Core Program Week 5

Circom実践
Tornado Cashで学ぶ

ZK Tokyo

Week 5 で学ぶこと

Week5 では学んだことを具体的に実装することが主な目的です。

Tornado Catsを教材にして、Tornado Cashを扱います。

目次

Circomのセットアップ

開発環境セットアップ

こちらのレポジトリを参照して環境をセットアップしましょう。
[core-program/circom at feature/circom-basics · zk-tokyo/core-program · GitHub]

プライバシー保護と送金ミキシングサービス

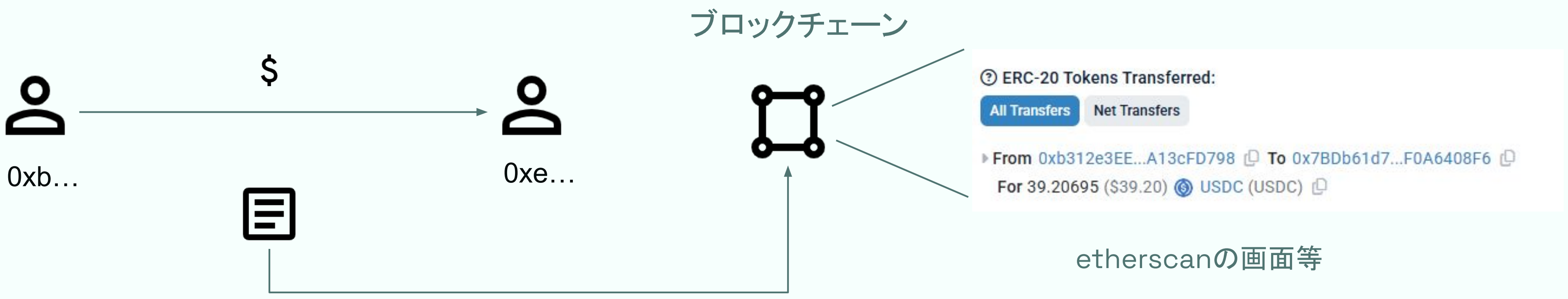
ブロックチェーン上での決済

Ethereumなどパブリックブロックチェーンでは取引内容は公開される。

取引内容の証明として機能はするものの、全てを公開したいとは限らない。

例えばB2Bや高額決済では決済情報を関係者外に公開することは好ましくない。

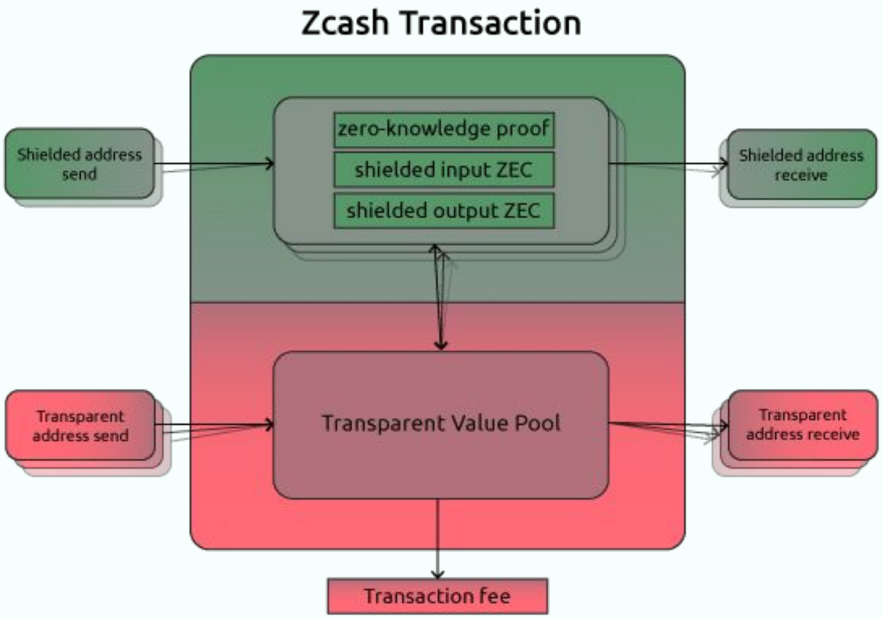
プライバシー保護と情報の透明性を担保することは大きな課題。



取引情報はブロックチェーンに書き込まれる

ZCash

2016年にBitcoinをベースにプライバシー機能を拡張してzkSNARKSを組み込んだ新しい暗号資産として開発。

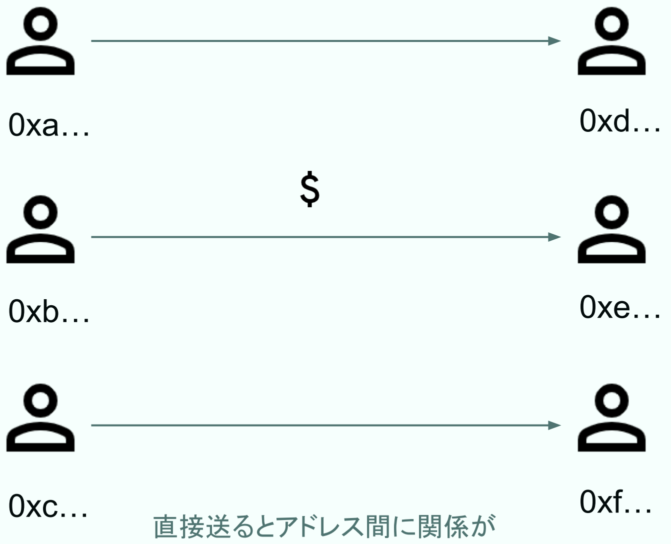


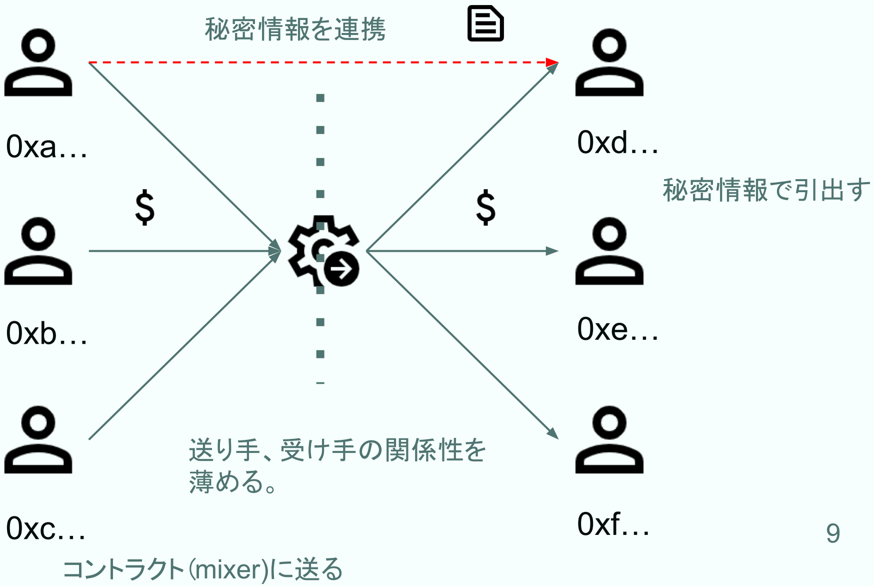
Tornado Cash （⇒このWeekのテーマ）

資金の送り手はミキシングサービスに資金を預け、引出に必要となる情報を別途受け手に渡す。

一定数の同じような送金を混ぜ合わせることで、引き出した資金が誰のものかわからなくなる。

多数の取引送金が混ざるため（mix）、mixingしたアセットの所有を証明するためにzkを利用しています。





Privacy Pool

2023年にVitalikらによって提案され、プライバシー保護とコンプライアンス遵守を実現することが目的。

ミキサーを利用する点はTornado Cashらと同じ仕組みであるが、引出す際に「自分は正当な参加者グループに属している」ことを証明できる。

グループは、例えば既知のKYC済アドレスやホワイトリストなど、合法性を証明可能な集合として設計可能であり、AML対応などへの活用が期待される。

Blockchain Privacy and Regulatory Compliance: Towards a Practical Equilibrium

Vitalik Buterin*, Jacob Illum†, Matthias Nadler‡, Fabian Schär‡, Ameen Soleimani§
*Ethereum Foundation, †Chainalysis, ‡University of Basel, §Privacy Pools

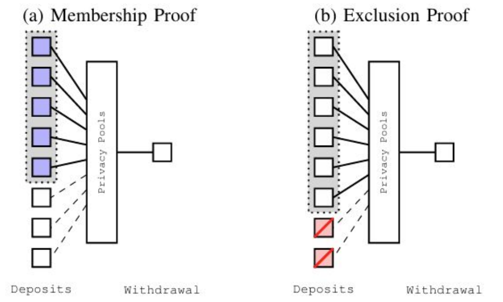


Fig. 5: The membership proof includes a specific collection of deposits in its association set while the exclusion proof's association set consists of anything but a specific collection of deposits. From a technical perspective they are identical, as they both prove against the Merkle root of an association set.

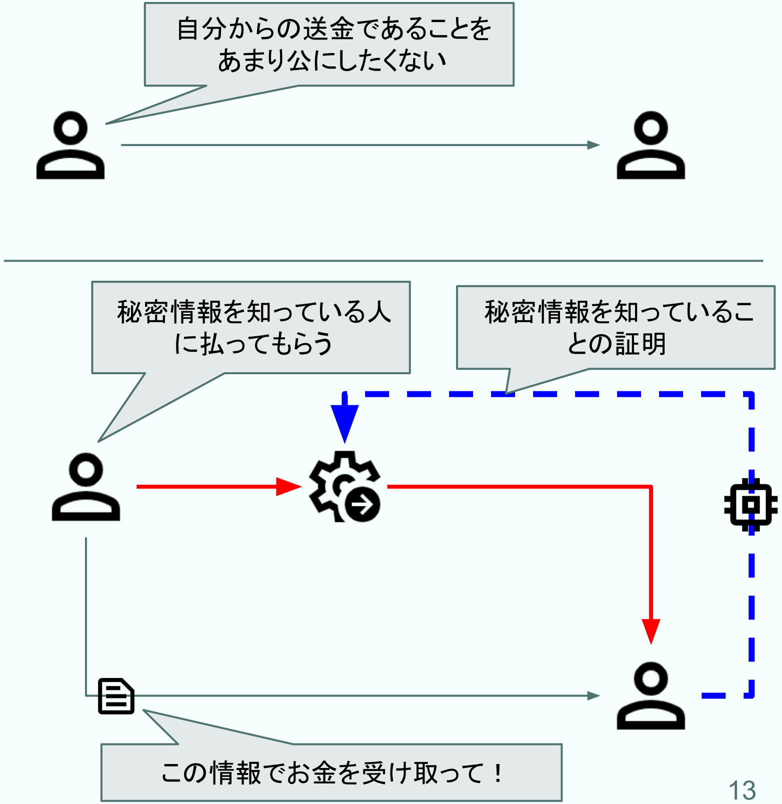
比較表

時期	技術	主な特徴	解決した課題	残された課題
2016年〜	ZCash	zk-SNARKで完全匿名	ブロックチェーンでの強固なプライバシー	専用チェーン・匿名性集合が小さい
2019年〜	Tornado Cash	Ethereumでミキサー型匿名化	Ethereum互換性・手軽な匿名送金	マネロン・規制リスク
2023年〜	Privacy Pools	証明可能な選択的匿名性	コンプライアンス対応・合法性証明	実利用（参加者グループ基準の決定）

Tornado Cash

前提の整理

入金者	資金を送るために入金する人
出金者	資金を引き出す人
ミキサー	スマートコントラクト
リレイヤー	出金者に代わって出金取引を実施
Merkle Tree	検証用のハッシュを格納 Tornado Cashの場合は高さ20 本実装ではSMTを利用
秘密情報	コミットメントを行うためのデータ
ハッシュテーブル	二重使用を避けるために使用したハッシュ値を登録するなどに利用



全体の流れ

1. 預け入れ(デポジット)
    - 乱数(秘密の値)を自分で生成し、それを元にハッシュ値(コミットメント)を作成し、コントラクトに送金＆登録する
2. コントラクト内部管理
    - コントラクトは送られたハッシュ値を「 Merkleツリー」というデータ構造で管理する
3. 引き出し要求
    - 別のアドレスから、元の秘密の値に基づいて「自分がツリー内のどこかに登録されていること」をzk-SNARK証明して引き出しを要求する
4. コントラクト検証
    - zk-SNARKの証明を検証して、正しければ新しいアドレスに資金を送金する(ただし、誰のデポジットかはわからない)

スマートコントラクトの機能（参考教材ベース）

資金管理

Tree管理

証明検証

$\mathcal{T}$

$\mathcal{H}_{\text{commitment}}$

$\mathcal{H}_{\text{nullifier}}$

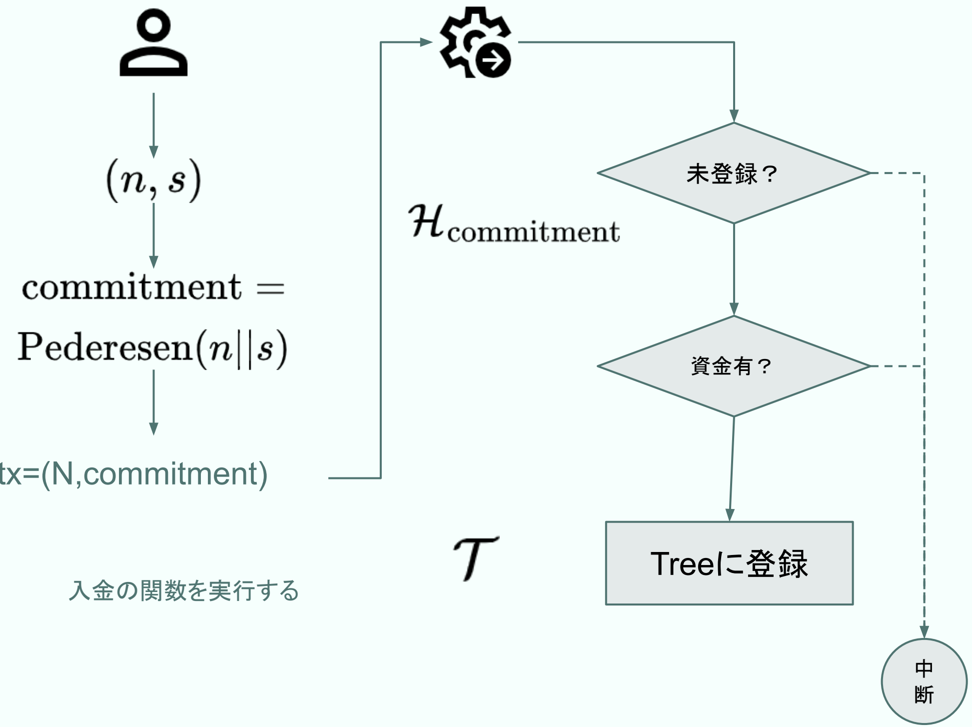
入金された資金はコントラクトアドレスで管理。

コミットメント値はMerkle Treeで管理

証明検証は「コミットメントのハッシュ値と、Merkletreeでの位置（leafの位置とルートまでの経路（オープニング）を知っている」ことを確認する。

出金に使われたハッシュ値（コミットメント）を別途管理して二重払いを避ける。

入金フローの図解



資金を送る準備;

Secret, nullifierを$\mathbb{Z}_p$から取得

PerdersenHash関数でコミットメントを計算して、トラン ザクションに必要な資金を含めてスマートコントラクトで預け入れを実行する。

コミットメントの値の登録有無や必要資金が含まれていることをチェックして、問題がなければ受け入れられる。

コミットメント値は MerkleTreeに登録される。

$$\mathbb{Z}_p$$

$$\text{commitment} = \text{Pederesen}(n||s)$$

(参考)トランザクションの中身

項目	内容
to	スマートコントラクトのアドレス
value	入金額(入金の場合)
data	関数呼出しのABIエンコード
gas	関数実行に必要なガス代
その他	署名情報やnonceなど

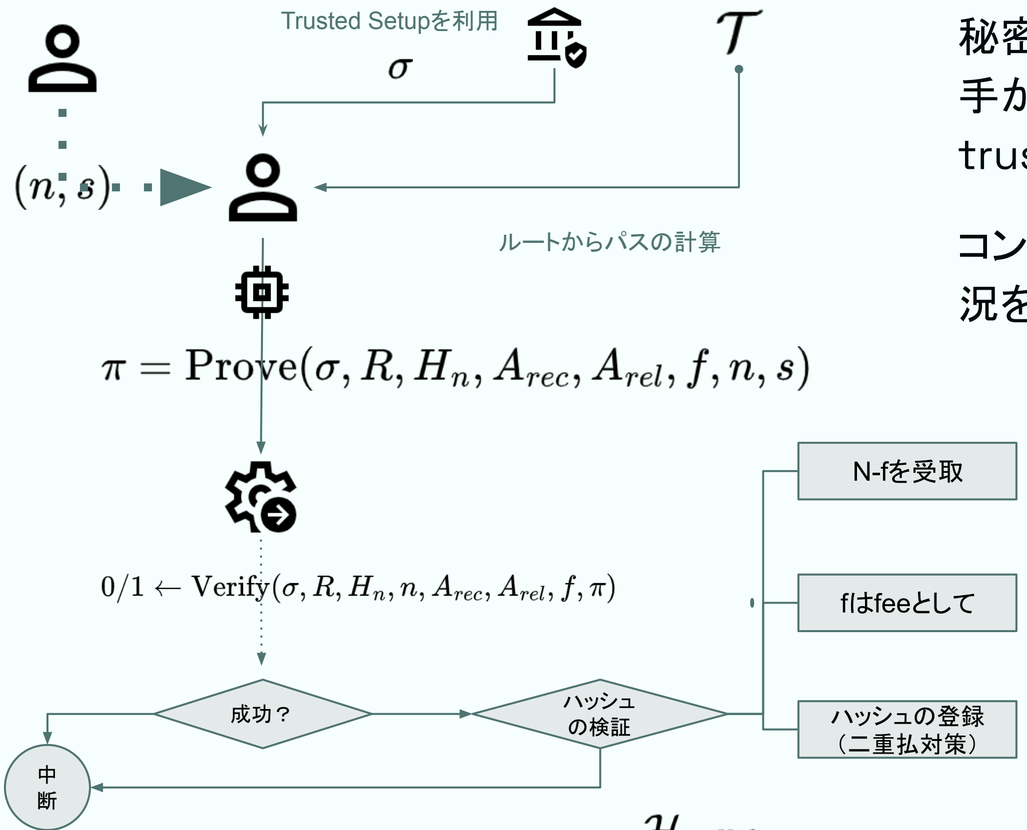
スマートコントラクトはブロックチェーン上で実行されるため実行に必要なトランザクションが必要となる。

関数呼び出しは、コード上ではdeposit(byte32 commitment)のようになるが、中身としてはABIエンコードでバイト化されている必要がある。

Valueは入金のみに利用を想定でトランザクションとともに資金を送る。

その他の情報としてトランザクションに署名するなど登録に必要な情報を入れる。

証明・検証フローの図解



秘密情報であるk,rやTreeに関する情報は送り手から共有され、証明・検証用の鍵ペアはtrusted setupから取得する。

コントラクトでは証明の検証とnullifierの使用状況を確認し、資金の引出しを実行する。

【参考】Sparse Merkle Treeについて

Merkle Treeは木構造のデータの持ち方で、データはハッシュ化されて格納されます。データの存在を効率的に証明できる。

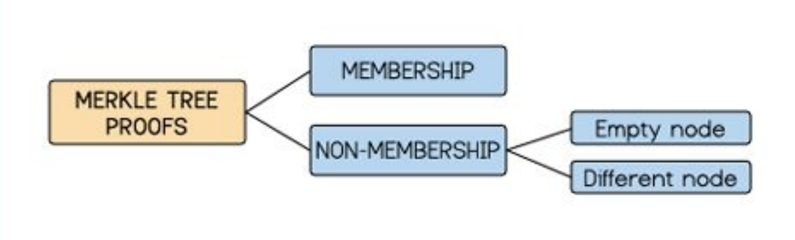
Sparse Merkle Treeでは、登録されるデータにはインデックスが与えられており、インデックスに対応する葉に対応するデータを格納する。

①大きな固定サイズのツリーを用意、②葉にデフォルト値を対応させ、インデックスに対応する葉の値がデフォルト値であることが"不存在"の証明になる。

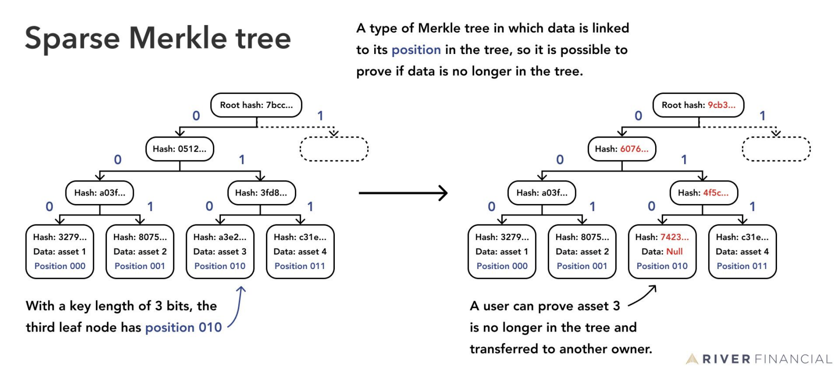
サンプルプログラムでの工夫

Tornado CashではMerkle Treeとハッシュ値のテーブルを利用しています。

実装例（リンク）ではSMT(Sparse Merkle Tree)を利用して包含証明と非包含証明を行い、検証に利用する回路を共通で利用するなど効率化が可能になります。



Merkle-Tree.pdf



Sparse Merkle Tree | River

QUIZ session（pending）

[core-program/week5_codes/trnadocashlike at feature/trnadocashlike · zk-tokyo/core-program · GitHub]

とTornado Catsの実装演習に関して議論できる内容を作る

Codeの実装

【発展】Poseidon Hash

開発経緯

2018年の秋,STARK論文に触発され研究を開始
2019年, ブロック暗号HADESMiMC, ハッシュ関数Poseidonとその後, Buletproof, Groth 16, Stark, Plonk上で効率良く働くことを目指してPoseidonが調節されていった。

Poseidon Hash論文の4ページ, “Historic Remarks.”の項より

PoseidonがzkフレンドリーGな理由

結論:

算術回路を用いるzkSnarkにおいて，ゲートの数を少なく保つことができるから

⋆裏を返せばsumcheck protocolやGKR では他のハッシュ関数が適している

理由:

ビット演算を用いず代わりに行列や有限体上での演算のみを用いるため。

実際にどれほどゲート数が少なくなるのか

Table 4: Number of R1CS constraints for a circuit proving a leaf knowledge in the Merkle tree of 2^{30} elements.

Arity	Width	R_F	R_P	Total constraints
POSEIDON-128				
2:1	3	8	57	7290
4:1	5	8	60	4500
8:1	9	8	63	4050
Rescue-x^5				
2:1	3	16	-	8640
4:1	5	10	-	4500
8:1	9	10	-	5400
Pedersen hash				
510	171	-	-	41400
SHA-256				
510	171	-	-	826020
Blake2s				
510	171	-	-	630180
MiMC-$2p/p$ (Feistel)				
1:1	2	324	-	19440

深さ30のMerkle Treeに対する証明をR1CS形式で証明:

Sha265では82万6020ゲートが必要
対してPoseidon-128では数千ゲートに収まっている

Poseidon Hash 14ページ Table4より引用

Poseidon Hashのアルゴリズム

用語:(必要になり次第,随時ここを参照してください)

スポンジ: 有限体上の一次元配列, rateとcapacityという二つの部分に分かれている
Poseidon Hashの入力: スポンジのrate部分
Poseidon Hashの出力: スポンジのrateの初めの要素
吸収: スポンジのrate部分にPoseidonの入力をセットすること
搾り出: スポンジから必要なだけデータを読み出すこと

S-Box: スポンジの各要素に対して累乗をする関数,ただし有限体上の累乗であることに注意すること
Arc: Add Round Constantsの略。各要素に対してランダムな定数を加える
MixLayer:スポンジ全体対し, MDS行列を乗算する
ラウンド : スポンジに対するS-Box, Arc, MixLayer操作をまとめてラウンドと呼ぶ

補足:
S-Boxという言葉からわかる通りPoseidonはAES暗号から影響を受けている。
ややこしさを減らすためにポセイドンの場合に限定している。
スポンジがどのような配列であるかはハッシュ関数による(三次元配列のものもある)
搾り出しの処理は関数の出力長により変わるがスポンジから要素を読み出すことという大枠は変わらない。

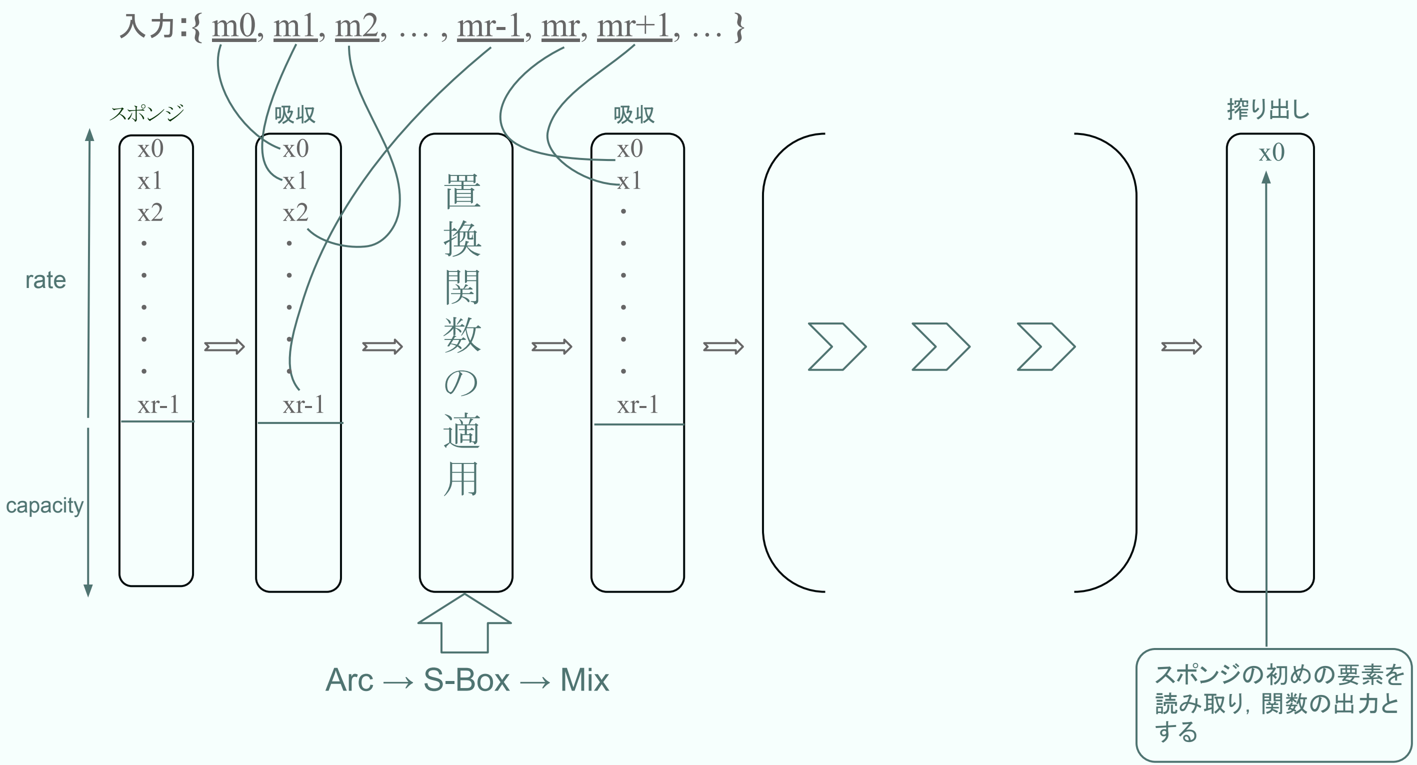
Poseidon Hashのアルゴリズム

アルゴリズムの概要
スポンジ構造とは:
吸収,置換,搾り出しという三つの部分により構成されるハッシュ関数の設計

1.吸収によりスポンジに入力を取り込む
2.スポンジに対して置換関数を繰り返し適用することで，スポンジ内部を攪拌する
3.最終的に必要な分だけスポンジから読み取り，それを出力結果とする

S-Box, Arc, MixLayerは置換関数を繰り返し適用する部分で用いられる操作

Poseidon Hashのアルゴリズム

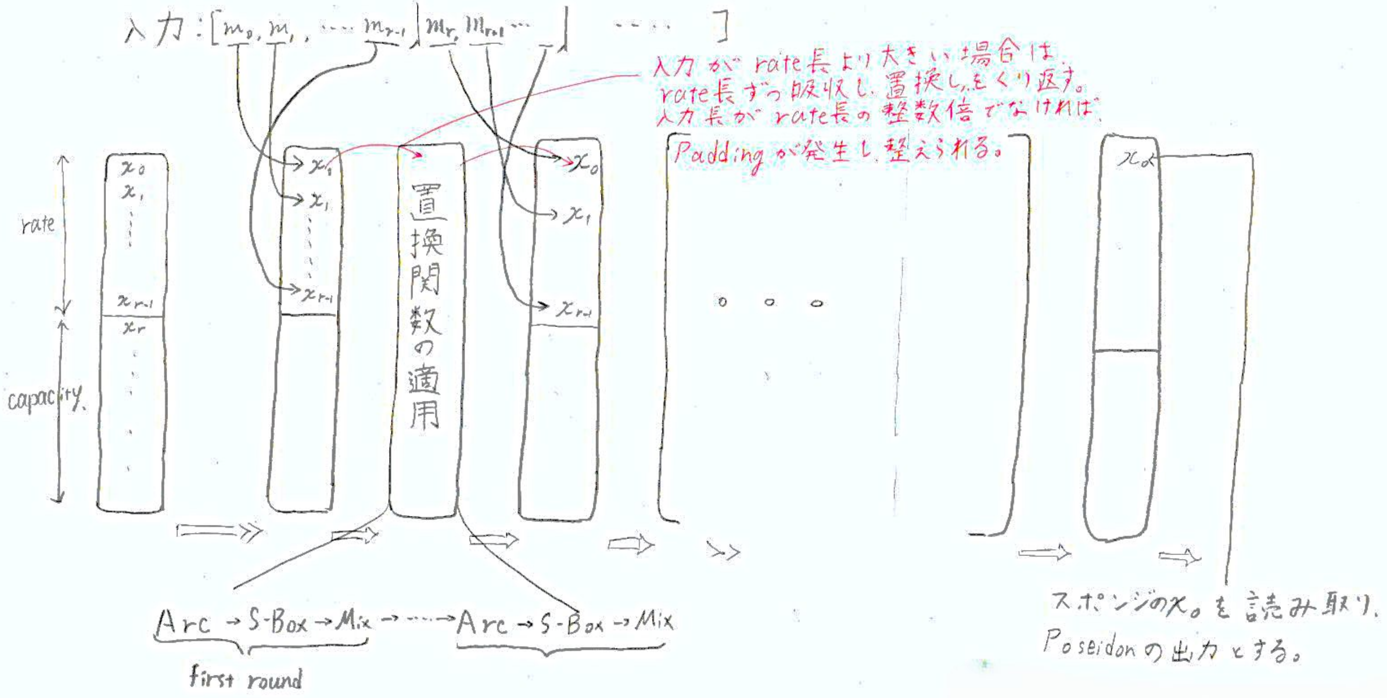


入力がrateの大きさ越えの時
rate分だけ吸収し，その後置換を適用する
残りをまた吸収し，残りがなくなるまで同じことを繰り返す
入力長がrateの整数倍でなければ,Paddingを施す

出力がx_iだけであることについて
あくまでPoseidonの場合は初めの要素のみが出力になる
他のスポンジ構造の関数では異なる

しかし，他の関数が扱っているのがビットであることに対して，Poseidonは有限体上の関数であり，かつzkSnarkはかなり大きな有限体上で運用される。
よって，一つの要素でも実際にはかなり長いビット列となる。

Poseidon Hashのアルゴリズム



出力がx_iだけであることについて
あくまでPoseidonの場合は初めの要素のみが出力になる
他のスポンジ構造の関数では異なる

しかし，他の関数が扱っているのがビットであることに対して，Poseidonは有限体上の関数であり，かつzkSnarkはかなり大きな有限体上で運用される。
よって，一つの要素でも実際にはかなり長いビット列となる。

ZK Tokyo

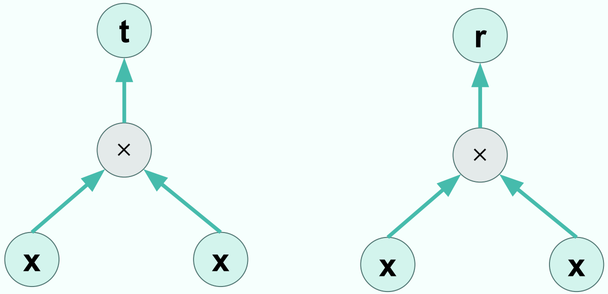
R1CSでのPoseidon：S-Box

S-Box：

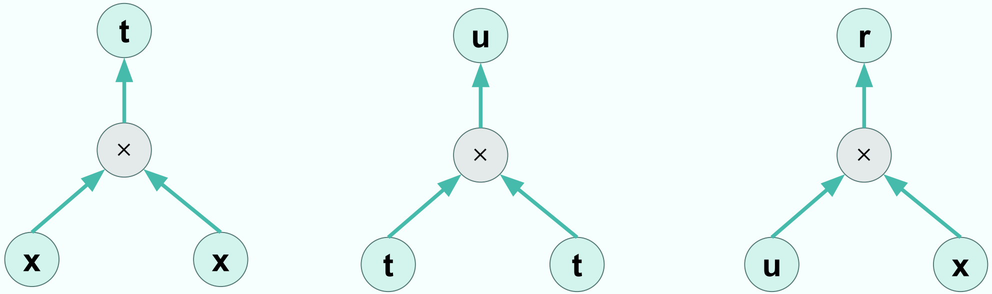
スポンジの各要素に対して,非線形関数を作用させる操作。
論文では$f(x) = x^3$と$f(x) = x^5$の場合について記述されている。
※有限体上の累乗であることに注意

算術回路：

$f(x) = x^3$



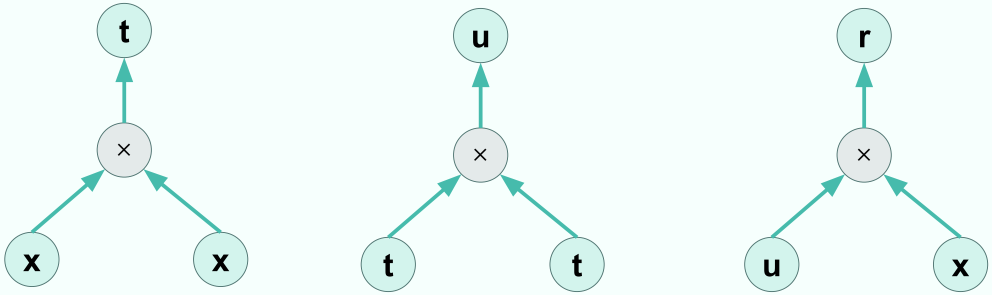
$f(x) = x^5$



R1CSでのPoseidon：S-Box

スポンジの1要素に対するS-Box適用方法

$$f(x) = x^5$$



ゲート1　ゲート2　ゲート3

ゲート1

$w_1 = [x, z_1]^T$

$A_1 = [1, 0],\ B_1 = [1, 0],\ C_1 = [0, 1]$

式: $(A_1 \cdot w_1) \times (B_1 \cdot w_1) = (C_1 \cdot w_1)$

ゲート2

$w_2 = [x, z_1, z_2]^T$

$A_2 = [0, 1, 0],\ B_2 = [0, 1, 0],\ C_2 = [0, 0, 1]$

式: $(A_2 \cdot w_2) \times (B_2 \cdot w_2) = (C_2 \cdot w_2)$

ゲート3

$w_3 = [x, z_1, z_2, y]^T$

$A_3 = [1, 0, 0, 0],\ B_3 = [0, 0, 1, 0],\ C_3 = [0, 0, 0, 1]$

式: $(A_3 \cdot w_3) \times (B_3 \cdot w_3) = (C_3 \cdot w_3)$

R1CSでのPoseidon：S-Box

スポンジの全要素に対するS-Boxの適用方法

制約 #	A	B	C
1	x_0	x_0	$z1_0$
2	$z1_0$	$z1_0$	$z2_0$
3	$z2_0$	x_0	y_0
...	...	...	...
3i + 1	x_i	x_i	$z1_i$
3i + 2	$z1_i$	$z1_i$	$z2_i$
3i + 3	$z2_i$	x_i	y_i
...	...	...	...
3t	$z2_{\square-1}$	$x_{\square-1}$	$y_{\square-1}$

1要素分

一つの要素に対して3制約，長さtのスポンジで3t個制約が生まれる。
それらの制約を通常のr1csと同じ操作でひとつの制約にまとめあげる。

R1CSでのPoseidon：MixLayer

復習

MixLayerはスポンジベクトルに対してMDS行列をかける操作

3×3行列と長さ3のベクトルの掛け算を例にとって理解しよう！

R1CSでのPoseidon：MixLayer

入力x, 行列M, 出力y

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

$$M = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}$$

$$\mathbf{y} = M \cdot \mathbf{x} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

行列の各行をそれぞれ以下のようにおく

$M_0 = [m_{00}, m_{01}, m_{02}]$,
$M_1 = [m_{10}, m_{11}, m_{12}]$,
$M_2 = [m_{20}, m_{21}, m_{22}]$

この時，出力ベクトルの各要素は次のような式となる

$$y_0 = \mathbf{M}_0 \cdot \mathbf{x} = m_{00}x_0 + m_{01}x_1 + m_{02}x_2$$
$$y_1 = \mathbf{M}_1 \cdot \mathbf{x} = m_{10}x_0 + m_{11}x_1 + m_{12}x_2$$
$$y_2 = \mathbf{M}_2 \cdot \mathbf{x} = m_{20}x_0 + m_{21}x_1 + m_{22}x_2$$

R1CSでのPoseidon：MixLayer

$$y_0 = m_{00}x_0 + m_{01}x_1 + m_{02}x_2$$

この関係式をr1csの形式にする
線形結合の形になっているので，

$$(m_{i0}x_0 + m_{i1}x_1 + m_{i2}x_2) \cdot 1 = y_i$$

という線形制約の形になる

二つの式の項を対応させる

$$(m_{i0}x_0 + m_{i1}x_1 + m_{i2}x_2) \cdot 1 = y_i$$

$$(\mathrm{a_i} \cdot \mathrm{z}) \times (\mathrm{b_i} \cdot \mathrm{z}) = \mathrm{c_i} \cdot \mathrm{z}$$

$$\mathbf{a}_0 \cdot \mathbf{z} = m_{00}x_0 + m_{01}x_1 + m_{02}x_2$$

$$\mathbf{b}_0 \cdot \mathbf{z} = 1$$

$$\mathbf{c}_0 \cdot \mathbf{z} = y_0$$

R1CSでのPoseidon：MixLayer

重複のないすべての中間変数を集めたベクトルzを次のように定義する

$$\mathbf{z} = \begin{bmatrix} 1 \\ x_0 \\ x_1 \\ x_2 \\ y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

zとの内積が右辺になるようなa, b, cベクトルを考える

$$[?, ?, ?, ?, ?, ?, ?] \cdot \begin{bmatrix} 1 \\ x_0 \\ x_1 \\ x_2 \\ y_0 \\ y_1 \\ y_2 \end{bmatrix} = m_{00}x_0 + m_{01}x_1 + m_{02}x_2$$

R1CSでのPoseidon：MixLayer

すべてのベクトルの穴埋めをした結果

制約番号	$\mathbf{a}_i$（左）	$\mathbf{b}_i$（右）	$\mathbf{c}_i$（出力）
$i=0$	$[0, m_{00}, m_{01}, m_{02}, 0, 0, 0]$	$[1, 0, 0, 0, 0, 0, 0]$	$[0, 0, 0, 0, 1, 0, 0]$
$i=1$	$[0, m_{10}, m_{11}, m_{12}, 0, 0, 0]$	$[1, 0, 0, 0, 0, 0, 0]$	$[0, 0, 0, 0, 0, 1, 0]$
$i=2$	$[0, m_{20}, m_{21}, m_{22}, 0, 0, 0]$	$[1, 0, 0, 0, 0, 0, 0]$	$[0, 0, 0, 0, 0, 0, 1]$

$$\mathbf{z} = \begin{bmatrix} 1 \\ x_0 \\ x_1 \\ x_2 \\ y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

R1CSでのPoseidon：MixLayer

それぞれの制約を行として結合し，
最終的な制約の行列を得る

$$\mathbf{A} = \begin{bmatrix} 0 & m_{00} & m_{01} & m_{02} & 0 & 0 & 0 \\ 0 & m_{10} & m_{11} & m_{12} & 0 & 0 & 0 \\ 0 & m_{20} & m_{21} & m_{22} & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

【1. 前提と記号の定義】

R1CS（Rank-1 Constraint System）上でベクトルに行列を掛ける操作を表現するために、以下の記号と前提を定義します。

- 計算はすべて有限体 $\mathbb{F}$ 上で行う。
- 入力ベクトル x は長さ3の列ベクトル：
  $x = [x_0, x_1, x_2]^t$
- 行列 M は 3×3 の正方行列：

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

$$M = \begin{bmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{bmatrix}$$

- 出力ベクトル $y = Mx = [y_0, y_1, y_2]^t$

【2. ベクトル×行列の通常の演算式】

行列とベクトルの積 y = Mx は、行列の各行ベクトルとベクトル x の内積として計算されます。

行列の各行を以下のように定義します:
$M_0 = [m_{00}, m_{01}, m_{02}]$, $M_1 = [m_{10}, m_{11}, m_{12}]$, $M_2 = [m_{20}, m_{21}, m_{22}]$

各出力成分は以下のように対応します:

$$
\begin{aligned}
y_0 &= m_{00}x_0 + m_{01}x_1 + m_{02}x_2 \\
y_1 &= m_{10}x_0 + m_{11}x_1 + m_{12}x_2 \\
y_2 &= m_{20}x_0 + m_{21}x_1 + m_{22}x_2
\end{aligned}
$$

【3. R1CS制約の形式】

R1CS（Rank-1 Constraint System）では、各制約は次の形式で表されます：

$$(a_i \cdot z) \times (b_i \cdot z) = c_i \cdot z$$

ここで：
- z はすべての変数をまとめたワイヤーベクトル
- a_i, b_i, c_i はそれぞれ制約の係数ベクトル
- “·” はベクトルの内積（ドット積）を表す

この制約は、2つの線形結合の積が別の線形結合に等しいという意味を持ち、
ゼロ知識証明の演算回路を構築する際の基本単位となります。

Thank you!