zkSNARK

zk-SNARKsの理論

ゼロ知識証明の3つの特性(完全性(Completeness)、健全性(Soundness)、ゼロ知識証明(Zero Knowledge))に加えて4つの性質(簡潔(Succinct)、非対話型(Non-interactive)、限界性?(ARgument)、知識の必要性?(Knowledge))があり、その頭文字を取って "SNARK" としている。

全体の流れ

証明者と検証者の間に第三者(Trusted Third Party)を置き、CRSと呼ばれる事前に公開される情報を証明者と検証者に送る

事前にCRSを作成することを Trusted Setup と呼ぶ

この第三者は証明者にも検証者にも「信頼される存在」になる

第三者を信用しないといけないが、その代わりに証明者と検証者が直接やりとりをしなくてもセキュリティを保ち、かつ証明プロセスがが効率的になるそうだ。

証明者はそのCRSを用いて、正しい情報を持っているという証明を生成し検証者に送る

検証者は受け取ったそれを検証する

zkSNARKsの理論

3つのアルゴリズム

Key Generator は Trusted Third Party が使うアルゴリズム

CRSを生成(証明者に渡す評価鍵 `pk` と検証者に渡す検証鍵 `vk` が含まれる)

Prover は証明者が使うアルゴリズム

以下のデータから証明 `π` を導く

評価鍵 `pk`

ステートメント `x`

証拠 `w`

Verifier は検証者が使うアルゴリズム

以下のデータから証明 `π` が真なら `1` 、そうでなければ `0` を返す

検証鍵 `vk`

ステートメント `x`

証明 `π`

複雑性クラス

(計算の複雑さを評価するクラス？？)

(規模が大きくなっても計算量がそこまで大きくならないタイプがクラスP。辞書から

この単語を探せ、みたいな問題は、辞書に載っている単語数が増えたりページが増えたりしても手間はかかるがそのくらい、みたいなタイプ)

(クラスNPは、答が分かっていれば確認は多項式時間でできるが、自分で答を見つけるのには指数時間かかる。多項式時間でできる、というのはネガティブな感じではなく、検証がしらみつぶしにやらないとできないというわけではないというポジティブな感じみたい)

多項式時間というのは。。。

$O()$ という表現でわかるなら、 $O(1)$ ではなく $O(n)$ のように多項式になるタイプ

範囲が広がるほど計算に時間がかかる

クラスNPのうち、解くのが最も難しい問題をNP完全問題と呼ぶ

SAT(命題論理式の充足可能問題)はNP完全問題の一つ

論理式があって、その式のパラメータそれぞれに `1` or `0` を与えて結果が `1` になる組み合わせが存在するか

Circuit-SAT

Circuit-SATというのは、SATを演算回路に変換したもの

論理式と演算回路の違い - ChatGPT

演算回路の方がハードウェアやソフトウェアの人にとって使いやすい、くらいでよいと思う

演算回路があって、その入力のそれぞれに `1` or `0` を与えて出力が `1` にすることができるか

$\forall(x, w)$ は、 R に属する (x, w) はどれでも $f(x, w)$ に与えると `1` になる、という意味・・・のはず。

R はあまり気にしなくてよいのかな？

$$f(x,w) = \begin{cases} 1 & \forall(x,w) \in \mathcal{R} \\ 0 & \forall(x,w) \notin \mathcal{R} \end{cases}$$

証明者は Circuit-SATに対する証拠 w を検証者に渡し、検証者は式に当てはめて出力が `1` となることを多項式時間で検証できる。

ただ w はサイズが非常に大きいので、この辺でZKPっぽい話が出てくる

Circuit-SATからQAPを構築する

「知っていること」に変換するそうだ

が・・・読んでも意味が分からない。。。

よくわからないが、演算回路が $h(x)t(x) = p(x)$ に変換され、これを満たす $h(x)$ を見つける。

$p(x)$ は $t(x)$ で割り切れるという関係というのを利用することになる(間違っていたら割り切れない)

$h(x) = \sum_{i=0}^{d} h_i x^i$ のようになって大きくなって効率が悪いので、さらに変換する

Schwartz-Zippleの補題なるものを利用する

全部の x_i ではなくランダムに選択した点 s で評価すると、間違っている場合は高確率で間違った答になるというものらしい。

ランダムで済ませるなんてあやしくないか？

「簡素な証明」と書いてあるので、1次でNGだったものはそれで終わり、残りを時間がかかる2次検査する、みたいな形か?

エンコーディング方式

ゼロ知識証明したいので、証明者は極力検証者に詳細を与えないようにする。

多項式や評価点を渡さないようにする。

評価点というのはランダムで選んだという s のこと？

ここら辺から付いていけなくなった・・・

Pinocchio Protocol

zk-SNARKsの厳密な構成（Zcash）

ZKP（zkSNARKs）の使い方と活用事例

Ethereumの場合はコントラクトを実行(コントラクトへの送金)を行うと、Bitcoinとはいろいろ違ったことができる

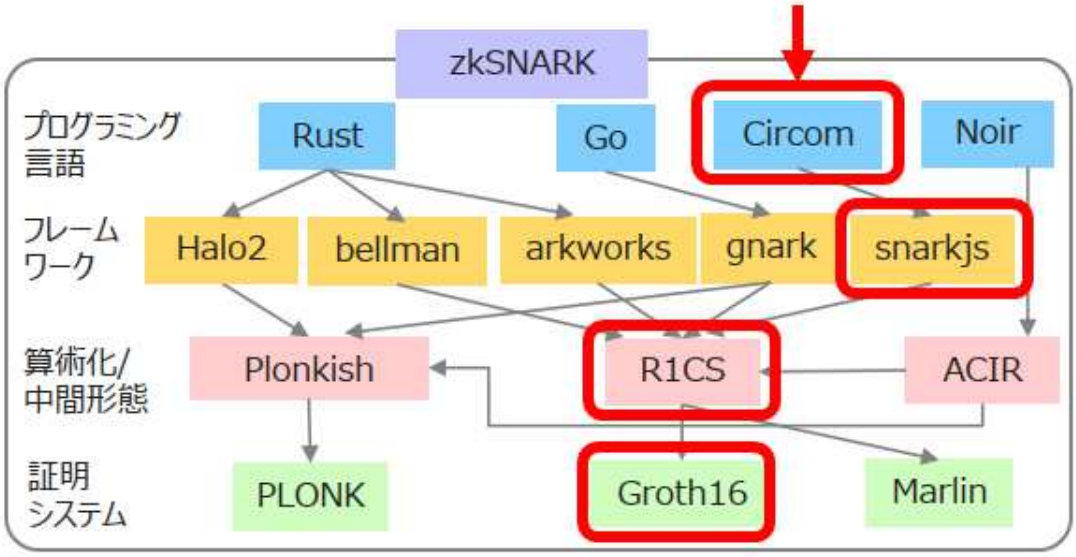
別のコントラクトを呼び出せる

データを保持できるので、たとえばどのアドレスからいくら送金された、みたいなデータを持つことができる

このサイトではいろいろある中で zkSNARKsのGroth16 を使ったアプリを紹介

zkSNARKsを使った開発

まだ手順しか分からないので、意味を順次追加していく。



用語

Trusted Setup

証明キーと検証キーを生成するために必要な概念、またはプロセス全体の名称を指します

ceremony

Trusted Setupを実現するための具体的な「イベント」や「儀式」を指します

「Trusted Setup」がその目的と概念を示すのに対し、「ceremony」はそれを実行する具体的な方法やイベントを示す、というニュアンスの違い

Powers of Tau

Trusted Setupで行う2つのフェーズのうち、最初に行う普遍的なセットアップの方。

Groth16 の phase 1 で使用される(他は調べていない)

目的は、証明と検証の両方に使用される**共通参照文字列（Common Reference String: CRS）**の一部を生成することです。このプロセスは、悪意のある情報（「毒」）が混入しないことを保証するために、多者間計算（MPC）として行われます。各参加者は自身のランダムネスを貢献し、そのランダムネスに関連する一時的な秘密情報を破棄します。

完了すると出力として「証明キー(Proving Key, PK)」と「検証キー(Verification Key, VK)」が生成される。

Groth16の検証では VK だけ使う(それ以外にいるは公開入力された値と、証明データ)

phase 1

普遍的なセットアップを指す。主なアプローチとしてPowers of Tau(Groth16など)や SNARKs with Universal and Updateable Setup(Sonic, Plonk, Marlin, Haloなど)などがある。

phase 2

回路固有のセットアップ

Groth16では Trusted Setupの一部だが Plonk では信頼性の要件の違いから入らないそうだ

snarkjsの場合

snarkjs は Groth16 に特化しているので、phase 1 = Powers of Tau なのでドキュメントでは Powers of Tau と書いている

Gemini調査

ZKP回路のコンパイルは Phase 2 までにやっておけばよい

Circomファイルのコンパイルはどの時点でやっても良い(Phase 1 の作業とは関係ないため)

ここでは、R1CS(constraint system)、WASM(WITNESS生成用)、SYM(デバッグ用シンボル)を生成する

`snarkjs r1cs info [R1CS file]` で情報を見ることができる

Phase 1 では ZKP回路は使わない

Ceremony を開始し参加(contribute)してもらう

zkSNARKsを使ってproofを生成する際に必要となる最初のパラメーターを生成するプロセスのこと

Ceremony への contribute(phase 1)

不特定多数の参加者が生成する秘密情報をリレー形式に繋いでZKPに必要なパラメーターを生成する作業

ここでは"powers of tau"という名前の Ceremony を開始する

❌Ceremonyの各参加者が利用した秘密情報が漏洩してしまうと、虚偽のproofが生成できてしまう

参加者は多い方がよい。事前に人数を決めるものではなさそうだ。

今回は `snarkjs` でやっているが、サードパーティーのソフトウェアを使っても contributeできるそうだ

順番に `.ptau` ファイルを回していく。継ぎ足されていく感じか？

contribute し終わった `.ptau` ファイルにランダム beaconを反映させる

Phase 2への準備、という作業がまあまあ時間がかかる

Phase 2 は ZKP回路に依存した作業

セットアップ

コンパイルした `.r1cs` と Phase 1 で準備した `.ptau` から最初の `.zkey` ファイルを作る

ここでは groth16 という方式？を使っている

重要 この `.zkey` はまだ contribute されておらず安全ではないので使用しな

いこと！

Ceremony への contribute(phase 2)

phase 1 と同じように contribute してもらう。

順番に `.zkey` ファイルを回していく。

phase 1 と違う人数でよいが最低でも 1回はしておく

contribute しおわった `.zkey` ファイルにランダム beaconを反映させる

verification keyをエクスポートする

Proof の作成

ZKP回路をコンパイルした時の WASM ファイルと、回路の検証に使う入力ファイル(`input.json`)を与えると `.wtns` ファイルが作られる

`.wtns` ファイルと Phase 2 の最後の `.zkey` から `proof.json` と `public.json` が作られる

`proof.json` : 実際のproofデータ

`public.json` : publicな入力データ

Proof を検証する Solidity のコードを生成

Proof の検証はコマンドラインからもできる

引数に `verification_key.json` 、 `public.json` 、 `proof.json` の3ファイルを与えている

開発

ツールは iden3 · GitHub というアカウントにあるリポジトリから取得する

circom

回路を作る言語であるCircom言語のコンパイラ(CIRcuit COMpiler)

クラスNP→SAT→CircuitSAT、の回路だろう

snarkjs

SNARKs

今回は Raspberry Pi4 上で動かした

ときどき時間を計っているので、それは目安程度に見てほしい。

https://github.com/hirokuma/zkp-tutorial1

ここではパスワードの検証(パスワードのハッシュ値を比較)

このページでは最初にcircuitが出てくるが、Trusted Setup の Phase 1 には circuit は関係しない(はず)

"248" ってなんの数値だろう？

後で出てくる `input.json` のデータ長が 31バイト=248bit だった

32バイトのデータではダメ？

コンパイルした結果は Trusted Setup の Phase 2で使う

`compile`

```
$ npm i snarkjs circomlib
$ circom passwd-circuit.circom --r1cs --wasm --sym
template instances: 13
non-linear constraints: 3274
linear constraints: 129
public inputs: 1
private inputs: 1
public outputs: 0
wires: 3404
labels: 7878
Written successfully: ./passwd-circuit.r1cs
Written successfully: ./passwd-circuit.sym
Written successfully: ./passwd-circuit_js/passwd-circuit.wasm
Everything went okay
```

Trusted Setup

ここでは tornado cash と同じやり方をするようだ

zkSnark の groth16 では 2フェーズに分かれている

phase 1 (Powers of Tau Ceremonyの開始とContribute)

新規作成

snarkjs

```
$ npx snarkjs powersoftau new bn128 14 pot14_0000.ptau -v
[DEBUG] snarkJS: Calculating First Challenge Hash
[DEBUG] snarkJS: Calculate Initial Hash: tauG1
[DEBUG] snarkJS: Calculate Initial Hash: tauG2
[DEBUG] snarkJS: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkJS: Calculate Initial Hash: betaTauG1
[DEBUG] snarkJS: Blank Contribution Hash:
                786a02f7 42015903 c6c6fd85 2552d272
                912f4740 e1584761 8a86e217 f71f5419
                d25e1031 afee5853 13896444 934eb04b
                903a685b 1448b755 d56f701a fe9be2ce
[INFO]  snarkJS: First Contribution Hash:
                bc0bde79 80381fa6 42b20975 91dd83f1
                ed15b003 e15c3552 0af32c95 eb519149
                2a6f3175 215635cf c10e6098 e2c612d0
                ca84f1a9 f90b5333 560c8af5 9b9209f4
```

Ceremonyへの参加者(contribute)は 3名

1回につき1ファイル作られている

新しく作られたファイルを次の人のinputに使っているように見えるので同

時に依頼したりはできなさそう

Raspberry Pi4 で2分くらいかかっているので、通常のPCでも多少時間がかかりそうだ

この人は `--name` を全部同じにしているが、たぶん通常は違う名前にする

contribute

```
$ npx snarkjs powersoftau contribute pot14_0000.ptau pot14_0001.ptau
--name="First contribution" -v -e="random text 1"
[DEBUG] snarkJS: Calculating First Challenge Hash
[DEBUG] snarkJS: Calculate Initial Hash: tauG1
[DEBUG] snarkJS: Calculate Initial Hash: tauG2
[DEBUG] snarkJS: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkJS: Calculate Initial Hash: betaTauG1
[DEBUG] snarkJS: processing: tauG1: 0/32767
[DEBUG] snarkJS: processing: tauG1: 16384/32767
[DEBUG] snarkJS: processing: tauG2: 0/16384
[DEBUG] snarkJS: processing: tauG2: 8192/16384
[DEBUG] snarkJS: processing: alphaTauG1: 0/16384
[DEBUG] snarkJS: processing: betaTauG1: 0/16384
[DEBUG] snarkJS: processing: betaTauG2: 0/1
[INFO]  snarkJS: Contribution Response Hash imported:
                 8775a3b8 aca5fd99 3fbf53eb 743982a5
                 25aa53cf c1cf5dcd eaa3b895 34e66d93
                 7d272715 8e6cc9a2 170037b0 b7740b19
                 eb4c3519 fdc91a08 de408037 84440280
[INFO]  snarkJS: Next Challenge Hash:
                 1bf169c7 02bd0b2c 3f7e13bc 0844dcab
                 8efbc21c e1e8e68d 9d4fe9d3 59c56d53
                 b3b40feb 3de7b58c aba70e0c 0c7f8028
                 cd7072a9 71a67ba9 4f9be2b3 318b3e7f
$ npx snarkjs powersoftau contribute pot14_0001.ptau pot14_0002.ptau
--name="First contribution" -v -e="random text 2"
[DEBUG] snarkJS: processing: tauG1: 0/32767
[DEBUG] snarkJS: processing: tauG1: 16384/32767
[DEBUG] snarkJS: processing: tauG2: 0/16384
[DEBUG] snarkJS: processing: tauG2: 8192/16384
[DEBUG] snarkJS: processing: alphaTauG1: 0/16384
[DEBUG] snarkJS: processing: betaTauG1: 0/16384
[DEBUG] snarkJS: processing: betaTauG2: 0/1
[INFO]  snarkJS: Contribution Response Hash imported:
```

```
                0ffa093e 3f3cbfce 4f72885f c2d25e12
                efa2dd6a a3ad0012 4b0e7424 3c359b16
                7eca6865 11b9952d 6b020e49 0c1ccb05
                fbe4ba74 2e3bdf5a ccb6a37d 8d067d9c
[INFO]  snarkJS: Next Challenge Hash:
                9b838954 4da96241 462bb8a7 c58c39fa
                7b44cea2 1d435080 3080ae04 04411a2a
                9a278706 347224fe b9272b9c 889814aa
                38aefacd 03af4f56 68dcdb84 d230d3ef
$ time npx snarkjs powersoftau contribute pot14_0002.ptau
pot14_0003.ptau --name="First contribution" -v -e="random text 3"
[DEBUG] snarkJS: processing: tauG1: 0/32767
[DEBUG] snarkJS: processing: tauG1: 16384/32767
[DEBUG] snarkJS: processing: tauG2: 0/16384
[DEBUG] snarkJS: processing: tauG2: 8192/16384
[DEBUG] snarkJS: processing: alphaTauG1: 0/16384
[DEBUG] snarkJS: processing: betaTauG1: 0/16384
[DEBUG] snarkJS: processing: betaTauG2: 0/1
[INFO]  snarkJS: Contribution Response Hash imported:
                08db01df ff7785b9 b6fa0425 15685e6e
                834db04b 8953cf4c 2ef78ffe fb47f111
                807974c4 b290cdc7 214ea576 cf1a02d3
                e7831eb3 29cd1714 0c5db11d 4a8bf988
[INFO]  snarkJS: Next Challenge Hash:
                9a179f7f 8b31cadb 65bd8463 cff1fa6a
                d703aa83 b96c971b 3372816e ff453e80
                9b78a5e1 29de09b6 0b1bb03a 04fdcebd
                a70c51ed 1ec8d2ec 9f8ec691 92c612b0

real    0m57.537s
user    2m42.555s
sys     0m0.623s
```

ランダムBeaconの反映

`ptau` ファイルが生成される

beacon

```
$ npx snarkjs powersoftau beacon pot14_0003.ptau pot14_beacon.ptau
0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f 10 -
n="Final Beacon"
[INFO]  snarkJS: Contribution Response Hash imported:
```

```
                b311f11a 6058671c ae42006d 801cf1e7
                5d5eb951 c8b28bb9 71ec5d27 353b4feb
                64abffd8 02a6d3f2 b7b076f0 28c3d39c
                b3af427c 69254c91 8f8076eb 7f636050
[INFO]  snarkJS: Next Challenge Hash:
                360f9fa9 0ae9aa89 829b8a32 ae2916a2
                b10be28f e475f508 4d38ee3b 0117e05b
                cbb86235 07f985ba 13f8972e 598cda15
                2886710e 96b3de7b 4240c1f3 05143991
```

phase2の準備。時間がかかる。

snarkjs

```
$ time npx snarkjs powersoftau prepare phase2 pot14_beacon.ptau
pot14_final.ptau -v
[DEBUG] snarkJS: Starting section: tauG1
[DEBUG] snarkJS: tauG1: fft 0 mix start: 0/1
[DEBUG] snarkJS: tauG1: fft 0 mix end: 0/1
.....
.....
[DEBUG] snarkJS: betaTauG1: fft  14  join: 14/14
[DEBUG] snarkJS: betaTauG1: fft 14 join  14/14  1/1 0/2
[DEBUG] snarkJS: betaTauG1: fft 14 join  14/14  1/1 1/2

real    15m54.092s
user    40m35.704s
sys     0m1.678s

$ ls -l
total 50440
drwxr-xr-x 52 xxxx xxxx     4096 May  7 16:25 node_modules
-rw-r--r--  1 xxxx xxxx    20794 May  7 16:25 package-lock.json
-rw-r--r--  1 xxxx xxxx      293 May  7 16:25 package.json
-rw-r--r--  1 xxxx xxxx      711 May  7 16:24 passwd-
circuit.circom
-rw-r--r--  1 xxxx xxxx   706820 May  7 16:27 passwd-circuit.r1cs
-rw-r--r--  1 xxxx xxxx   530948 May  7 16:27 passwd-circuit.sym
drwxr-xr-x  2 xxxx xxxx     4096 May  7 16:27 passwd-circuit_js
-rw-r--r--  1 xxxx xxxx  6291664 May  7 16:58 pot14_0000.ptau
-rw-r--r--  1 xxxx xxxx  6293188 May  7 17:00 pot14_0001.ptau
-rw-r--r--  1 xxxx xxxx  6294712 May  7 17:01 pot14_0002.ptau
```

```
-rw-r--r--  1 xxxx xxxx  6296236 May  7 17:02 pot14_0003.ptau
-rw-r--r--  1 xxxx xxxx  6297789 May  7 17:13 pot14_beacon.ptau
-rw-r--r--  1 xxxx xxxx 18880429 May  7 17:31 pot14_final.ptau
```

検証？

ここでは手順になかったがsnarkjs guideにはこの手順がある

確認だけなので省略したのかもしれない

2回実行して出力が一致したのでランダム値を使ったものではないようだ

snarkjs

```
$ npx snarkjs powersoftau verify pot14_final.ptau
[INFO]  snarkJS: Powers Of tau file OK!
[INFO]  snarkJS: Next challenge hash:
                360f9fa9 0ae9aa89 829b8a32 ae2916a2
                b10be28f e475f508 4d38ee3b 0117e05b
                cbb86235 07f985ba 13f8972e 598cda15
                2886710e 96b3de7b 4240c1f3 05143991
[INFO]  snarkJS:
---------------------------------------------------
[INFO]  snarkJS: Contribution #4: Final Beacon
[INFO]  snarkJS: Next Challenge:
                360f9fa9 0ae9aa89 829b8a32 ae2916a2
                b10be28f e475f508 4d38ee3b 0117e05b
                cbb86235 07f985ba 13f8972e 598cda15
                2886710e 96b3de7b 4240c1f3 05143991
[INFO]  snarkJS: Response Hash:
                b311f11a 6058671c ae42006d 801cf1e7
                5d5eb951 c8b28bb9 71ec5d27 353b4feb
                64abffd8 02a6d3f2 b7b076f0 28c3d39c
                b3af427c 69254c91 8f8076eb 7f636050
[INFO]  snarkJS: Response Hash:
                9a179f7f 8b31cadb 65bd8463 cff1fa6a
                d703aa83 b96c971b 3372816e ff453e80
                9b78a5e1 29de09b6 0b1bb03a 04fdcebd
                a70c51ed 1ec8d2ec 9f8ec691 92c612b0
[INFO]  snarkJS: Beacon generator:
0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
[INFO]  snarkJS: Beacon iterations Exp: 10
[INFO]  snarkJS: Powers Of tau file OK!
[INFO]  snarkJS:
---------------------------------------------------
```

```
[INFO]  snarkJS: Contribution #3: First contribution
[INFO]  snarkJS: Next Challenge:
                9a179f7f 8b31cadb 65bd8463 cff1fa6a
                d703aa83 b96c971b 3372816e ff453e80
                9b78a5e1 29de09b6 0b1bb03a 04fdcebd
                a70c51ed 1ec8d2ec 9f8ec691 92c612b0
[INFO]  snarkJS: Response Hash:
                08db01df ff7785b9 b6fa0425 15685e6e
                834db04b 8953cf4c 2ef78ffe fb47f111
                807974c4 b290cdc7 214ea576 cf1a02d3
                e7831eb3 29cd1714 0c5db11d 4a8bf988
[INFO]  snarkJS: Response Hash:
                9b838954 4da96241 462bb8a7 c58c39fa
                7b44cea2 1d435080 3080ae04 04411a2a
                9a278706 347224fe b9272b9c 889814aa
                38aefacd 03af4f56 68dcdb84 d230d3ef
[INFO]  snarkJS: Powers Of tau file OK!
[INFO]  snarkJS:
---------------------------------------------------
[INFO]  snarkJS: Contribution #2: First contribution
[INFO]  snarkJS: Next Challenge:
                9b838954 4da96241 462bb8a7 c58c39fa
                7b44cea2 1d435080 3080ae04 04411a2a
                9a278706 347224fe b9272b9c 889814aa
                38aefacd 03af4f56 68dcdb84 d230d3ef
[INFO]  snarkJS: Response Hash:
                0ffa093e 3f3cbfce 4f72885f c2d25e12
                efa2dd6a a3ad0012 4b0e7424 3c359b16
                7eca6865 11b9952d 6b020e49 0c1ccb05
                fbe4ba74 2e3bdf5a ccb6a37d 8d067d9c
[INFO]  snarkJS: Response Hash:
                1bf169c7 02bd0b2c 3f7e13bc 0844dcab
                8efbc21c e1e8e68d 9d4fe9d3 59c56d53
                b3b40feb 3de7b58c aba70e0c 0c7f8028
                cd7072a9 71a67ba9 4f9be2b3 318b3e7f
[INFO]  snarkJS: Powers Of tau file OK!
[INFO]  snarkJS:
---------------------------------------------------
[INFO]  snarkJS: Contribution #1: First contribution
```

```
[INFO]  snarkJS: Next Challenge:
                1bf169c7 02bd0b2c 3f7e13bc 0844dcab
                8efbc21c e1e8e68d 9d4fe9d3 59c56d53
                b3b40feb 3de7b58c aba70e0c 0c7f8028
                cd7072a9 71a67ba9 4f9be2b3 318b3e7f
[INFO]  snarkJS: Response Hash:
                8775a3b8 aca5fd99 3fbf53eb 743982a5
                25aa53cf c1cf5dcd eaa3b895 34e66d93
                7d272715 8e6cc9a2 170037b0 b7740b19
                eb4c3519 fdc91a08 de408037 84440280
[INFO]  snarkJS: Response Hash:
                bc0bde79 80381fa6 42b20975 91dd83f1
                ed15b003 e15c3552 0af32c95 eb519149
                2a6f3175 215635cf c10e6098 e2c612d0
                ca84f1a9 f90b5333 560c8af5 9b9209f4
[INFO]  snarkJS:
-------------------------------------------------
[INFO]  snarkJS: Powers of Tau Ok!
```

phase 2

回路をコンパイルした結果を使う。

こちらはそんなに時間がかからない。

snarkjs

```
$ npx snarkjs groth16 setup passwd-circuit.r1cs pot14_final.ptau
circuit_0000.zkey
[INFO]  snarkJS: Reading r1cs
[INFO]  snarkJS: Reading tauG1
[INFO]  snarkJS: Reading tauG2
[INFO]  snarkJS: Reading alphatauG1
[INFO]  snarkJS: Reading betatauG1
[INFO]  snarkJS: Circuit hash:
                157e0861 d5ebc836 76edfab7 e0ce7344
                a9e22da8 02ede2e3 c75d3fc8 48074a0f
                d53db4f4 466defde 508ba58a 505fdb17
                5a587390 0097ce32 fdd340e1 e817f3da
$ npx snarkjs zkey contribute circuit_0000.zkey circuit_0001.zkey --
name="1st Contributor Name" -v -e="random text 1"
[DEBUG] snarkJS: Applying key: L Section: 0/3402
[DEBUG] snarkJS: Applying key: H Section: 0/4096
[INFO]  snarkJS: Circuit Hash:
```

```
                157e0861 d5ebc836 76edfab7 e0ce7344
                a9e22da8 02ede2e3 c75d3fc8 48074a0f
                d53db4f4 466defde 508ba58a 505fdb17
                5a587390 0097ce32 fdd340e1 e817f3da
[INFO]  snarkJS: Contribution Hash:
                778919d2 31cc355f 273839c4 3e2eb02c
                0cb66a22 6757e9ff ef28ad73 7b12e965
                cba35870 096db761 d9d4d31e 3ebe872d
                ef63ca2e c38543c6 3f6088b4 552327e4
$ npx snarkjs zkey beacon circuit_0001.zkey circuit_final.zkey
0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f 10 -
n="Final Beacon phase2"
[INFO]  snarkJS: Contribution Hash:
                a42f73ce 3eec7053 c42b7aba 8cf30c6b
                c29238fa 2bb67785 a3888e4a f7040027
                84d1744d 15b7f7f0 ab9eb8f2 9ce70c4e
                53e8c414 fe1b1a70 0edb73da 815b8169
$ npx snarkjs zkey export verificationkey circuit_final.zkey
verification_key.json
[INFO]  snarkJS: EXPORT VERIFICATION KEY STARTED
[INFO]  snarkJS: > Detected protocol: groth16
[INFO]  snarkJS: EXPORT VERIFICATION KEY FINISHED
$ ls -l
total 55760
-rw-r--r--  1 xxxx xxxx  1812268 May  7 17:53 circuit_0000.zkey
-rw-r--r--  1 xxxx xxxx  1812682 May  7 17:55 circuit_0001.zkey
-rw-r--r--  1 xxxx xxxx  1813130 May  7 17:57 circuit_final.zkey
drwxr-xr-x 52 xxxx xxxx     4096 May  7 16:25 node_modules
-rw-r--r--  1 xxxx xxxx    20794 May  7 16:25 package-lock.json
-rw-r--r--  1 xxxx xxxx      293 May  7 16:25 package.json
-rw-r--r--  1 xxxx xxxx      711 May  7 16:24 passwd-circuit.circom
-rw-r--r--  1 xxxx xxxx   706820 May  7 16:27 passwd-circuit.r1cs
-rw-r--r--  1 xxxx xxxx   530948 May  7 16:27 passwd-circuit.sym
drwxr-xr-x  2 xxxx xxxx     4096 May  7 16:27 passwd-circuit_js
-rw-r--r--  1 xxxx xxxx  6291664 May  7 16:58 pot14_0000.ptau
-rw-r--r--  1 xxxx xxxx  6293188 May  7 17:00 pot14_0001.ptau
-rw-r--r--  1 xxxx xxxx  6294712 May  7 17:01 pot14_0002.ptau
-rw-r--r--  1 xxxx xxxx  6296236 May  7 17:02 pot14_0003.ptau
-rw-r--r--  1 xxxx xxxx  6297789 May  7 17:13 pot14_beacon.ptau
```

```
-rw-r--r--  1 xxxx xxxx 18880429 May  7 17:31 pot14_final.ptau
-rw-r--r--  1 xxxx xxxx     2928 May  7 17:58 verification_key.json
```

Proofの作成

おそらく circuit で使われている Pedersen Hash というハッシュ計算なのだろうがSHA256みたいにオンラインツールが見つからない

`input.json` の値は16進数**ではなく**10進数である

input.json

```
{
  "secretHash":
"6809011718265561644929127248771906476776948110202239571066532094237
689135794",
  "secret":
"2156396070657850491449436131561646905457091084277685133828327542851
84579360"
}
```

snarkjs

```
$ ls -l  passwd-circuit_js/
total 92
-rw-r--r-- 1 xxxx xxxx   697 May  7 16:27 generate_witness.js
-rw-r--r-- 1 xxxx xxxx 73825 May  7 16:27 passwd-circuit.wasm
-rw-r--r-- 1 xxxx xxxx 10356 May  7 16:27 witness_calculator.js
$ node passwd-circuit_js/generate_witness.js passwd-circuit_js/
passwd-circuit.wasm input.json witness.wtns
$ npx snarkjs groth16 prove circuit_final.zkey witness.wtns
proof.json public.json
$ ls -l
total 55880
-rw-r--r--  1 xxxx xxxx  1812268 May  7 17:53 circuit_0000.zkey
-rw-r--r--  1 xxxx xxxx  1812682 May  7 17:55 circuit_0001.zkey
-rw-r--r--  1 xxxx xxxx  1813130 May  7 17:57 circuit_final.zkey
-rw-r--r--  1 xxxx xxxx      189 May  7 18:04 input.json
drwxr-xr-x 52 xxxx xxxx     4096 May  7 16:25 node_modules
-rw-r--r--  1 xxxx xxxx    20794 May  7 16:25 package-lock.json
-rw-r--r--  1 xxxx xxxx      293 May  7 16:25 package.json
-rw-r--r--  1 xxxx xxxx      711 May  7 16:24 passwd-circuit.circom
-rw-r--r--  1 xxxx xxxx   706820 May  7 16:27 passwd-circuit.r1cs
-rw-r--r--  1 xxxx xxxx   530948 May  7 16:27 passwd-circuit.sym
drwxr-xr-x  2 xxxx xxxx     4096 May  7 16:27 passwd-circuit_js
```

```
-rw-r--r--  1 xxxx xxxx  6291664 May  7 16:58 pot14_0000.ptau
-rw-r--r--  1 xxxx xxxx  6293188 May  7 17:00 pot14_0001.ptau
-rw-r--r--  1 xxxx xxxx  6294712 May  7 17:01 pot14_0002.ptau
-rw-r--r--  1 xxxx xxxx  6296236 May  7 17:02 pot14_0003.ptau
-rw-r--r--  1 xxxx xxxx  6297789 May  7 17:13 pot14_beacon.ptau
-rw-r--r--  1 xxxx xxxx 18880429 May  7 17:31 pot14_final.ptau
-rw-r--r--  1 xxxx xxxx      804 May  7 20:00 proof.json
-rw-r--r--  1 xxxx xxxx       83 May  7 20:00 public.json
-rw-r--r--  1 xxxx xxxx     2928 May  7 17:58 verification_key.json
-rw-r--r--  1 xxxx xxxx   109004 May  7 19:58 witness.wtns
```

proof.json

proof.json


```
{
 "pi_a": [

"311433140286199543130606489796277504598347824371156805477444998238124059186 0",

"213436702952914375496320999149111025549723072847543660468046111636078987638 50",
  "1"
 ],
 "pi_b": [
  [

"920082163985580229998604005191420601162261944276435470880613388145498916717 2",

"129885281912872396279333693674334053495038247862666705294565502137943850883 7"
  ],
  [

"729406399501373248747868981526280934652840613467257169343810131578887681831 1",

"217047838507707238109715636787650434918751997269784711176778485351029036628 70"
  ],
```

```
      [
       "1",
       "0"
      ]
     ],
     "pi_c": [

    "490510867108517223181509073290131557244589348782329683993354432915213560512
    3",

    "125284869153150155655595521798585482748157593521614068642991237981679605434
    14",
      "1"
     ],
     "protocol": "groth16",
     "curve": "bn128"
    }
```

public.json

public.json

```
    [

    "680901171826556164492912724877190647677694811020223957106653209423768913579
    4"
    ]
```

Proofの検証

snarkjs

```
$ npx snarkjs groth16 verify verification_key.json public.json
proof.json
[INFO]  snarkJS: OK!
```

Solidity Verifierの作成

snarkjs

```
$ npx snarkjs zkey export solidityverifier circuit_final.zkey
verifier.sol
[INFO]  snarkJS: EXPORT VERIFICATION KEY STARTED
[INFO]  snarkJS: > Detected protocol: groth16
[INFO]  snarkJS: EXPORT VERIFICATION KEY FINISHED
```

150行くらいの sol ファイルが生成された。コントラクト名は `Groth16Verifier` 。

solのfunction引数

```
function verifyProof(
  uint[2] calldata _pA,
  uint[2][2] calldata _pB,
  uint[2] calldata _pC,
  uint[1] calldata _pubSignals
) public view returns (bool)
```

呼び出しsimulationもある

おそらく `proof.json` の値と一致している。

```
call
  [

  "0x06e2a62bf29f8bf9aa9af084cec5f8e50983cf33df6c419a4d51d4dcd31fa5
  f4",

  "0x2f3016db6019d42bdd53b6ff8ebb72d95cee25cfdecab039741075946943b2
  4a"
  ],
  [
    [

  "0x02df1fe8a7ced256a1336e0980de03d38348ded304ceb0d648605098cd5bfb
  25",

  "0x14577b022de74e1ab9d10159f8eda8146e798b6e42a567d9317e5bb38ffc4a
  44"
    ],
    [

  "0x2ffc78e996110fb63b99ba562099b9c7e2106e114cf89384cd289adf9d99b9
  16",

  "0x10204b3e568f6d2e772570d88a5bf98a72a4fb542890ca10e309d090fd5c27
  87"
    ]
  ],
  [

  "0x0ad83170869a62dcb5e8675ffa017f11f9a00a2371304da9a4da0a9a47c2ab
  83",
```

```
"0x1bb2df4f403d2af966321ae9c2c60599b20c15330dc3322dcd1eae443cede0
b6"
],
["0x0f0dc3997476856747e963ab300f642f51e52ad8550cf5f7f661b7a46e264
2b2"]
```

別の回路でやってみる。

https://docs.circom.io/getting-started/writing-circuits/

ここの `A*B + C = 0` が `c <== a * b` という制約(constraints)になるのが納得いかないからだ。

`A * B + C = 0` なんだから `A * B = -C` になって `c <== -a * b` のような制約になるのでは？

circomドキュメントの誤記を疑っているのだ。

https://github.com/hirokuma/zkp-tutorial2

・・・正しかった、というか、circom の制約は自分で書くので、 `c == a * b` になるように入力データを書いて動いたし、そもそも `A*B + C = 0` が `c <== a * b` になるという説明文ではなく、形式として `A*B + C = 0` だという話をしたかっただけなのだろう。

誰も悪くなかった。

snarkjsのtutorialをやってみる

snarkjs/guide