無開示性を持つ
カードベース暗号
プロトコルについて

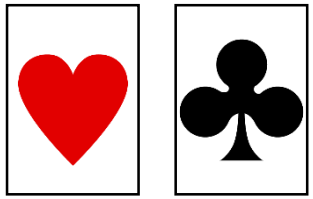
真鍋義文
工学院大学

工学院大学
KOGAKUIN UNIVERSITY

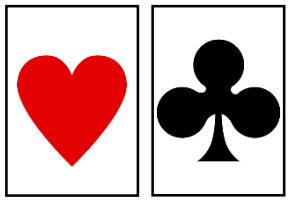
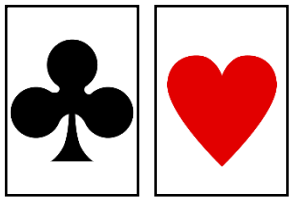
発表の流れ

- プライベートモデルのカードベース暗号プロトコル
- プライベート処理時の情報漏洩の可能性について
- 無開示性を持つプロトコル
- プロトコルの実例
- 任意の関数の実現法
- 計算可能な関数族

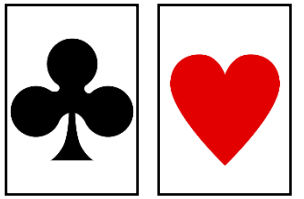
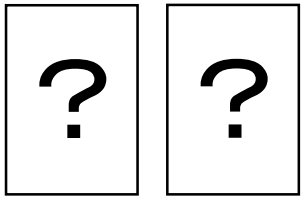
用いるカードとその意味


いずれも裏は ？ 判別不能

- 1ビットデータのエンコーディング


＝1

＝0

- データとそのコミットメント


裏返したもの


x $commit(x)$

1枚エンコーディング

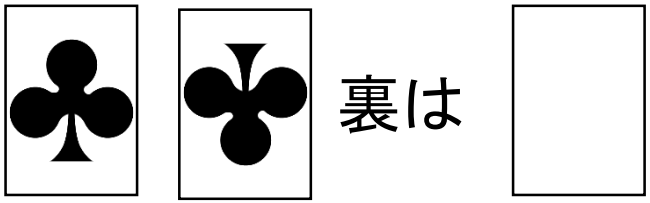

＝1

＝0

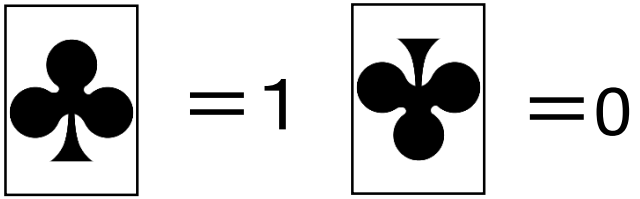
裏返したもの

$commit(x)$

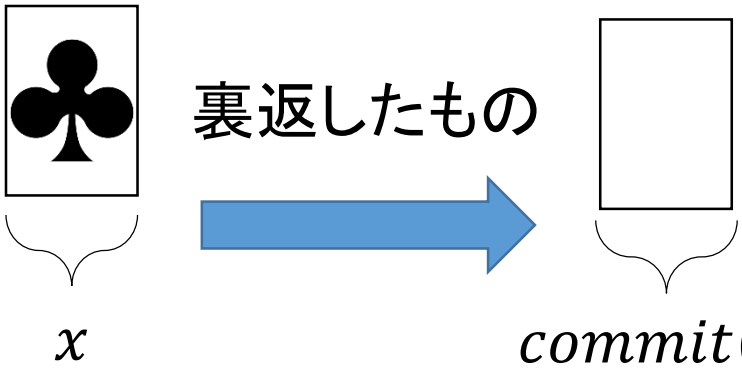
非対称カードの場合



- 1ビットデータのエンコーディング



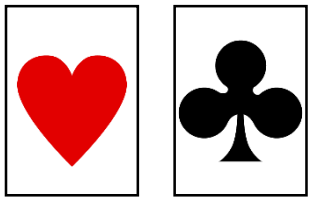
- データとそのコミットメント



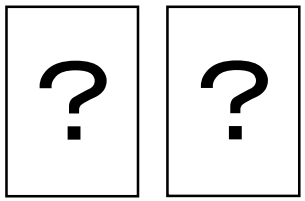
否定の計算

- ２枚で1ビット表現
- 非対称カード

で可能



の場合



$commit(x)$

$commit(\bar{x})$

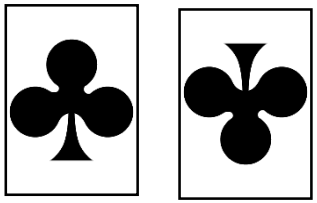
左右入れ替え

１枚エンコーディング

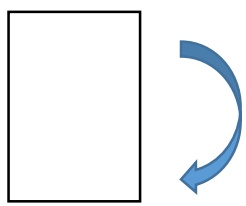
＝1

＝0

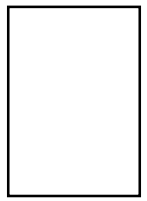
ではcommit(x)の
否定は困難



の場合



上下反転



$commit(x)$

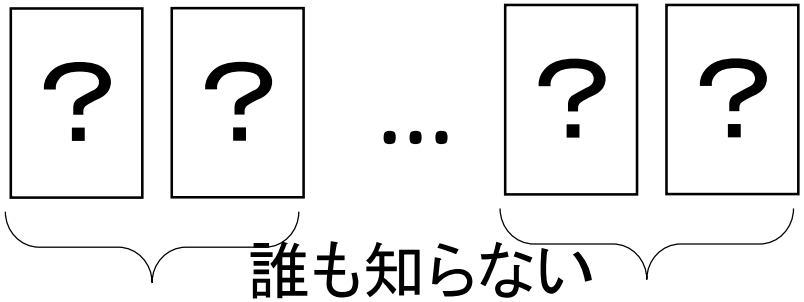
$commit(\bar{x})$

プライベート処理とは

- 他のプレイヤーの見えないところで行う処理
  - 後ろを向いて／机の下で　等
  - カード使用枚数が少なく、効率的プロトコルが得られる可能性
  - ２種類の状況
    - 秘密値計算：全プレイヤーが知らない秘密値による計算
    - 秘匿入力：各プレイヤーが秘密の入力値を持つ

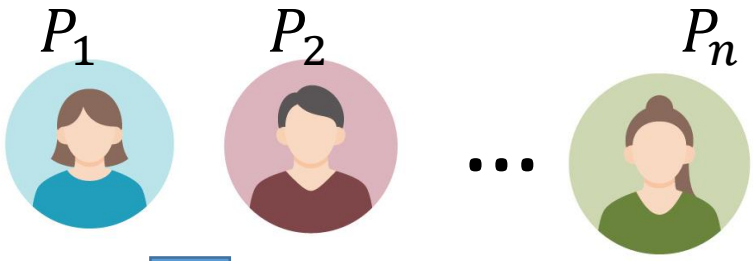
秘密値計算

入力

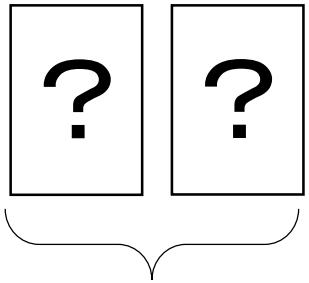


誰も知らない

$commit(x_1)$ $commit(x_m)$

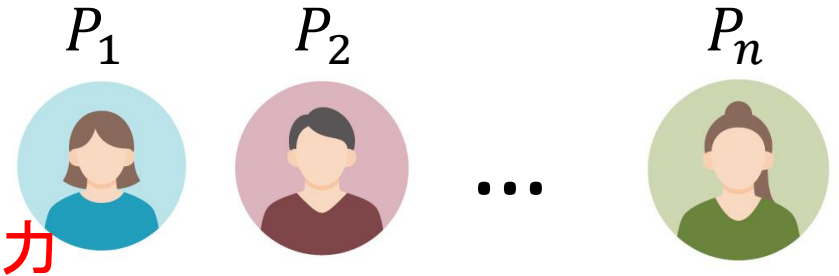


プライベート処理を含む処理



$commit(f(x_1, x_2, \ldots, x_m))$

秘匿入力

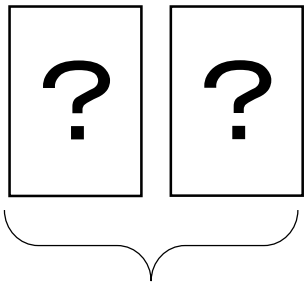


入力

x_1 x_2 x_n

他のプレイヤーは知らない

プライベート処理を含む処理



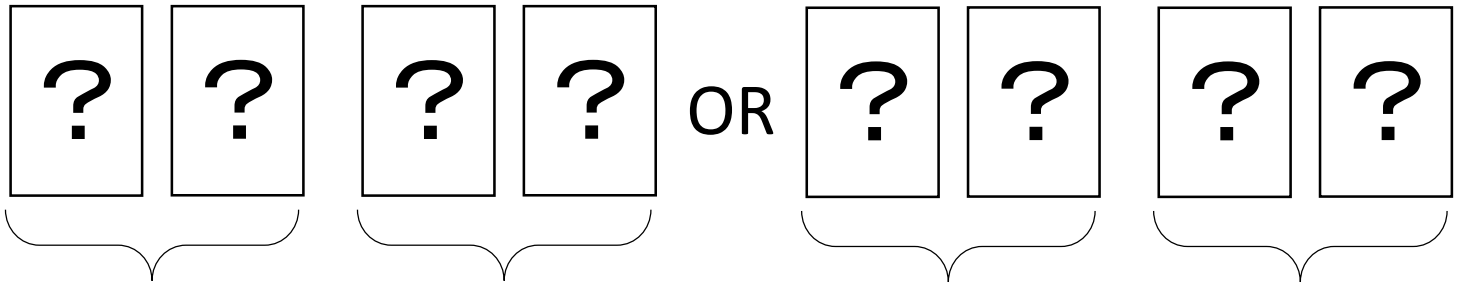
$commit(f(x_1, x_2, \ldots, x_n))$

（例）秘密値計算ANDプロトコル

- 入力：$commit(x), commit(y)$、出力：$commit(x \wedge y)$

（1）Alice:
xに対して
Private random
bisection cut

（2）Bob:
$x \oplus b$を
private reveal:
見た値によって
カード列をセット



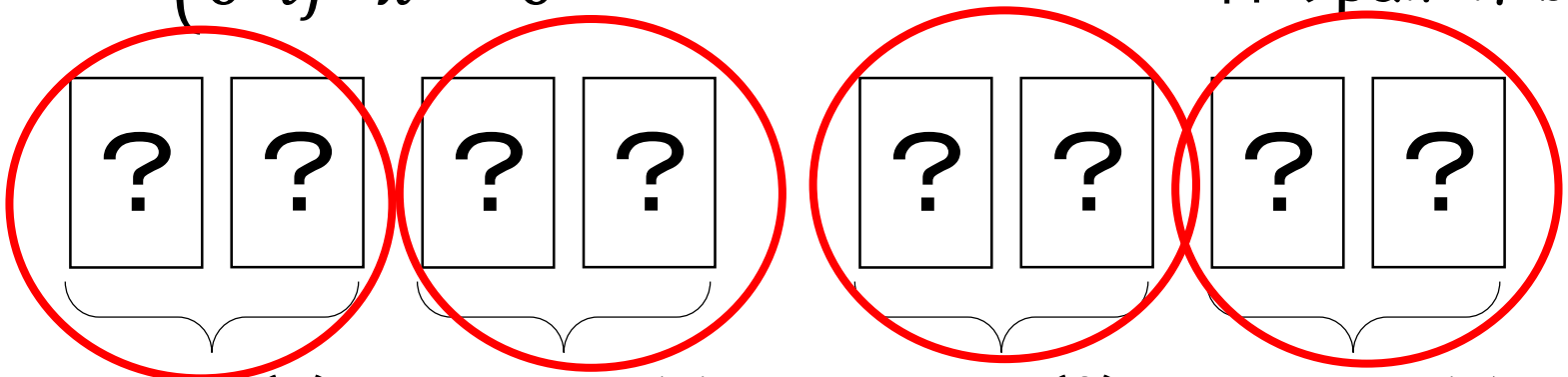
（3）Alice:
Private reverse selection
b =0なら左、b = 1なら右の
2枚を選択

使用枚数：4枚
$commit(x \oplus b)$を$commit(0)$に再利用

ANDプロトコルの正しさ

$$x \wedge y = \begin{cases} y & if \quad x = 1 \\ 0 & if \quad x = 0 \end{cases}$$

Alice の選択　左のpair if $b = 0$
右のpair if $b = 1$



$commit(y)$　$commit(0)$　　$commit(0)$　$commit(y)$

$x \oplus b = 1$　　$x \oplus b = 0$

$commit(y)$を選択：$(x \oplus b = 1$ and $b = 0)$ or $(x \oplus b = 0$ and $b = 1) \Leftrightarrow x = 1$

$commit(0)$を選択：$(x \oplus b = 1$ and $b = 1)$ or $(x \oplus b = 0$ and $b = 0) \Leftrightarrow x = 0$

プライベート処理の問題点

- プレイヤーがプライベート処理時に誤って（あるいは故意に）許されないカードを見る→情報漏洩の可能性
  - （例）前述ANDプロトコルで$commit(y)$をBobが見る
  - 不正／誤りがあっても他のプレイヤーにわからない
- 既存の防止法：処理は複雑化
  - （プレイヤー3者以上の場合）相互監視
  - 追加カードや追加ツールを用いた誤り検出・修正

相互監視
Y. Abe, M. Iwamoto, and K. Ohta: How to Detect Malicious Behaviors in a Card-Based Majority Voting Protocol with Three Inputs Proc. of ISITA, 2020.
H. Ono and Y. Manabe: Card-Based Cryptographic Logical Computations Using Private Operations,' New Generation Computing, 2021
追加ツール（封筒）
Y. Manabe and H. Ono: `` Secure Card-based Cryptographic Protocols Using Private Operations Against Malicious Players,' Proc. of SecITC 2020.
品川: 秘匿互換に基づくカードベース暗号プロトコル, SCIS2019.

無開示性の導入

- プライベート処理時のprivate reveal(秘匿開示）に問題
- →秘匿開示を行わないプロトコルであればカードを封筒等に入れる（簡単に取り出せない）ことで過失／故意による情報漏洩を防止可能
- ×　簡単に取り出せないカード：公開開示も手間が大きくなり、プロトコルの実行時間増

- いっそのこと公開開示もない「無開示性」を持つプロトコルは？

無開示性の定義・意義

- （定義）開示処理のないプロトコル（秘匿入力）
  - 公開開示・秘匿開示どちらもなし
  - 各プレイヤーのプライベート処理結果の（裏向けの）カード列の並び（枚数等）は秘匿入力値に依存しない
- （意義）
  - 開けにくい封筒にカードを入れる等により、プロトコル実行時のミス／故意による情報漏洩を防止可能
  - 上記に伴う実行時間増なし
  - プロトコルの安全性証明は自明
  - プロトコル実行中にミスがあっても最終結果をオープンしなければ漏洩問題なし

無開示性を持つプロトコル例(1)

- AND
- P_iが値$x_i \in \{0,1\}$を持つ
- $f(x_1, x_2, \ldots, x_n) = x_1 \wedge x_2 \wedge \cdots \wedge x_n$

XORも
計算可能

- 等号判定
- P_iがベクトル$x_i \in \{0,1\}^m$を持つ
- $f(x_1, x_2, \ldots, x_n) = 1$ if $x_1 = x_2 = \cdots = x_n$
  otherwise 0

品川: 秘匿互換に基づくカードベース暗号プロトコル, SCIS2019.

無開示性を持つプロトコル例(2)

- majority voting, threshold function
- P_iが値$x_i \in \{0,1\}$を持つ
- $f(x_1, x_2, \ldots, x_n) = 1$ if $\sum_{i=1}^{n} x_i > t$, otherwise 0

Y. Abe, T. Nakai, Y. Watanabe, M. Iwamoto, and K. Ohta: Computationally Efficient Card-Based Majority Voting Protocol with Fewer Cards in the Private Model, IEICE Transactions on Fundamentals E106-A, 2023.
Y. Abe, T. Nakai, Y. Kuroki, S. Suzuki, Y. Koga, Y. Watanabe, M. Iwamoto, and K. Ohta, Efficient Card-Based Majority Voting Protocols, New Generation Computing, Vol. 40, 2022.
S . Nakai, S. Shirouchi, Y. Tokushige, M. Iwamoto, and K. Ohta: Secure Computation for Threshold Functions with Physical Cards: Power of Private Permutations, New Generation Computing, Vol. 40, 2022.

無開示性を持つプロトコル例(3)

- private set intersection
- P_iが$A_i \subset X = \{x_1, x_2, \ldots x_n\}$を持つ$(i = 1,2)$
- $f(x_j) = 1$ if $x_j \in A_1 \cap A_2$ otherwise 0 $(1 \leq j \leq n)$

A. Doi, T. Ono, T. Nakai, K. Shinagawa, Y. Watanabe, K. Nuida, and M. Iwamoto: Card-based Cryptographic Protocols for Private Set Intersection, Proc of ISITA 2022

無開示性を持つプロトコル例(4)

- 金持ち比べ

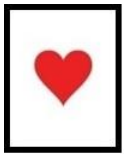
AliceとBobのどちらの財産が多いかの比較結果を、財産の額を互いに知られることなく得る

H. Ono and Y. Manabe: Efficient Card-based Cryptographic Protocols for the Millionaires‘ Problem Using Private Input Operations,‘‘ Proc. of AsiaJCIS, 2018

金持ち比べプロトコル：入出力

- Aliceの入力: $a(i)(i = 1, \dots, n)$
- Bobの入力: $b(i)(i = 1, \dots, n)$
- 出力:　commit(s)

$a(1), b(1)$: LSB

$a \geq b$ if $s =$ 

$a < b$ if $s =$ 

$s^{(i)}$：iビット目までの結果

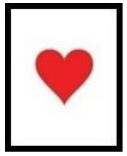
$s = s^{(n)}$

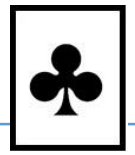
例:

$a(3) = 0, a(2) = 1, a(1) = 1$

$b(3) = 1, b(2) = 1, b(1) = 0$

$s^{(1)} =$  　$1 > 0$

$s^{(2)} =$  　$11 > 10$

$s = s^{(3)} =$  　$011 < 110$

1ビット目の計算

- 入力: $a(1), b(1)$
- 出力: $s^{(1)}$

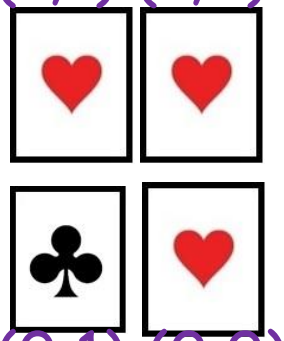
正しさ:
$s^{(1)}$ when $(a(1), b(1))$ is

Alice

Bob

プライベート処理でセット

(1,1) (1,0)



$$x_1x_2 = \begin{cases} if\ \ a(1) = 1 \\ if\ a(1) = 0 \end{cases}$$

(0,1) (0,0)

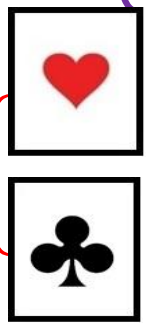
(裏面でおく)

x_1x_2を渡す

プライベート処理で選択

$$s^{(1)} = \begin{cases} \text{左 } if\ \ b(1) = 1 \\ \text{右 } if\ b(1) = 0 \end{cases}$$

(非選択カード削除)



$$s^{(1)} = \begin{cases} \text{if } a(1) \geq b(1) \\ \text{if } a(1) < b(1) \end{cases}$$

iビット目の計算 ($i > 1$, i 偶数)

- 入力: $a(i), b(i), s^{(i-1)}$
- 出力: $s^{(i)}$

Alice

正しさ:

$s^{(i)}$ when $(a(i), b(i))$ is

Bob

プライベート処理でセット

(1,1) (0,1)

$$S = \begin{cases} s^{(i-1)} \clubsuit & if \ \ b(i) = 1 \\ \heartsuit \ s^{(i-1)} & if \ b(i) = 0 \end{cases}$$

(1,0) (0,0)

(裏面でおく)

Sを送付

プライベート処理で選択

$$s^{(i)} = \begin{cases} \text{左} & if \ \ a(i) = 1 \\ \text{右} & if \ a(i) = 0 \end{cases}$$

(非選択カード削除)

$$s^{(i)} = \begin{cases} \heartsuit & \text{if } (a(i), b(i)) = (1,0) \\ \clubsuit & \text{if } (a(i), b(i)) = (0,1) \\ s^{(i-1)} & \text{if } a(i) = b(i) \end{cases}$$

iビット目の計算 ($i > 1$, i 奇数)

- 入力: $a(i), b(i), s^{(i-1)}$
- 出力: $s^{(i)}$

Alice

$s^{(i)}$

プライベート処理でセット

(1,1) (1,0)

$$S = \begin{cases} s^{(i-1)} & \heartsuit \quad if \;\; a(i) = 1 \\ \clubsuit & s^{(i-1)} \quad if \;\; a(i) = 0 \end{cases}$$

(0,1) (0,0)

(裏面でおく)

S送付

Bob

プライベート処理でセット

$$s^{(i)} = \begin{cases} 左 & if \;\; b(i) = 1 \\ 右 & if \;\; b(i) = 0 \end{cases}$$

(非選択カード削除)

正しさ:

$s^{(i)}$ when $(a(i), b(i))$ is

$$s^{(i)} = \begin{cases} \heartsuit & \text{if } (a(i), b(i)) = (1,0) \\ \clubsuit & \text{if } (a(i), b(i)) = (0,1) \\ s^{(i-1)} & \text{if } a(i) = b(i) \end{cases}$$

プロトコルの評価

nビットの金額	Ono et al.	Nakai et al.
カード枚数	$2n+1$	2^{n+1}
ラウンド数	$n+1$	2

Nakai et al., Efficient Card-Based Cryptographic Protocols for Millionaires' Problem Utilizing Private Permutations, CANS2016

任意の関数の計算(1)

- 入力: $x_1, x_2, \ldots x_n \in \{0,1\}$:プレイヤー$P_i$が$x_i$を持つ
- 出力:$commit\big(f(x_1, x_2, \ldots, x_n)\big)$
- プロトコル
  - (1)全プレイヤーが共同で2^n個の$commit\big(f(a_1, a_2, \ldots, a_n)\big)$ $a_i = 0,1(1 \leq i \leq n)$を作成
  - (2)P_1がプライベート処理で$commit\big(f(x_1, a_2, \ldots, a_n)\big)$を選択
  - (3)P_2がプライベート処理で$commit\big(f(x_1, x_2, \ldots, a_n)\big)$を選択

  .....
  - (n+1) P_nがプライベート処理で$commit\big(f(x_1, x_2, \ldots, x_n)\big)$を選択

任意の関数の計算(2)

- 入力: $x_1, x_2, \ldots x_n \in \{0,1\}$:プレイヤー$P_i$が$x_i$を持つ
- 出力:$commit\big(f(x_1, x_2, \ldots, x_n)\big)$
- Barringtonの定理により、任意の深さdの論理回路fを幅が5のbranching programで構成可能⇒カード5枚で計算可能
- 計算ステップは$O(4^d)$

品川: 秘匿互換に基づくカードベース暗号プロトコル, SCIS2019.

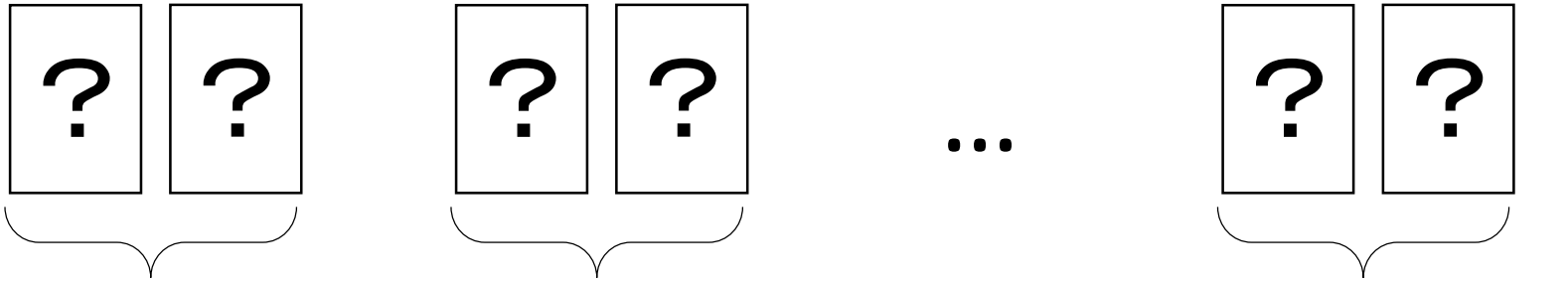
任意の対称関数の計算(1)

- 対称関数　$f(x_1, x_2, \ldots, x_n) = f\left(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}\right)$ for any $\pi: \{1,2, \ldots, n\} \to \{1, 2, \ldots, n\}$
- 関数値は$x_1, x_2, \ldots, x_n$の1の数で決まる→
  $g(j)$: $x_1, x_2, \ldots, x_n$の1の数がjのときの関数値、とすると

  fを$(g(0), g(1), \ldots, g(n))$の組で表現可能:

任意の対称関数の計算(2)

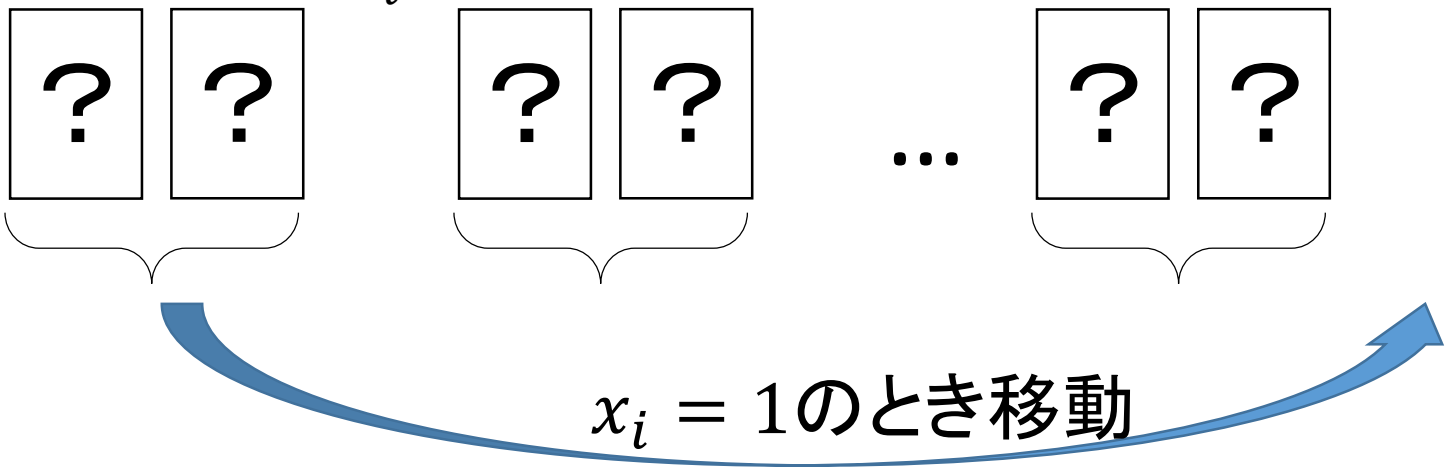
- 入力$x_1, x_2, \ldots x_n \in \{0,1\}$:プレイヤー$P_i$が$x_i$を持つ
- 出力$commit\big(f(x_1, x_2, \ldots, x_n)\big)$
- プロトコル
  - (1)全プレイヤーが共同で以下のカード列作成



任意の対称関数の計算(3)

(2)$i = 1,2,\ldots,n$に対して以下を実行

- P_iが$x_i = 1$のときに左端の$commit$カード(組)を右端に移動 ($x_i = 0$のときは何も行わない)



(3)左端のカード(組)を出力とする

$x_1, x_2, \ldots, x_n$の1の個数がm→左端は$commit(g(m))$

重み付きvotingに拡張可能(複数個の移動)

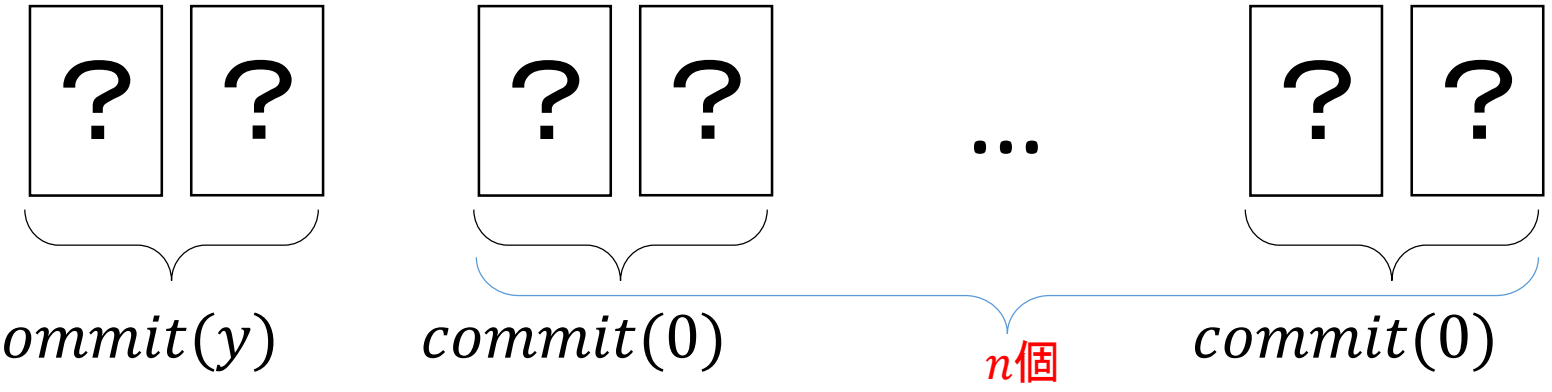
課題

- 任意の関数の計算を行う、効率的なプロトコル
  - 使用枚数最小（5枚）:ステップ$O(4^d)$
  - 使用枚数指数（2^n枚):ステップ最小n
- 対象とする問題の性質を利用した、より効率的なプロトコル
  - AND, XOR,等号判定,voting, private set intersection, 金持ち比べ以外の問題に対するプロトコル

プレイヤーが一部の値を知らない場合に拡張

- （例1）AND関数$f = y \wedge x_1 \wedge x_2 \wedge \cdots \wedge x_n$
- 入力:$x_i \in \{0,1\}$:プレイヤーP_iが持つ

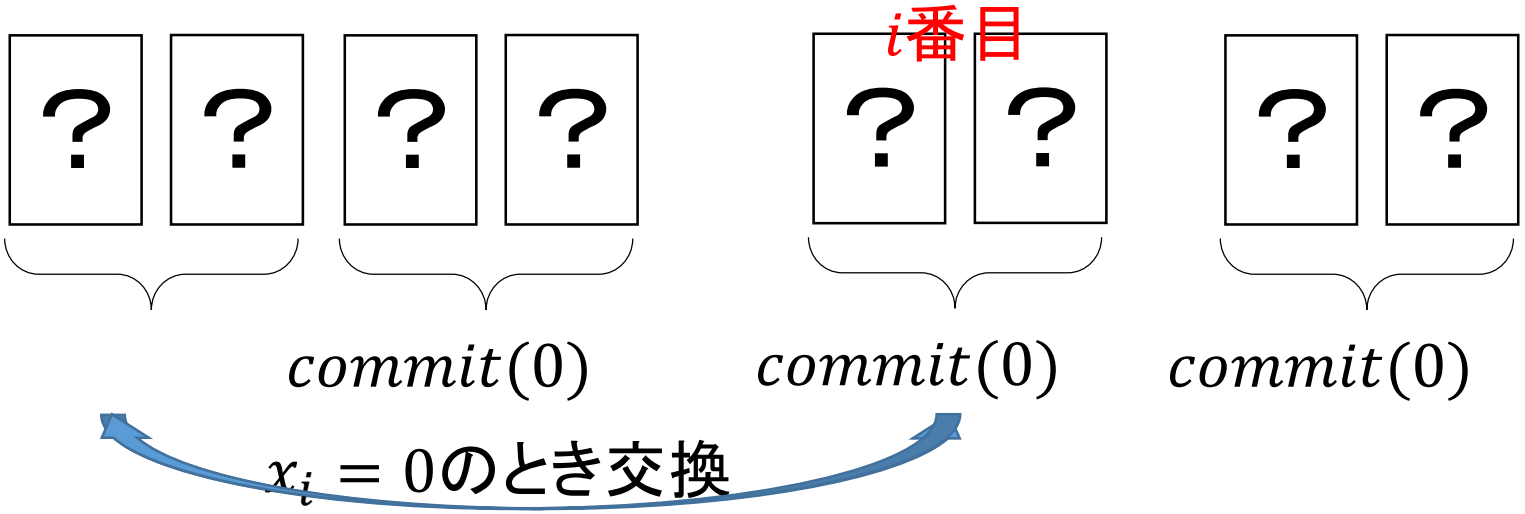
  $commit(y)$: 誰も知らない
- 出力:$commit(y \wedge x_1 \wedge x_2 \wedge \cdots \wedge x_n)$
- プロトコル
  - (1)全プレイヤーが共同で以下のカード列作成



プレイヤーが一部の値を知らない場合に拡張

(2) $i = 1,2, \ldots, n$に対して以下を実行

- P_iがプライベート処理で$x_i = 0$のときに左端の$commit$カード(組)とi番目のカード(組)を交換($x_i = 1$のときは何も行わない)



(3)左端のカード(組)を出力とする

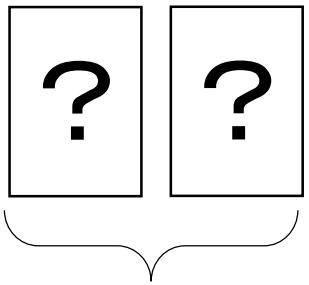
結果は　$commit(y)$　$if\ x_i = 1\ (1 \le i \le n)$

$commit(0)$　それ以外

プレイヤーが一部の値を知らない場合に拡張

- （例2）XOR関数$f = y \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_n$
- 入力:$x_i \in \{0,1\}$:プレイヤーP_iが持つ

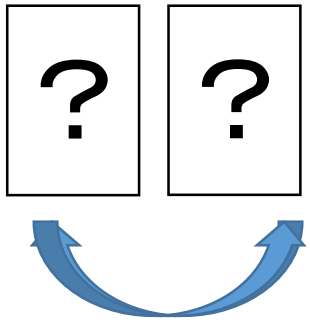
  $commit(y)$: 誰も知らない
- 出力:$commit(y \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_n)$
- プロトコル
  - (1)$commit(y)$をおく



プレイヤーが一部の値を知らない場合に拡張

(2) $i = 1,2, \ldots, n$に対して以下を実行

- P_iがプライベート処理で$x_i = 1$のときに左右入れ替え（$x_i = 0$のときは何も行わない）



非対称カードの
場合は上下入れ替え

$x_i = 1$のとき左右入れ替え

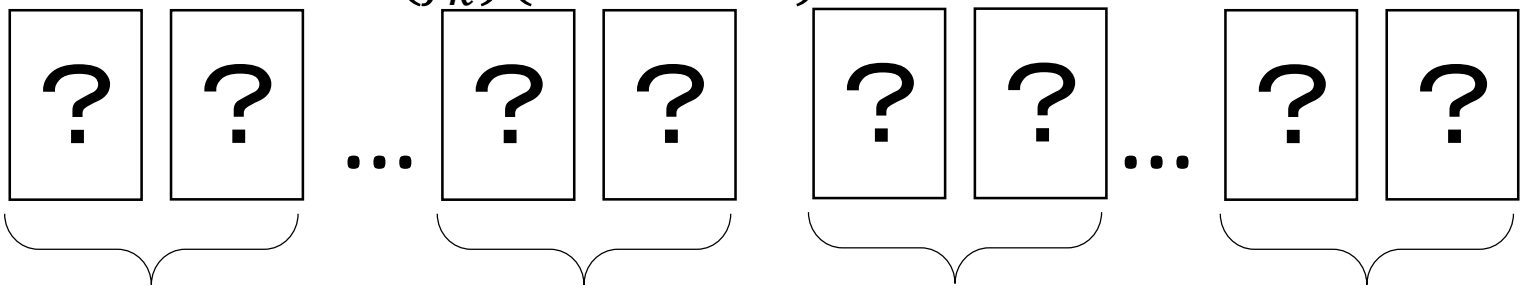
(3)最終結果を出力とする

このプロトコルを一般化

- 入力:$x_i \in \{0,1\}$:プレイヤー$P_i(1 \leq i \leq n)$が持つ

  $commit(y_j)\ (1 \leq j \leq m)$: 誰も知らない

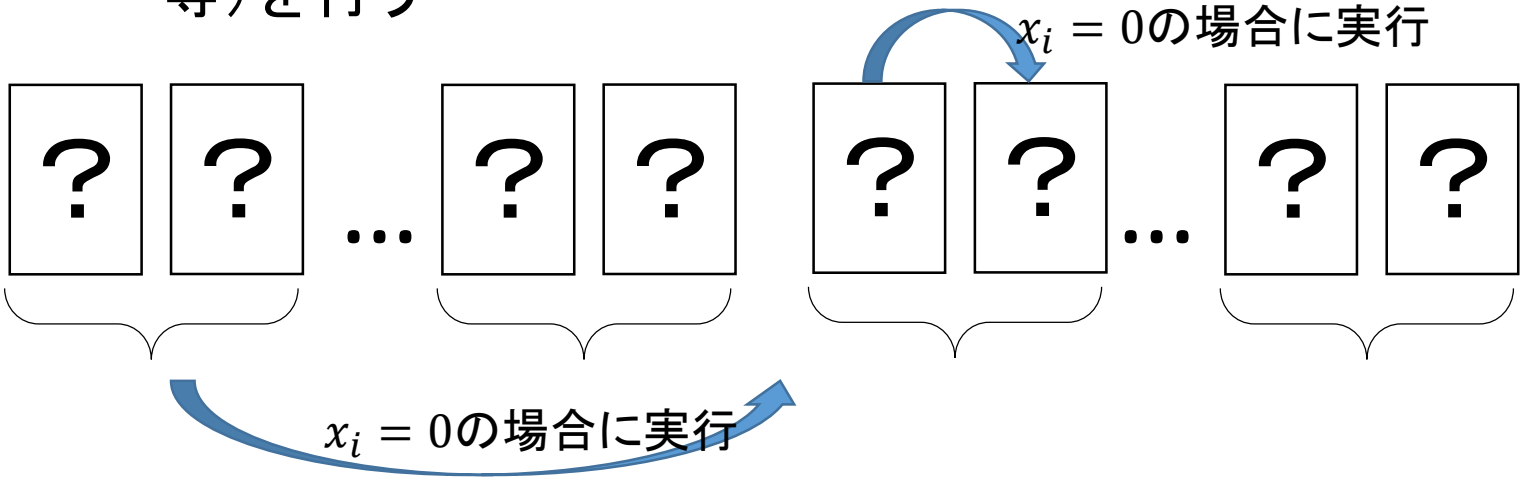
- プロトコル
  - (1)$x_i(1 \leq i \leq n)$と定数のみからなる任意の関数$f_k(1 \leq k \leq l)$に対し$commit(f_k)$を計算
  - (2)全プレイヤーが共同で$commit(y_j)(1 \leq j \leq m)$, $commit(f_k)(1 \leq k \leq l)$ を並べる



このプロトコルを一般化

(3)以下の手続きを繰り返す

- P_iがプライベート処理でx_iの値に依存した位置の入れ替え(場所の入れ替え、コミットメントの左右入れ替え等)を行う



(4)あらかじめ決めておいた場所のカード(左端の組、等)を出力とする

一般化手続きに関する課題

- 「プレイヤー」による繰り返し」はどこまで必要？
  - 各プレイヤー1回で十分？　もっと必要？
- この手続きで得られる関数族は？
  - １枚表現（♡または♣）では否定ができない→どのような関数族となるか？
  - 同じ関数族を、制限した手続きで得られるか？
- この手続きで得られる有用な応用プロトコルは？
- これより広い関数族を得られる手続きはあるか？
- （頂いたコメント）一般のプロトコルを無開示性を持つプロトコルに変換する手続きはあるか？

まとめ

- プライベート処理によるカードベース暗号プロトコルにおける無開示性
  - 各プレイヤーが秘匿入力値を持つ場合に可能
  - 無開示性のメリット
  - 任意の関数を計算可能
  - （残された課題１） 効率的なプロトコル
  - （残された課題２） 計算可能関数の解明