(s)}$, $g^{\alpha O} = g^{\alpha O(s)}$, $g^H = g^{H(s)}$, $g^Z = g^{Z(s)}$ を計算して, Verifier に送信する

- Verifier: $e(g^L, g^R) = e(g^H, g^{T(s)}) \cdot e(g^O, g)$ となるか確認する. また,
  $e(g^L, g^\alpha) = e(g^{\alpha L}, g)$,
  $e(g^R, g^\alpha) = e(g^{\alpha R}, g)$,
  $e(g^O, g^\alpha) = e(g^{\alpha O}, g)$,
  $e(g^L, g^{\beta_l \gamma}) \cdot e(g^R, g^{\beta_r \gamma}) \cdot e(g^O, g^{\beta_o \gamma}) = e(g^Z, g^\gamma)$ が成立するか確認する.

1.3.3 これまでの手続きを効率的にする Pinoccio プロトコル

(17) 式の比較のためには, 計算コストの高いペアリングを 4 回実行する必要がある. これを 2 回の手続きに減らしたのが Pinoccio プロトコルである. 最初にその手続きについて説明する.

- Verifier: 秘密の値 $s, \alpha_l, \alpha_r, \alpha_o, \beta, \gamma, \rho_l, \rho_r, \rho_o = \rho_l \cdot \rho_r$ をランダムに生成する. その後, 生成元として, $g_l = g^{\rho_l}$, $g_r = g^{\rho_r}$, $g_o = g^{\rho_o}$ を定義する. $g_l^{l_i(s)}$, $g_r^{r_i(s)}$, $g_o^{o_i(s)}$, $g_l^{\alpha_l l_i(s)}$, $g_r^{\alpha_r r_i(s)}$, $g_o^{\alpha_o o_i(s)}$, $g^{z_i(s)} = g_l^{\beta l_i(s)} \cdot g_r^{\beta r_i(s)} \cdot g_o^{\beta o_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ を計算して Prover に送信する.
- Prover: そして, $g_l^{l_i(s)}$, $g_r^{r_i(s)}$, $g_o^{o_i(s)}$, $g_l^{\alpha_l l_i(s)}$, $g_r^{\alpha_r r_i(s)}$, $g_o^{\alpha_o o_i(s)}$, $g_l^{\beta l_i(s)}$, $g_r^{\beta r_i(s)}$, $g_o^{\beta o_i(s)}$ $(i = 1, 2, ..., n)$ と g^{s^j} $(j = 0, 1, 2, ..., d)$ から, $g_l^L = g_l^{L(s)}$, $g_r^R = g_r^{R(s)}$, $g_o^O = g_o^{O(s)}$, $g_l^{\alpha_l L} = g_l^{\alpha_l L(s)}$, $g_r^{\alpha_r R} = g_r^{\alpha_r R(s)}$, $g_o^{\alpha_o O} = g_o^{\alpha_o O(s)}$, $g^H = g^{H(s)}$,
  $g^Z = g^{Z(s)} = \prod_{i=1}^{n} \left(g_l^{\beta l_i(s)} \cdot g_r^{\beta r_i(s)} \cdot g_o^{\beta o_i(s)}\right)^{c_i}$ を計算して, Verifier に送信する
- Verifier: $e(g_l^L, g_r^R) = e(g^H, g_o^{T(s)}) \cdot e(g_o^O, g)$ $(\rho_l \rho_r LR = \rho_l \rho_r T(s) H + \rho_l \rho_r O$ となる),
  $e(g_l^L, g^{\alpha_l}) = e(g_l^{\alpha_l L}, g)$,
  $e(g_r^R, g^{\alpha_r}) = e(g_r^{\alpha_r R}, g)$,
  $e(g_o^O, g^{\alpha_o}) = e(g_o^{\alpha_o O}, g)$,
  $e(g_l^L \cdot g_r^R \cdot g_o^O, g^{\beta\gamma}) = e(g^Z, g^\gamma)$ が成立するか確認する.

本稿では, 付録 D, E に Pinoccio プロトコルが暗号学的に安全になる証明を記述する.

1.4 検証者による入出力が与えられた Pinoccio プロトコル

1.3.3 節までの手続きでは, 入力も出力も考慮していなかった. 例えば, ある出力を与える入力を知っていることや, ある入出力が QAP に従って正しく計算されている (Verifiable Computation) ことを証明するケースについては考えていなかった. 本節では, その手法について理解する.

1.4.1 QAP による変数への制約

例えば, 変数 c_i が次の式を満たす場合, $c_i = 0$ or 1 となる.

$$c_i = c_i * c_i$$

また, $c_i = 2$ としたい場合は, 次式を満たす QAP を作ればよい.

$$0 = (c_i - 2) * 1$$

また, 4 bit の数 $c_4 = 8 * c_3 + 4 * c_2 + 2 * c_1 + c_0$ を表現したい場合は, 次式のように表現できればよい.

$$c_4 = (8 * c_3 + 4 * c_2 + 2 * c_1 + c_0) * 1$$
$$c_3 = c_3 * c_3$$
$$c_2 = c_2 * c_2$$
$$c_1 = c_1 * c_1$$
$$c_0 = c_0 * c_0$$

このため, $l_3(1) = 8, l_2(1) = 4, l_1(1) = 2, l_0(1) = 1$ のように定義できればよい (これは, 付録 A のラグランジュ補完より容易). 定数を取り扱いたい場合は, 検証者は自身で用意した $c_* = v_{one} = 1$ を用いた演算をすればよい.

1.4.2 検証者により定めることのできる入出力変数

n 個の変数の内, $m+1$ 個を検証者が与える変数とする ($c_0 \equiv v_{\text{one}} = 1$ とする). また, $i = 0$ から m までの演算は検証者が実行するため, 前節までの証明者が行うべき手続きは, $i = m+1$ から $i = n$ までの変数に関して実行すればよい. 1 例として, $g^{L(s)}$ は次のように計算される.

$$g^{L_p(s)} = \prod_{i=m+1}^{n} g^{c_i l_i(s)} \tag{18}$$

$$g^{L_v(s)} = g^{l_0(s)} \cdot \prod_{i=1}^{m} g^{c_i l_i(s)} \tag{19}$$

$$g^{L(s)} = g^{L_v(s)} \cdot g^{L_p(s)} \tag{20}$$

(18) 式の値は, 証明者の下で計算される. (20) 式の値は, 証明者から送信された (18) 式の値と, 検証者自身が計算した (19) 式の値を代入することで計算される. 同様に, $g^{R(s)}, g^{O(s)}, g^{Z(s)}$ も計算される.

1.5 ゼロ知識性をもたらす定数項 δ による加算

ゼロ知識性を出すには, 暗号化した値にランダムな暗号化した値を足せばよいだけである. よって, Naive な直観に従うと, 各 $g^{L(s)}, g^{R(s)}, g^{O(s)}$ にランダムな値 $g^{\delta_l}, g^{\delta_r}, g^{\delta_o}$ を足せばよい. この結果, 多項式の演算の検証は次式によって行われることになるが, 当然式は一致しない.

$$e(g_l^{\delta_l + L(s)}, g_r^{\delta_r + R(s)}) \neq e(g_o^{T(s)}, g^{H(s)}) \cdot e(g_o^{\delta_o + O(s)}, g) \tag{21}$$

式を一致させるために, $H(s)$ に調整役となる $\delta_l, \delta_r, \delta_o, L(s), R(s), O(s)$ に依存した項を加える. $H(s) = \Delta + H(s)$ と再定義すると, (21) 式の統合が成立するような Δ は次式のようになる.

$$\Delta = \frac{\delta_l R(s) + \delta_r L(s) + \delta_l \delta_r - \delta_o}{T(s)} \tag{22}$$

従って, $\delta_l, \delta_r, \delta_o, \Delta$ に $T(s)$ をかけたものを $\delta_l, \delta_r, \delta_o, \Delta$ に関して置き換えれば検証式を次式のように変形できる. 秘匿のための計算部分は赤字で示した.

$$L(s) \cdot R(s) - O(s) + T(s)(\delta_l R(s) + \delta_r L(s) + T(s)\delta_l \delta_r - \delta_o) = T(s)H(s) + T(s)\Delta \tag{23}$$

暗号化された値の等号を一致させるように, $\delta_l, \delta_r, \delta_o, \Delta$ を調整するのは不可能である (ECDLP を解くことになるため). 従って, 上記のようにすることでゼロ知識性が確認できた.

2 zk-SNARK のまとめ

前までの手続きはインタラクティブなプロトコルを前提としているが, 予め誰にもランダムな値を公開しないことで, 非インタラクティブな手続きへと変更することができる. しかし, 仮にこのランダムな値がパブリックに公開されると, 楕円曲線による準同型暗号の暗号化された値同士の積を計算する (g, g^a, g^b から g^{ab} を求める DH 問題を解く) のが困難という性質が意味を持たなくなる (g, g^a, b から $\{g^a\}^b = g^{ab}$ と計算できてしまう). これが, Trusted setup が zk-SNARK に必要とされる理由である. 近年は, Trusted setup が不要な zk-STARK なども提案されており, 引き続き調査を続けたい.

最後にまとめると, zk-SNARK は次の 3 段階の手続きで実行される. 赤の手続きはゼロ知識証明用の手続きである.

- 信頼できるセットアップ (Trusted setup):

  1. 生成元 (generator) となる楕円曲線上の点 g を生成し, その楕円曲線上の 2 点から計算される対称ペアリング e を定義する.
  2. QAP により定義される $\{l_i(x), r_i(x), o_i(x)\}$ $(i = 1, 2, ..., n)$ と $t(x)$ を計算する ($t(x)$ の次数 d は, (1, 2) 式のような等式の数である).
  3. $s, \rho_l, \rho_r, \alpha_r, \alpha_l, \alpha_o, \beta, \gamma$ をランダムに生成する (これら値は, 誰にも知られず消去される必要がある. この手続きを含めて Trusted setup と呼ばれる [8]).
  4. $\rho_o = \rho_l \cdot \rho_r$, $g_l = g^{\rho_l}$, $g_r = g^{\rho_r}$, $g_o = g^{\rho_o}$ を計算する.
  5. 証明者用の鍵を公開する (各 $g_l^{l_i(s)}$, $g_l^{\alpha_l l_i(s)}$ 等は楕円曲線上の点で独立なベクトルのように見做せる).
     $\Big(\{g^{s^k}\}_{k\in\{0,1,...,d\}}, \{g_l^{l_i(s)}, g_r^{r_i(s)}, g_o^{o_i(s)}\}_{i\in\{0,1,...,n\}},$
     $\{g_l^{\alpha_l l_i(s)}, g_r^{\alpha_r r_i(s)}, g_o^{\alpha_o o_i(s)}, g^{z_i(s)} = g_l^{\beta l_i(s)} g_r^{\beta r_i(s)} g_o^{\beta o_i(s)}\}_{i\in\{m+1,1,...,n\}},$
     $\{g_l^{t(s)}, g_r^{t(s)}, g_o^{t(s)}, g_l^{\alpha_l t(s)}, g_r^{\alpha_r t(s)}, g_o^{\alpha_o t(s)}, g_l^{\beta t(s)}, g_r^{\beta t(s)}, g_o^{\beta t(s)}\}$ $\Big)$
  6. 検証者用の鍵を公開する.
     $\big(g^1, g_o^{t(s)}, \{g_l^{l_i(s)}, g_r^{r_i(s)}, g_o^{o_i(s)}\}_{i\in\{0,1,...,m\}}, g^{\alpha_l}, g^{\alpha_r}, g^{\alpha_o}, g^{\gamma}, g^{\beta\gamma} \big)$

- 証明者 (Prover):

  1. 与えられた入力 $\boldsymbol{u}$ (一部は検証者が知らない値あり (zcash なら coin の量など)) から各 Argument (変数) の値 $\{c_i\}_{i\in\{0,...,m\}}$ (検証者も知る値), $\{c_i\}_{i\in\{m+1,...,n\}}$ (検証者が知らない値) を計算する.
  2. 全値から多項式を評価する $L(x) = l_0(x) + \sum_{i=1}^{n} c_i \cdot l_i(x)$ (同様に $R(x)$, $O(x)$ を計算する)
  3. ランダムな δ_l, δ_r, δ_o を 生成する. (この値を破棄すれば, 二度と同じ証明は作れなくなる. つまり, 誰かに秘密情報を公開されてもそれが正しいか判断できない!)
  4. $h(x)$ を次のように計算する. $h(x) = \frac{L(x)R(x)-O(x)}{t(x)} + \delta_r L(x) + \delta_l R(x) + \delta_l\delta_r t(x) - \delta_o$ (δ_l, δ_r が $L(x), R(x)$ に対して, クロスしていることに注意!)
  5. 証明者のみが計算可能な, 暗号化されている多項式部分 (Encrypted variable polynomials) を計算する. そして, ゼロ知識のために δ-shift した値を追加する.
     $g_l^{L_p(s)} = \big(g_l^{t(s)}\big)^{\delta_l} \cdot \prod_{i=m+1}^{n} \big(g_l^{l_i(s)}\big)^{c_i}$ (同様に, $g_r^{R_p(s)}, g_o^{O_p(s)}$ を計算する)
  6. 上記手続き 5 の α シフトバージョンを計算する.
     $g_l^{L'_p(s)} = \big(g_l^{\alpha_l t(s)}\big)^{\delta_l} \cdot \prod_{i=m+1}^{n} \big(g_l^{\alpha_l l_i(s)}\big)^{c_i}$ (同様に, $g_r^{R'_p(s)}, g_o^{O'_p(s)}$ を計算する)
  7. 各多項式 $L(x), R(x), O(x)$ に代入される変数 c_i $(i = m+1, ..., n)$ の一致を保証する.
     $g^{Z(s)} = \big(g_l^{\beta t(s)}\big)^{\delta_l} \big(g_r^{\beta t(s)}\big)^{\delta_r} \big(g_o^{\beta t(s)}\big)^{\delta_o} \cdot \prod_{i=m+1}^{n} \big(g_l^{\beta l_i(s)} g_r^{\beta r_i(s)} g_o^{\beta o_i(s)}\big)^{c_i}$
  8. 検証者への証明を作成する $\big(g_l^{L_p(s)}, g_r^{R_p(s)}, g_o^{O_p(s)}, g^{h(s)}, g_l^{L'_p(s)}, g_r^{R'_p(s)}, g_o^{O'_p(s)}, g^{Z(s)} \big)$

- 検証者 (Verifier):

  1. 証明をパースする $\big(g_l^{L_p}, g_r^{R_p}, g_o^{O_p}, g^{h}, g_l^{L'_p}, g_r^{R'_p}, g_o^{O'_p}, g^{Z} \big)$
  2. 自分の既知の変数で多項式の一部分を作成する (zcash で言うと所持金や秘密の値などのハッシュ値である)
     $g_l^{L_v(s)} = g_l^{l_0(s)} \cdot \prod_{i=1}^{m} \big(g_l^{l_i(s)}\big)^{i}$ (同様に, $g_r^{R_v(s)}, g_r^{O_v(s)}$ を計算する)

3. 多項式の制約確認 $e(g_l^{L_p}, g^{\alpha}) = e(g_l^{L'_p}, g)$ (同様に, $g_r^{R_p(s)}, g_o^{O_p(s)}$ の検証をする)
4. 各多項式で扱われる変数の一致を確認 $e(g_l^{L_p} g_r^{R_p} g_o^{O_p}, g^{\beta\gamma}) = e(g^Z, g^{\gamma})$
5. 正しい計算がされていることを確認 $e(g_l^{L_p} g_l^{L_v(s)}, g_r^{R_p} g_r^{R_v(s)}) = e(g_o^{t(s)}, g^h) \cdot e(g_o^{O_p} g_o^{O_v(s)}, g)$

参考文献

[1] Maksym Petkus, "Why and How zk-SNARK Works: Definitive Explanation," https://arxiv.org/pdf/1906.07221.pdf.

[2] B. Parno and C. G. M. Raykova, "Pinocchio: Nearly Practical Verifiable Computation," https://eprint.iacr.org/2013/279.pdf.

[3] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf.

[4] 光成滋生 「クラウドを支えるこれからの暗号技術」 秀和システム (2015).

[5] R. Gennaro C. Gentry B. Parno and M. Raykova "Quadratic span programs and succinct NIZKs without PCPs " in EUROCRYPT 2013.

[6] V. Buterin, "Zk-SNARKs: Under the Hood," https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6.

A 補足: ラグランジュ補完の例

2 次の x に関する多項式 $f(x)$ が点 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ を通過する場合, その時の $f(x)$ は次式によってユニークに表現できる. このことに関する解説は V. Buterin の Medium への寄稿 [8] が分かりやすい.

$$\begin{aligned} f(x) &= y_1 \cdot \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + y_2 \cdot \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + y_3 \cdot \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} \\ &= y_1 \cdot f_1(x) + y_2 f_2(x) + y_3 f_3(x) \end{aligned} \tag{24}$$

これを QAP を満たす (4) 式の作成に応用する. r_j を代入することで j 番目の式を評価可能にする $R_j(x)$ は次式で表せる.

$$R_j(x) = \frac{\prod_{k=1,k\neq j}^{d}(x - r_k)}{\prod_{k=1,k\neq j}^{d}(r_j - r_k)} \tag{25}$$

これに加えて, 変数が全ての式に出現しないようにする $\overline{R}(x)$ を次式で定義する.

$$R_{\text{all}}(x) = \prod_{k=1}^{d}(x - r_k) \tag{26}$$

$R_j(x)$ は他の $R_k(x)$ $(k \neq j)$, $\overline{R}(x)$ の線型結合で表現できない事に注意しよう. このことは, 次式の左辺右辺に $x = r_j$ を代入して評価すればすぐに分かる.

$$R_j(x) \neq \sum_{k=1,k\neq j}^{d} c_i R_i(x) + \overline{c}\overline{R}(x) \tag{27}$$

つまり, 多項式 $R_j(x)$ $(j = 1, 2, ..., d)$ と $\overline{R}(x)$ によって, 任意の d 次の多項式はユニークに計算される. これが, $z_i(s)$ の線型結合で計算された $Z(s)$ によって, $L(s), R(s), O(s)$ がユニークに定まる理由である ($d = 1$ の場合は, $R_1(x) = 1$, $\overline{R}(x) = x - 1$ とすれば, 任意の 1 次の多項式 $ax + b$ が一意の α, β によって, $\alpha\overline{R}(x) + \beta R_1(x)$ と表せる $(\alpha = a, \beta = a + b)$).

B α-shift による多項式の制約について

(13) 式により, α-shift が検証している式は次式が任意の i について $c_i = c'_i$ となることである ((14, 15) 式についても同様である).

$$\left(\prod_{i=0}^{d}(g^{s^i})^{c_i}\right)^{\alpha} = \prod_{i=0}^{d}(g^{\alpha s^i})^{c_i} = \prod_{i=0}^{d}(g^{\alpha s^i})^{c'_i} \tag{28}$$

付録 C の DH 仮定より, g^{s^i} は i が異なれば, 互いに異なるベクトルのように見做すことができる. 従って, $c_i \neq c'_i$ となる c'_i を見つけるのは困難である. このような想定 (assumption) は, [2] 内で d-PKE assumption と呼ばれている (Appendix D の Assumption を参照).

c_i がユニークに定まるという事は, s によって評価された多項式がユニークに定まるということである. 従って, 付録 A より, 多項式を表現する $R_i(x)$ $(i = 1, ..., n)$ の係数についてもユニークに定まる ($R_i(x)$ は互いに独立な多項式).

C DH (Diffie-Hellman) 仮定

DH 仮定とは 「楕円曲線上の点 P について, P, aP, bP が与えられた場合に, ランダムな c から計算された値 cP が abP と一致しているか判定する問題」 を解くのが困難という仮定である ([4] の P 159 を参照). 言い換えれば P, aP, bP から abP を求めるのが困難ということである. また, 以下の 3 つは DH 問題と等価である.

I. P, aP, bP から abP を求める.

II. P, aP から a^2P を求める.

III. P, aP から $a^{-1}P$ を求める.

II は I において $b = a$ とした特殊なケースである. また, II において $P, aP, bP, (a + b)P$ から $a^2P, b^2P, (a + b)^2P$ が求められるので, $2abP$ が計算できてこれを 2 で割ると abP が得られる. 従って, I と II は等価な問題である.

II において (aP) と $a^{-1}(aP)$ から $a^{-1}P$ が得られる. また, III において $(a^{-1}P)$ と $a^{-1}(a^{-1}P)$ から a^2P を求められる. 従って, II と III は等価な問題である.

これらから, 次のことが困難と予想される.

IV. $P, aP, a^2P, ..., a^nP$ から $a^{n+1}P$ を求める.

V. $P, aP, a^2P, ..., a^nP$ から $a^{-1}P$ を求める.

IV, V は n-weak DH 問題と呼ばれる．また, n-weak 問題を少し緩めた n-SDH (Strong DH) と呼ばれる問題もある．n-SDH では, $P, aP, a^2P, ..., a^nP$ から a^{-1} に限らず $(a+c)^{-1}$ (c は任意の数) を求めればよい．より優しい問題になったが, セキュリティ的には脆弱になっているため, n-SDH と呼ばれる．しかし, この問題を解くのも十分に難しいことが知られている (Appendix D の Assumption D.3 を参照).

D Pinoccio プロトコルの安全性証明のための 3 つの想定

D.1 q-PDH Assumption

q-power Diffie-Hellman の略である．楕円曲線上の点 $g, g^s, g^{s^2}, ..., g^{s^q}, g^{s^{q+2}}, ..., g^{s^{2q}}$ から $g^{s^{q+1}}$ を求めるのは困難という仮定である．

D.2 q-PKE Assumption

q-power Polynomial Knowledge of Exponent の略である．未知の α に対して, $g.g^s, g^{s^2}, ..., g^{s^q}$, $g^\alpha, g^{\alpha s}, g^{\alpha s^2}, ..., g^{\alpha s^q}$ から次式を満たす c を見つけるのは困難という仮定である．

$$g^{c'} = \{g^c\}^\alpha \wedge c \neq \prod_{i=0}^{q} g^{a_i s^i} \tag{29}$$

つまり, α によって, c が制約されるという α-shift の有効性を表現する仮定である．

D.3 q-SDH Assumption

q-Strong Diffie Hellman の略である．$g, g^s, g^{s^2}, ..., g^q$ から $g^{1/(s+r)}$ を見つけるのは困難という仮定である (r は任意の値). [2] の論文では, ペアリングを含めた $y = e(g,g)^{1/(s+r)}$ となる y すら見つけるのが困難という仮定で証明を行っている．

E [2] の Appendix B, Security Proof の概要説明

本付録では, Pinoccio プロトコルがなぜ安全となるか説明する．まず, D.2 節の q-PKE 仮定より, Pinoccio プロトコルで計算される g^L, g^R, g^O は $g, g^s, g^{s^2}, ..., g^{s^q}$ の線型結合で表される．一方, 各 $g^{z_i(s)}$ は次式で表されることに注意しよう．

$$g^{z_i(s)} = g^{\beta(\rho_l l_i(s) + \rho_r r_i(s) + \rho_o o_i(s))} \ (\rho_o = \rho_l \cdot \rho_r) \tag{30}$$

ここで, $\rho_l = \rho_l' s^{d+1}$, $\rho_r = \rho_r' s^{2(d+1)}$, $\rho_o = \rho_l \cdot \rho_r = \rho_o' s^{3(d+1)}$ と設定しよう (このようにおいても, 乱数 $s^{k(d+1)}$ $(k = 1, 2, 3)$ に一様乱数をかければ一様乱数になるだけなので, 全く問題ない). こうすることで, $l_i(s), r_i(s), o_i(s)$ は高々 d 次であるため, 各々線型結合で互いに表現が困難な楕円曲線上の点に飛ばされたと考えられる (q-PDH の仮定). つまり, $s^{d+1} l_i(s)$ を $g^{s^{2(d+1)} r_i(s)}$, $g^{s^{3(d+1)} o_i(s)}$ $(i = 1, ..., n)$ の線型結合で表すのは困難だろうということである ($r_i(s), o_i(s)$ も同様). 従って, 検証を通過する証明が与えられた時, $g^{Z(s)}$ は必ず $g^{z_i(s)}$ の線型結合によって計算されることさえ示されれば充分である．この証明のため, β を $\beta = s^{q-(4d+3)} \beta_{poly}(s)$ とする．ここで, $\beta_{poly}(x)$ は, 高々 $3d+3$ 次式で, 任意の i について, 多項式 $\beta_{poly}(x)(\rho_l' l_i(x) + \rho_r' s^{d+1} r_i(x) + \rho_o' s^{2(d+1)} o_i(x))) = \beta_{poly}(x) u_i(x)$ の x^{3d+3} の係数が 0 となるように定義する ($u(x)$ は高々 $3d+2$ 次). このような $\beta_{poly}(x)$ は存

在する. また, 上記制約の下でランダムに生成した $\beta_{poly}(x)$ は, $u_i(x)$ の一次結合で表現されない $u(x)$ に関して, $\beta_{poly}(x)u(x)$ の x^{3d+3} の係数は有限体 F_p 上で一様に分布する. つまり, この係数が偶然にゼロとなる $u_i(x)$ の一次結合で表現されない $u(x)$ が運良く選ばれる確率は $1/p$ で抑えられる.

上記の前提の下で, 各種パラメータの s に関する次数について考える. γ を $\gamma = \gamma' s^{q+2}$ として一様乱数から生成する. g^{γ} は, 定義より $g^{s^{q+1}}$ を成分に持たない. $g^{\gamma\beta}$ も $g^{s^{q+1}}$ は成分に持たず, その次数は $q-(4d+3)+(3d+3)+(q+2) = 2q+2-d \leq 2q$ で抑えられる (正確には, $d \geq 2$ が必要である. [2] の論文の P. 15, 2 段組み右の上から 8 行目ではこのことに言及していない. しかし, q-PDH の前提知識に $g^{s^{2q+1}}, g^{s^{2q+2}}$ が加わっても困難なことに違いはないので大きな誤りではない). また, $g^{z_i(s)} = g^{\beta(\rho_l l_i(s)+\rho_r r_i(s)+\rho_o o_i(s))}$ も s^{q+1} の成分は持たず, 高々 $2q$ 次の s に関する多項式を暗号化したものである ($q-(4d+3)+(3d+3)+(3d+3)+d = q+(3d+3) \leq 2q$). 他の与えられている暗号化されたパラメータは全て s に関して高々 d 次の式に高々 $3(d+1)$ 次の式をかけたものに過ぎないので, q-PDH の仮定以上の知識は与えていない.

以上より, 証明が正しく検証を通過するにも関わらず, 命題が間違っているケースについて考える. このケースは 2 パターンに分類される. 1) $T(x)$ が $P(x)$ を割り切らない, または 2) $Z(s)$ が $z_i(s)$ の線型結合で表されない, この 2 つである.

1) が成立する場合, $T(x)$ に含まれ $P(x)$ を割り切れない 1 つの多項式を $(x-r)$ と置き, 多項式に関する最大公約数を $d(x)$, また $T(x) = d(x)\overline{d}(x)(x-r)$ となる $\overline{d}(x)$ を定義する. 多項式に関する拡張ユークリッドの互除法によって $a(x)T(x)+b(x)P(x) = d(x)$ となる多項式 $a(x), b(x)$ が計算できる (各次数は $2d-1$, $d-1$ である). また, 両辺を $T(x)$ で割り $\overline{d}(x)$ をかけると, $a(x)\overline{d}(x)+b(x)\overline{d}(x)P(x)/T(x) = 1/(x-r) \equiv A(x)+B(x)H(x)$ となる. つまり, 証明を満たす $P(s)/T(s) = H(s)$ となるような g^H が既知であるので, $e(g^{A(s)}, g)\cdot e(g^{B(s)}, g^H) = e(g,g)^{1/(s-r)}$ が計算できてしまう. これは, q-SDH 問題を解いたことになるので, このような証明を作るのは不可能である.

2) が成立する場合, 高い確率で Z の s^{q+1} の係数は非ゼロとなる. 従って, 与えられた g^L, g^R, g^O を用いて, その g^{s^j} $(j = 0, ..., d)$ の係数に関する知識が与えられる (この知識を与える仮想のマシンは, [2] 内で d-PKE knowledge extractor と呼ばれている) と, $g^{\rho_l' L(s)}$ の各 g^{s^j} $(j = 0, 1, ..., d)$ の係数が分かる. 従って, 次式で与えられる g^Z から g^{s^j} $(j \neq q+1)$ を引くことで $g^{s^{q+1}}$ に関する知識を得る事ができる. これは, q-PDH 問題を解いたことになる.

$$g^Z = g^{s^{q-(4d+3)}\beta_{poly}(s)(\rho_l' s^{(d+1)}L(s)+\rho_r' s^{2(d+1)}R(s)+\rho_o' s^{3(d+1)}O(s))} \tag{31}$$

各種パラメータは, q-PDH の仮定によってのみ計算可能なことに改めて注意しておきたい (s が分からなくても, 例えば $g^{\beta\gamma} = g^{s^{q-(4d+3)}\beta_{poly}(s)\gamma' s^{q+2}}$ の中に $g^{s^{q+1}}$ が含まれていないので, γ' さえ分かれば暗号化された値の計算は可能).