カードベースガーブルド回路のカード枚数削減手法とその応用

戸澤一成（東京大学）

産学連携と数理・暗号分野連携によるカードベース暗号の深化と新境地 II 28 May, 2025

この発表について

発表の内容

本発表の内容は、以下の国際会議および論文誌にて発表された研究成果に基づいています：

- 国際会議 UCNC 2023

  K. Tozawa, H. Morita, and T. Mizuki. “Single-shuffle card-based protocol with eight cards per gate”

- 論文誌 Natural Computing, Vol. 24

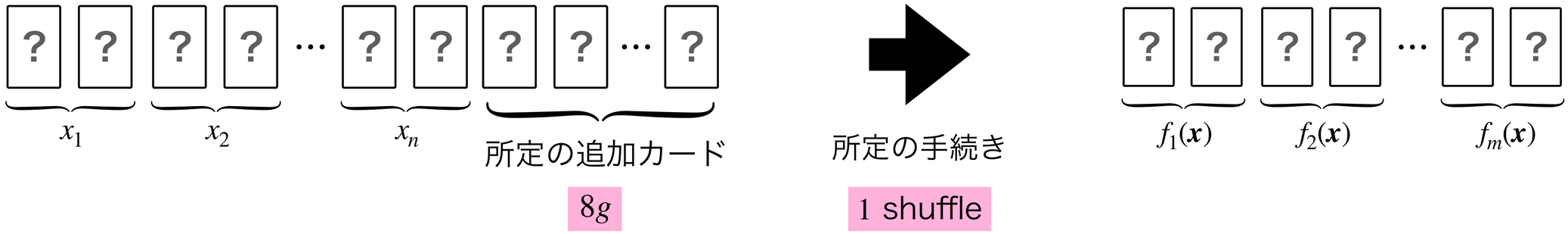
  K. Tozawa, H. Morita, and T. Mizuki. “Single-shuffle card-based protocol with eight cards per gate and its applications”

結果 (UCNC ’23)

Efficient Card-based Protocol for Any Boolean Function

f をブール関数，C を入力数 n, ゲート数 g, 出力数 m の論理回路で f を計算するものとする.

任意のブール関数 f を秘匿計算する効率の良いカードベースプロトコルを構成した.



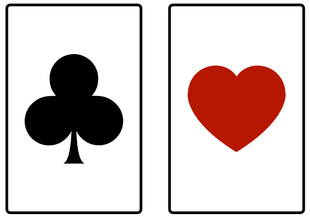
- プロトコル中の shuffle の回数は合計で 1 回. （時間計算量）
- プロトコル実行に必要なカードの枚数は合計で $2n + 8g$. （空間計算量）
- 同様の機能を実現している single-shuffle プロトコル [SN21]と比較して，追加カードの枚数を $\frac{1}{3}$ に削減している. ($2n + 24g$ 枚から $2n + 8g$ 枚)

[SN21] K. Shinagawa and K. Nuida. A single shuffle is enough for secure card-based computation of any boolean circuit. Discrete Applied Mathematics, 289:248–261, 2021.

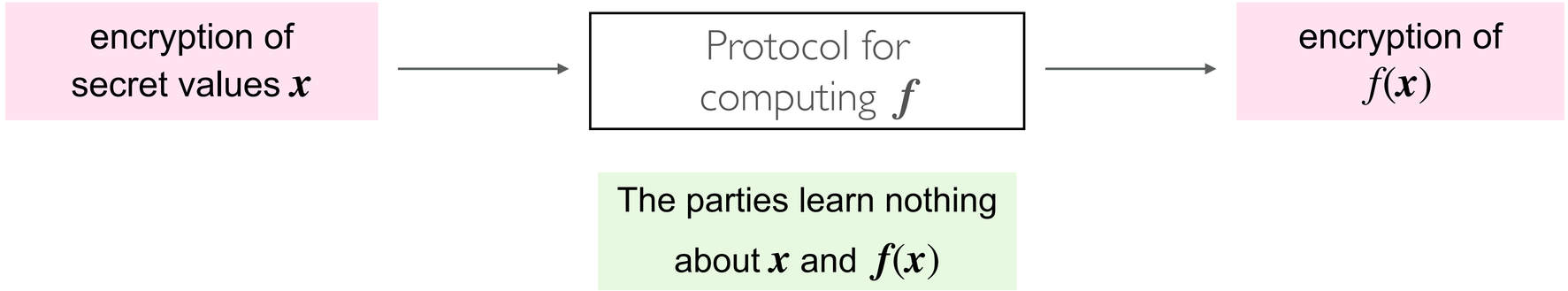
背景

Card-based Cryptography for Any Boolean Function

- **カードベース暗号:** カードを用いて暗号プロトコルを実現する手法
  - コンピュータを使用する必要がなく、カードを準備するだけで容易に実行可能
  - 操作が非常に簡潔であり、正当性や安全性を直感的に理解しやすい



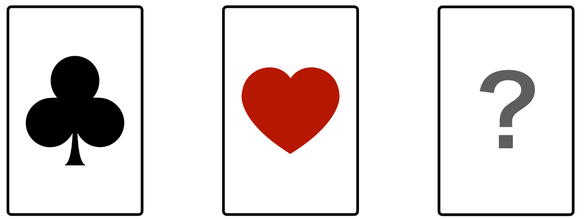
- **秘密計算:** 入力値を秘匿したまま、所望の関数を安全に計算するための暗号プロトコル
  - 入力: 秘密の値 $x_1, \ldots, x_n$ の暗号文 $[x_1], \ldots, [x_n]$
    出力: 所望の値 $f(x_1, \ldots, x_n)$ の暗号文 $[f(x_1, \ldots, x_n)]$
  - 暗号化された値のみを用いて計算を行うため、秘密の値が漏洩しない



設定

Standard Encoding for Card-based Cryptography

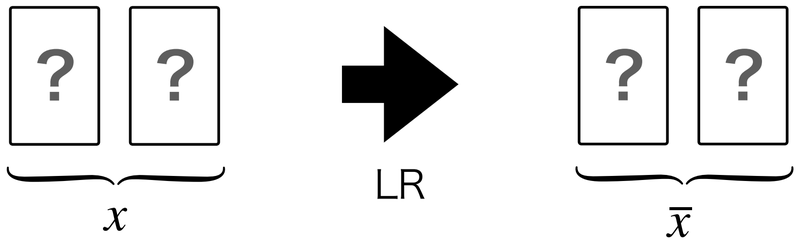
- 裏面から区別できない２種類のカードを利用する
  - 同じ種類のカードは区別できない (上下の反転は区別できない)
  - 各カードのデッキ中の位置は公開



裏面

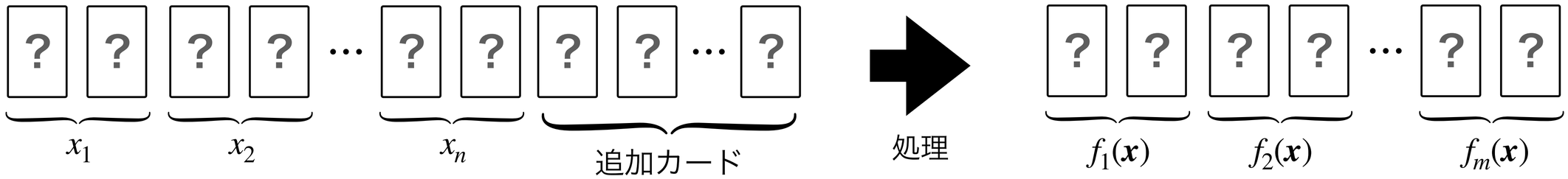
- ブール値は２枚のカード列で表現される (standard encoding):

- 値 x を表現するカード列が裏になっているとき x のコミットメントとよぶ
  - 値 x のコミットメントに対し，$\bar{x}$ (= NOT x) のコミットメントは左右のカードをスワップすることで得られる



設定
コミット型プロトコル

- 入力および出力はコミットメントで与えられる (**コミット型**プロトコル)



- プロトコルは以下の操作から構成される：
  - (Perm, σ): カード列 D を順列 $\sigma \in S_N$ に従って並び替える
  - (Shuffle, G): カード列 D を置換群 G から一様ランダムに選んだ順列 $\sigma \in G$ で並び替える
  - (Open, S): カード列 D の各 $s_i \in S$ 番目のカードを反転する
  - (Output, S): カード列 D の $s_i \in S$ 番目のカードを出力する

- プロトコルは次の性質を満たすものとする：
  - 正当性: プロトコルの出力が有限回の操作によって得られ，所望の値 $f(x)$ のコミットメントになっている
  - (Semi-honest) 安全性: プロトコルの実行中に得られる情報から，入出力 $x, f(x)$ の情報が漏れない

手法

Main Idea: Card-based Variant of Point-and-Permute Technique

- **Card-based Garbled Circuit** [SN21] のカード枚数を削減する
- **Garbled Circuit:** 暗号学的ハッシュ関数を用いた秘密計算方式 [Yao86]
  - [SN21] はカードベース暗号を用いて Garbled Circuit を実現している
  - **Point-and-Permute technique** [BMR90] は Garbled circuit の効率化手法の一種.
- Point-and-Permute technique のカードベース暗号での類似を提案し [SN21] を効率化したい

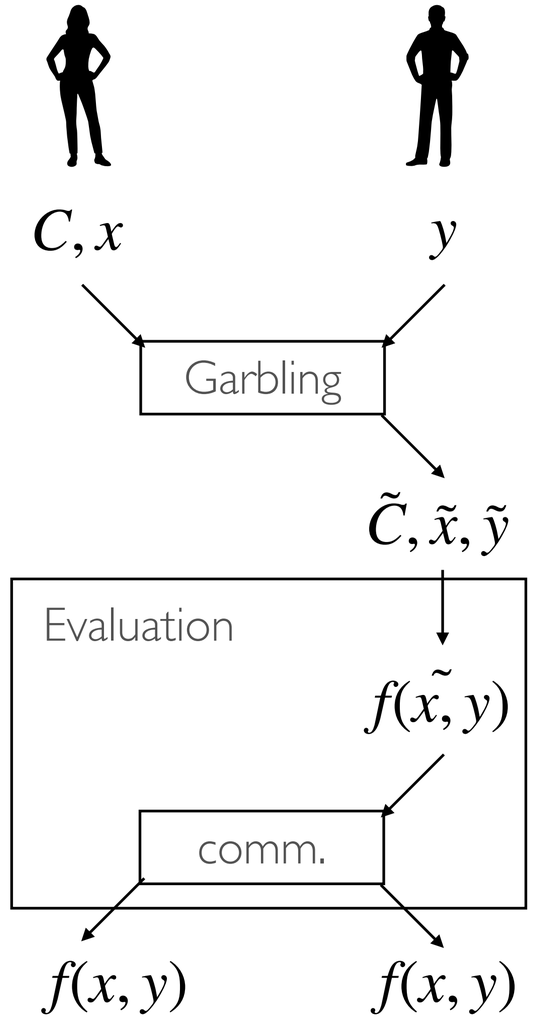
Garbled Circuit	**カードベース暗号**
Yao's Garbled Circuit [Yao86]	Card-based Garbled Circuit [SN21]
Point-and-Permute Technique [BMR90]	**Ours**

[Yao86] A. C. Yao. How to generate and exchange secrets. In 27th IEEE FOCS, 162–167, 1986.
[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In 22nd ACM STOC, 503–513, 1990.

Garbled Circuit

Yao's Garbled Circuit [Yao86] の概要

- 暗号学的ハッシュ関数を用いた秘密計算方式 [Yao86]
  - Garbler と Evaluator によるマルチパーティ計算
  - 任意の論理回路を計算可能
  - 計算に必要な通信回数は $O(1)$ 回
  - 通信量（データサイズ）は回路サイズに比例して増加



- 以下の２つの手続きから構成される：
  - **Garbling Phase**:

    論理回路 C を暗号化して得られた Garbled circuit $\tilde{C}$ を Evaluator に送る

    入力ブール値 $x_1, \ldots, x_n$ に対応するキー $\tilde{x}_1, \ldots, \tilde{x}_n$ を協調的に計算し Evaluator に出力
  - **Evaluation Phase**:

    Evaluator は garbled circuit $\tilde{C}$ と入力のキー $\tilde{x}_1, \ldots, \tilde{x}_n$ から所望の出力 $f(\boldsymbol{x})$ のキーを計算

    パーティ間通信を行い最終的な出力を得る

Garbled Circuit

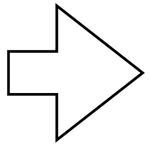
Yao’s Garbled Circuit [Yao86] の構成

- **Garbling Phase**: 回路 C をエンコードして garbled circuit $\tilde{C}$ を計算

  回路中の各ワイヤ w に対し，ブール値 $0,1$ に対応するランダム値のキー k_0^w, k_1^w を選ぶ
  各ゲートの真理値表をハッシュ関数 H を用いてエンコードする

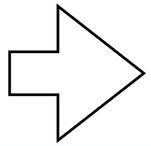
A	B	f
0	0	f(0,0)
0	1	f(0,1)
1	0	f(1,0)
1	1	f(1,1)

truth table



encryption

$$H(k_0^A, k_0^B) \oplus k_{f(0,0)}^C$$
$$H(k_0^A, k_1^B) \oplus k_{f(0,1)}^C$$
$$H(k_1^A, k_0^B) \oplus k_{f(1,0)}^C$$
$$H(k_1^A, k_1^B) \oplus k_{f(1,1)}^C$$



random permutation

$$H(k_{a_1}^A, k_{b_1}^B) \oplus k_{f(a_1,b_1)}^C$$
$$H(k_{a_2}^A, k_{b_2}^B) \oplus k_{f(a_2,b_2)}^C$$
$$H(k_{a_3}^A, k_{b_3}^B) \oplus k_{f(a_3,b_3)}^C$$
$$H(k_{a_4}^A, k_{b_4}^B) \oplus k_{f(a_4,b_4)}^C$$

garbled table

- **Evaluation Phase**: 入力キー $k_{x_1}^{w_1}, \ldots, k_{x_n}^{w_n}$ と garbled circuit $\tilde{C}$ を受け取り出力ワイヤの値 $k_{f(x)}^{w_{out}}$ を計算

  各ワイヤにおいて，Evaluator はキーの一方のみを知ることができる.
  各ゲートにおいて，手持ちの２つのキーを用いて真理値表の４つの値をデコードし，成功したもの一つを出力ワイヤのキーとする.

Garbled Circuit

Point-and-Permute Technique の概要

- **Garbling Phase**: 回路 C をエンコードして garbled circuit $\tilde{C}$ を計算
  回路中の各ワイヤ w に対し，ブール値 $0,1$ に対応するランダム値のキー k_0^w, k_1^w を選ぶ
  ただし，最下位ビットは異なるものとする. ($k_0^w[0] \oplus k_0^w[1] = 1$. 以下，$s_w := k_0^w[0]$ と書く)
  各ゲートの真理値表をハッシュ関数 H を用いてエンコードする

A	B	f
0	0	f(0,0)
0	1	f(0,1)
1	0	f(1,0)
1	1	f(1,1)

encryption

$$H(k_0^A, k_0^B) \oplus k_{f(0,0)}^C$$
$$H(k_0^A, k_1^B) \oplus k_{f(0,1)}^C$$
$$H(k_1^A, k_0^B) \oplus k_{f(1,0)}^C$$
$$H(k_1^A, k_1^B) \oplus k_{f(1,1)}^C$$

XOR permutation
according to s_A, s_B

$$H(k_{s_A}^A, k_{s_B}^B) \oplus k_{f(s_A,s_B)}^C$$
$$H(k_{s_A}^A, k_{\overline{s_B}}^B) \oplus k_{f(s_A,\overline{s_B})}^C$$
$$H(k_{\overline{s_A}}^A, k_{s_B}^B) \oplus k_{f(\overline{s_A},s_B)}^C$$
$$H(k_{\overline{s_A}}^A, k_{\overline{s_B}}^B) \oplus k_{f(\overline{s_A},\overline{s_B})}^C$$

$(s_A, s_B) = (0,0)$ のとき, $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}$

$(s_A, s_B) = (0,1)$ のとき, $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$

$(s_A, s_B) = (1,0)$ のとき, $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}$

$(s_A, s_B) = (1,1)$ のとき, $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$

- **Evaluation Phase**: 入力値 $x_1, \ldots, x_n$ に対応するキー $k_{x_1}^{w_1}, \ldots, k_{x_n}^{w_n}$ と garbled circuit $\tilde{C}$ を受け取り出力ワイヤの値 $k_{f(x)}^{w_{out}}$ を計算
  各ワイヤでは一方のキーが得られる．各ゲートでは，キーの最下位ビット b_A, b_B を参照し特定した 1 つの値のみをデコードする.

$(b_A, b_B) = (0,0)$ のとき, 1st value

$(b_A, b_B) = (0,1)$ のとき, 2nd value

$(b_A, b_B) = (1,0)$ のとき, 3rd value

$(b_A, b_B) = (1,1)$ のとき, 4th value

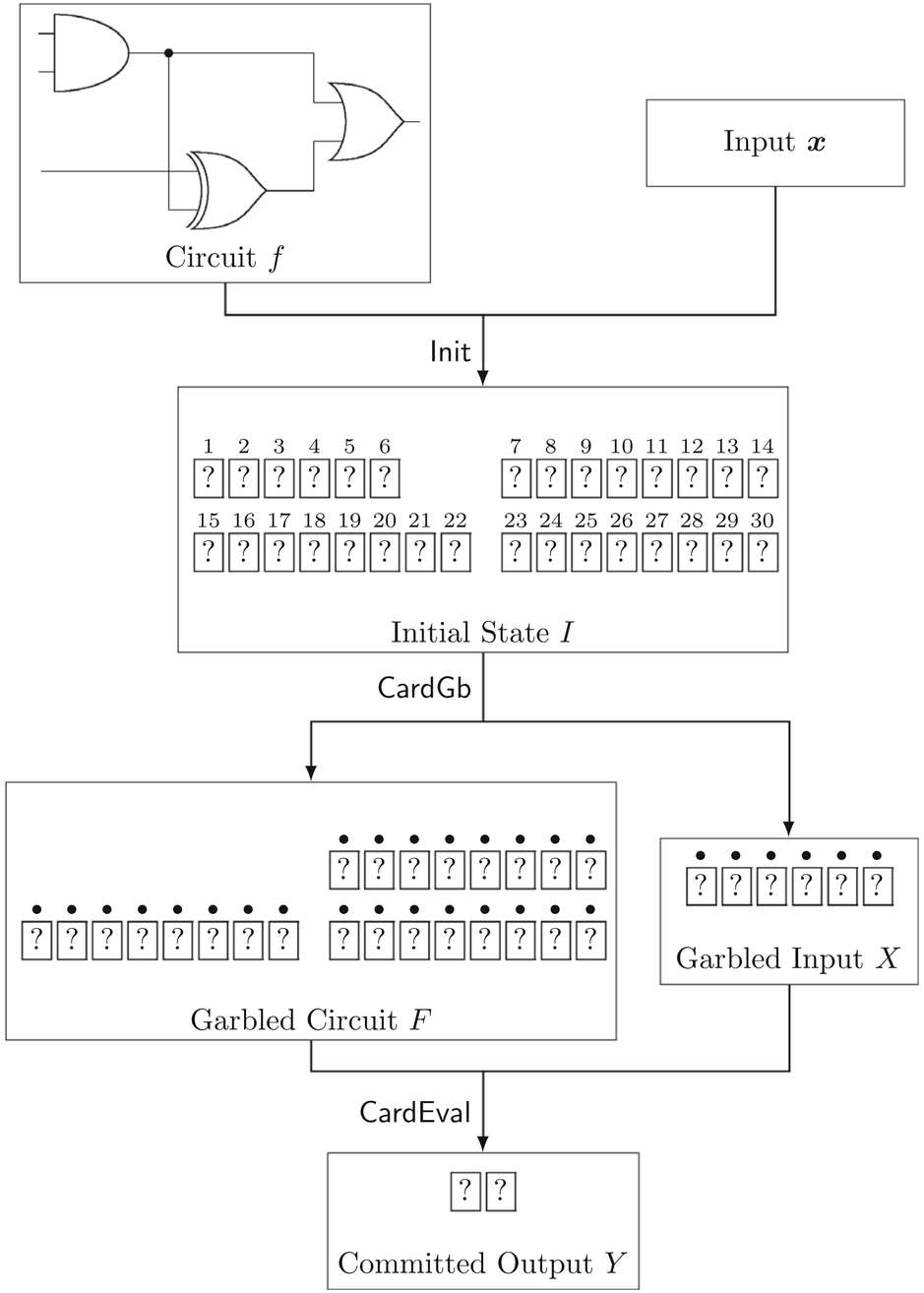
$x_A = b_A \oplus s_A$, $x_B = b_B \oplus s_B$ なので, デコードするセルは $H(k_{x_A}^A, k_{x_B}^B) \oplus k_{f(x_A,x_B)}^C$ であり，$k_{f(x_A,x_B)}^C$ を出力

Card-based Garbled Circuit

概要

- カードベース暗号を用いたガーブルド回路の実装 [SN21]
  - 任意の論理回路を計算可能
  - 計算に必要なシャッフル回数は1回
  - カード枚数は回路サイズに比例して増加

- 以下の３つの手続きから構成される：
  - **Initialization**:
    計算したい回路と入力値をカードを用いた表現にエンコードする
  - **Garbling**:
    Shuffle 操作を行うことで回路の意味を保ったままランダマイズする
  - **Evaluation**:
    Open 操作を行い Garbled Circuit の一部を公開することで所望の結果を計算する



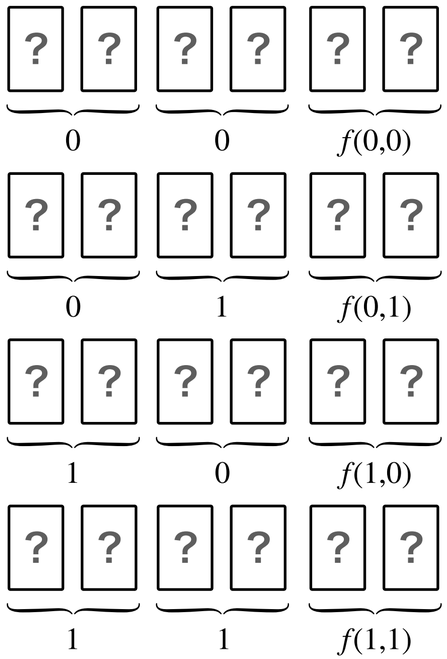
Card-based Garbled Circuit

Card-based Garbled Circuit [SN21] の概要

- **Initialization**: 回路 C をカード列の表現にエンコードする
  回路中の各ゲートについて，真理値表の全てのセルに対応するコミットメントを順番に並べる

x0	x1	f
0	0	f(0,0)
0	1	f(0,1)
1	0	f(1,0)
1	1	f(1,1)

encryption

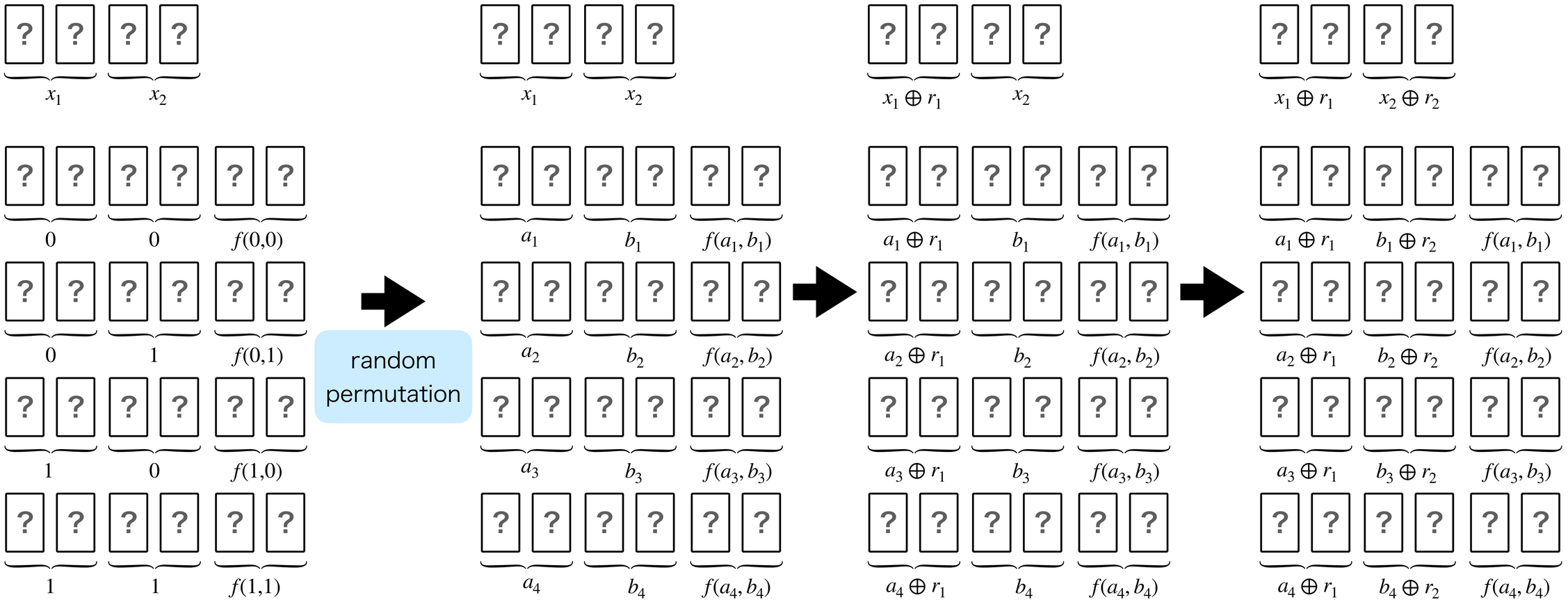


ゲートごとに追加カード24枚

Card-based Garbled Circuit

Card-based Garbled Circuit [SN21] の概要

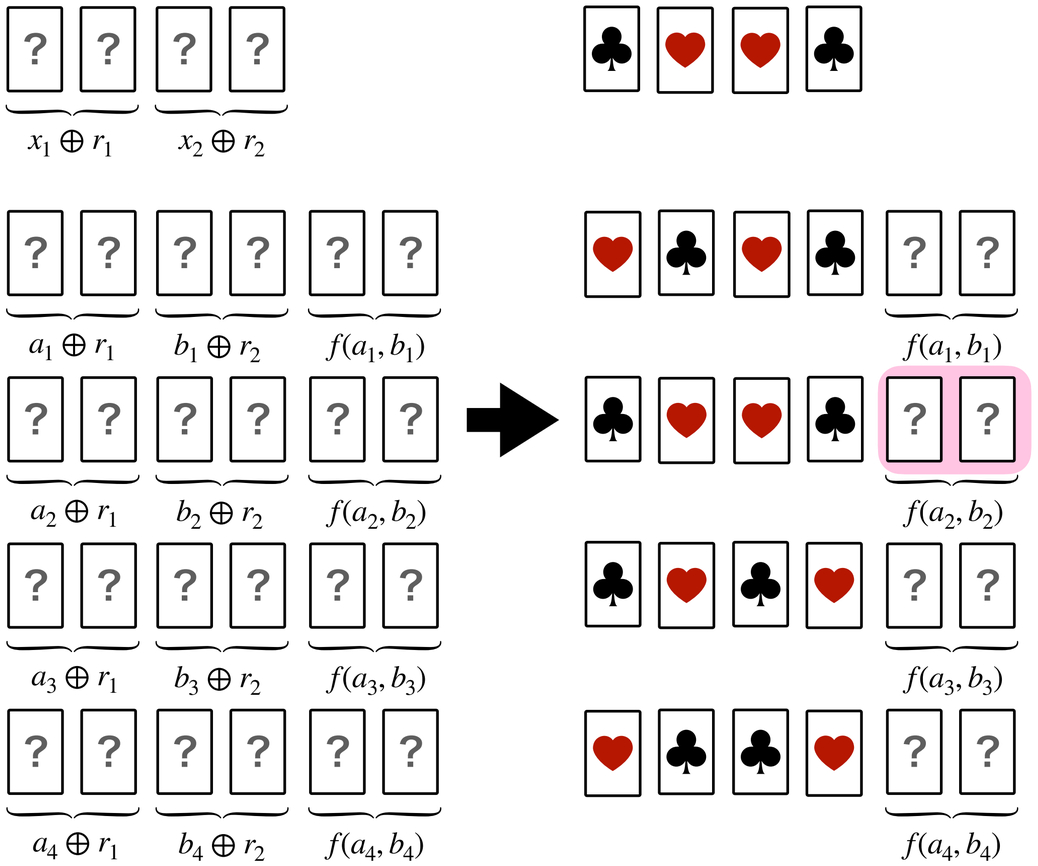
- **Garbling**: 入力値 $x_1, \ldots, x_n$ と回路 C のコミットメントをシャッフルし，Garbled circuit $\tilde{C}$ を計算
  各ゲートと各ワイヤに対し, pile-scramble shuffle を一度ずつ行う.



Card-based Garbled Circuit

Card-based Garbled Circuit [SN21] の概要

- **Evaluation**: garbled circuit の一部を Open し，所望のコミットメントを得る
  各ゲートにおいて，入力値と真理値表の入力列を Open する．入力と一致する行のコミットメントをゲートの出力とする．



$$x_1 \oplus r_1 = a_i \oplus r_1$$

$$x_2 \oplus r_2 = b_i \oplus r_2$$

$$f(a_i, b_i) = f(x_1, x_2)$$

提案手法

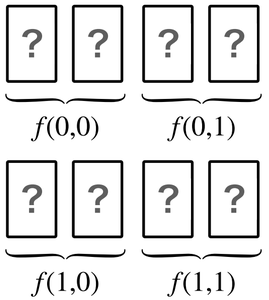
メインアイデア: Point-and-Permute Technique in Card-based Garbled Circuit

- **Initialization**:

  真理値表の演算結果のコミットメントのみを並べたカード列を各ゲートのエンコードとする

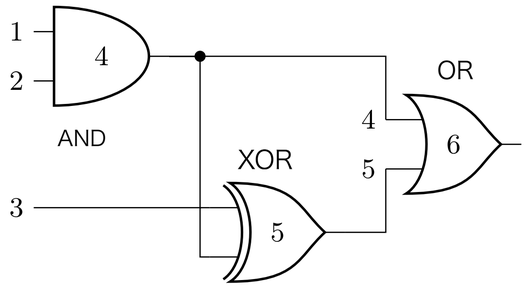
x0	x1	f
0	0	f(0,0)
0	1	f(0,1)
1	0	f(1,0)
1	1	f(1,1)

encryption

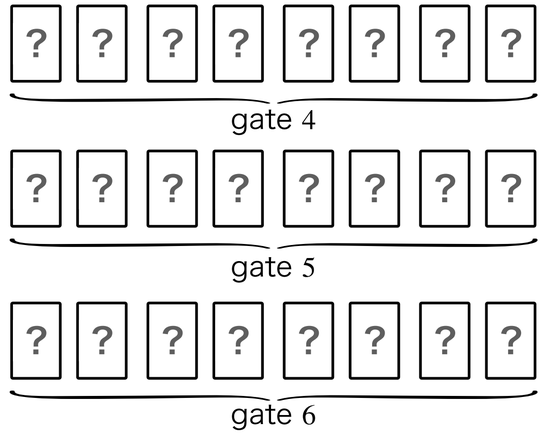


ゲートごとに追加カード8枚

回路中のすべてのゲートのエンコードを順番に並べたカード列を回路のエンコードとする



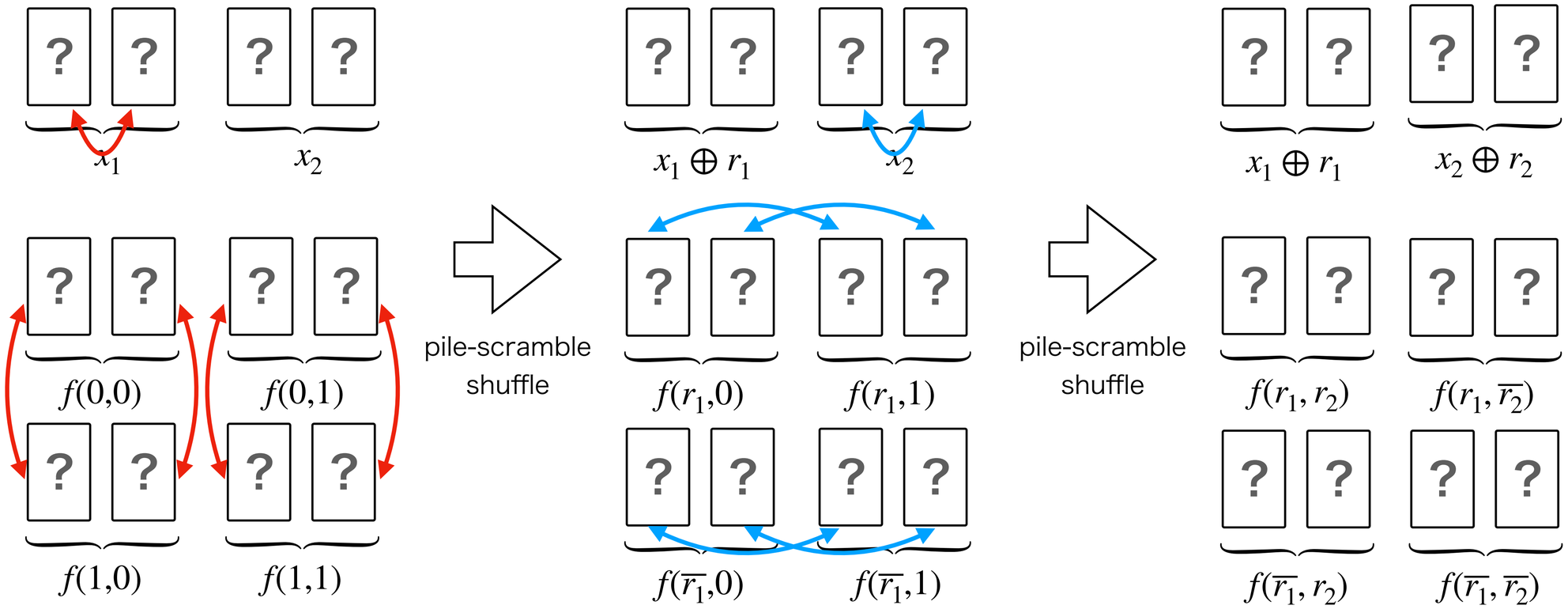
encryption



提案手法

メインアイデア: Point-and-Permute Technique in Card-based Garbled Circuit

- **Garbling**: 入力値 $x_1, \ldots, x_n$ と回路 C のコミットメントをシャッフルし，Garbled circuit $\tilde{C}$ を計算
  各ワイヤに割り当てられる値の候補と直後のゲートのカード列に対し, pile-scramble shuffle を一度ずつ行う.

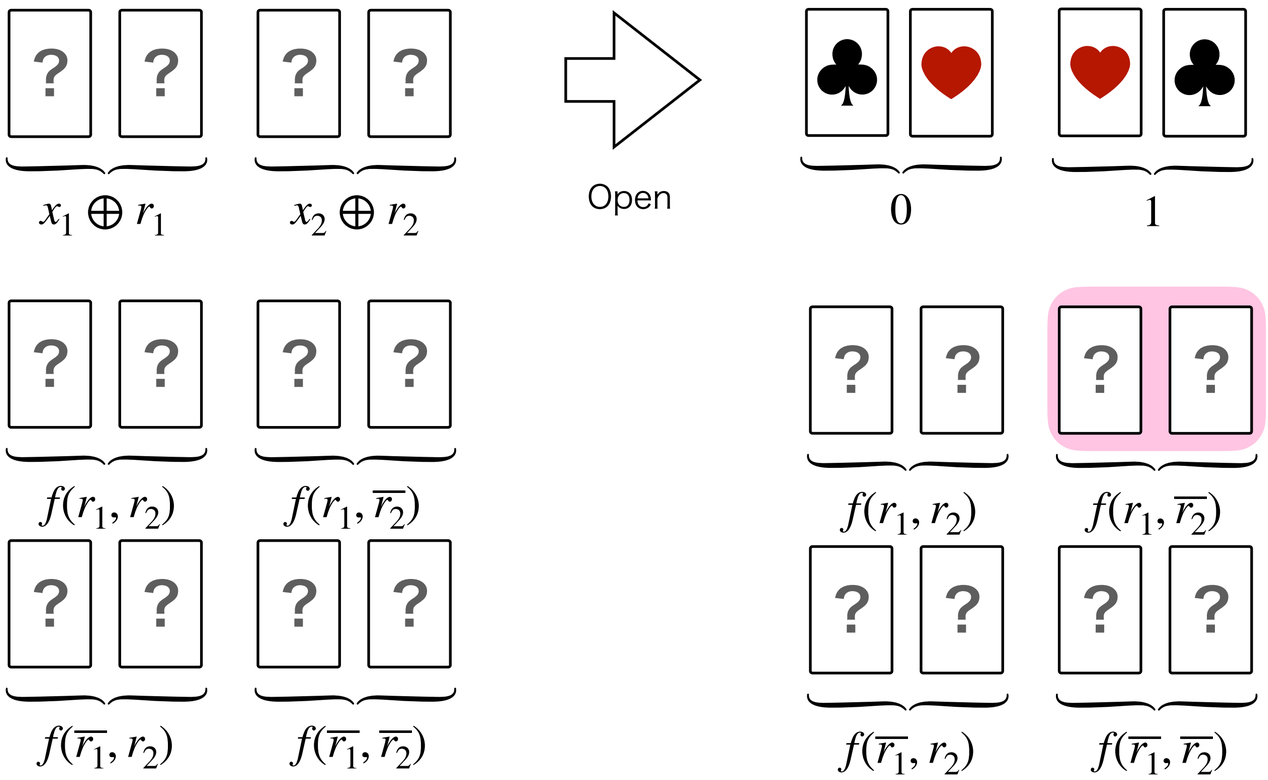


全体で XOR shuffle になっている

提案手法

メインアイデア: Point-and-Permute Technique in Card-based Garbled Circuit

- **Evaluation**: garbled circuit の一部を Open し，所望のコミットメントを計算
  各ゲートにおいて，Open された入力値に対応する位置のコミットメントをゲートの出力とする.



$(b_A, b_B) = (0,0)$ のとき, 1st value
$(b_A, b_B) = (0,1)$ のとき, 2nd value
$(b_A, b_B) = (1,0)$ のとき, 3rd value
$(b_A, b_B) = (1,1)$ のとき, 4th value

安全性

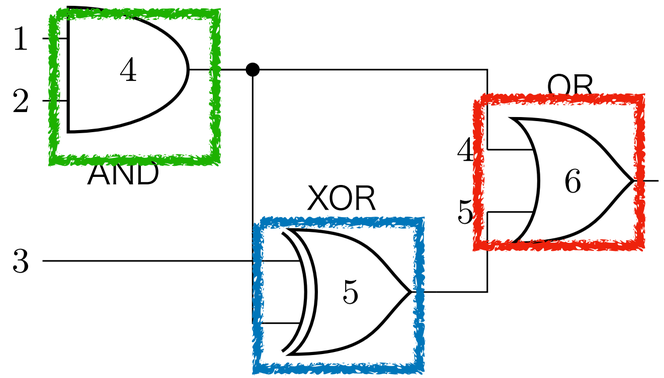
値 r_1, r_2 は一様ランダムなので,
公開される値は情報を漏らさない

正当性

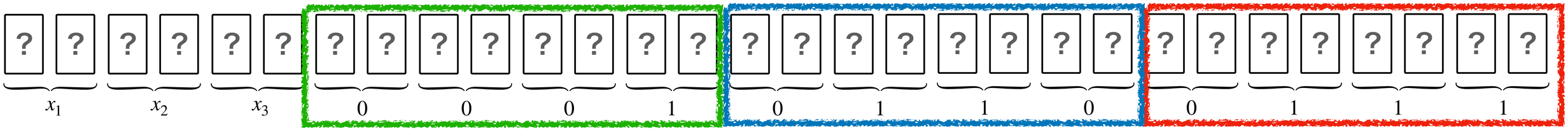
2度の pile-scramble shuffle は
XOR shuffle になっているため
$f(x_1, x_2)$ のコミットメントを出力

回路の計算例

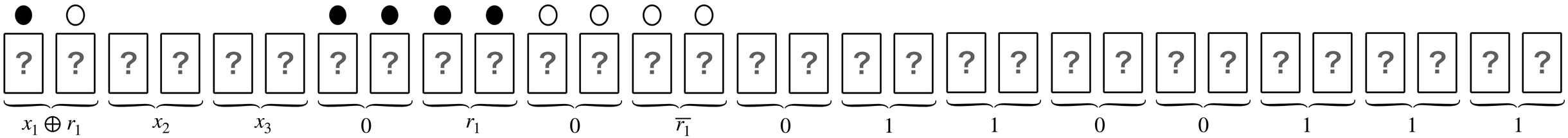
Garbling Phase



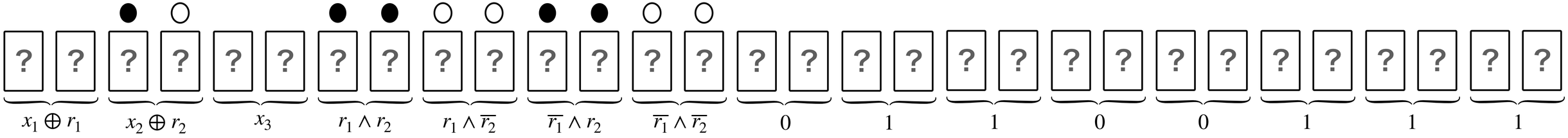
Initial state



shuffle for wire 1

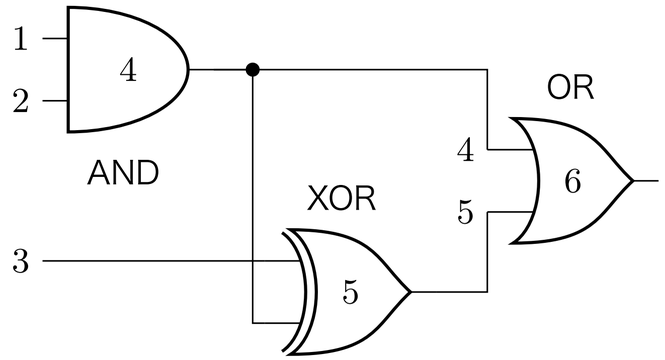


shuffle for wire 2

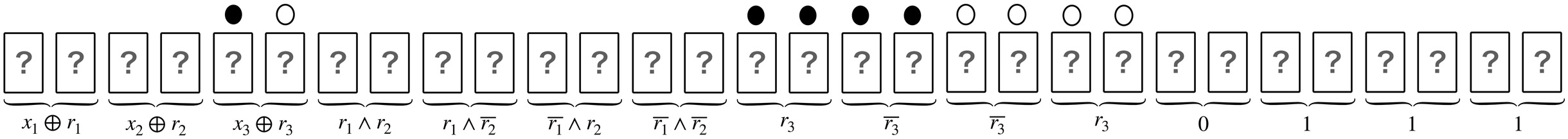


計算例

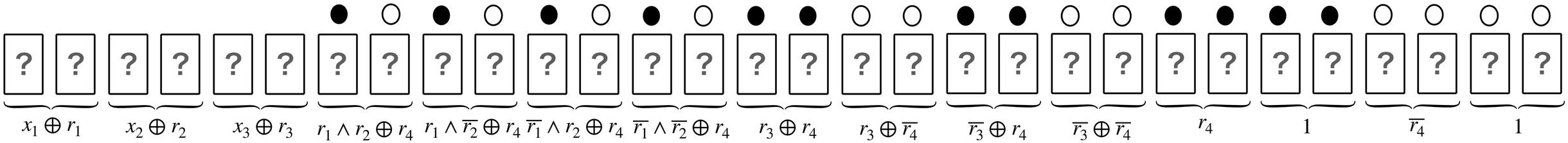
Garbling Phase



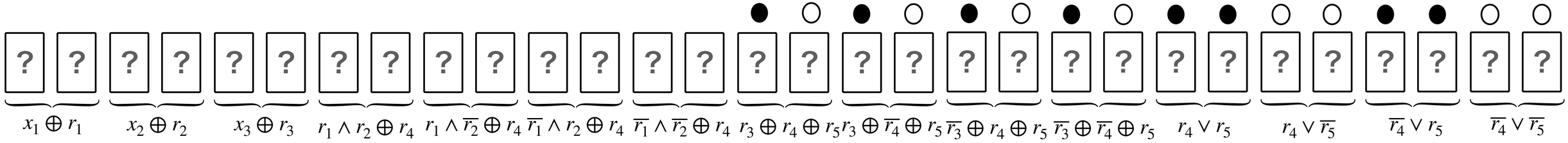
shuffle for wire 3



shuffle for wire 4

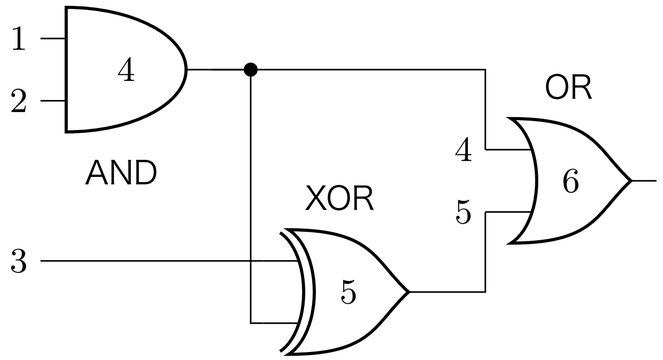


shuffle for wire 5

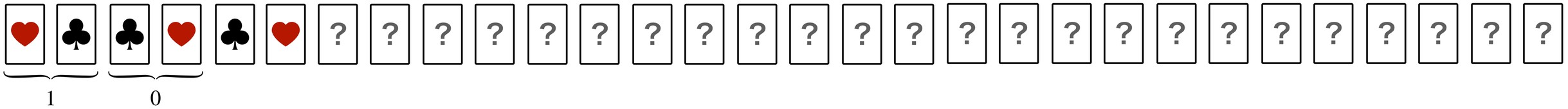


計算例

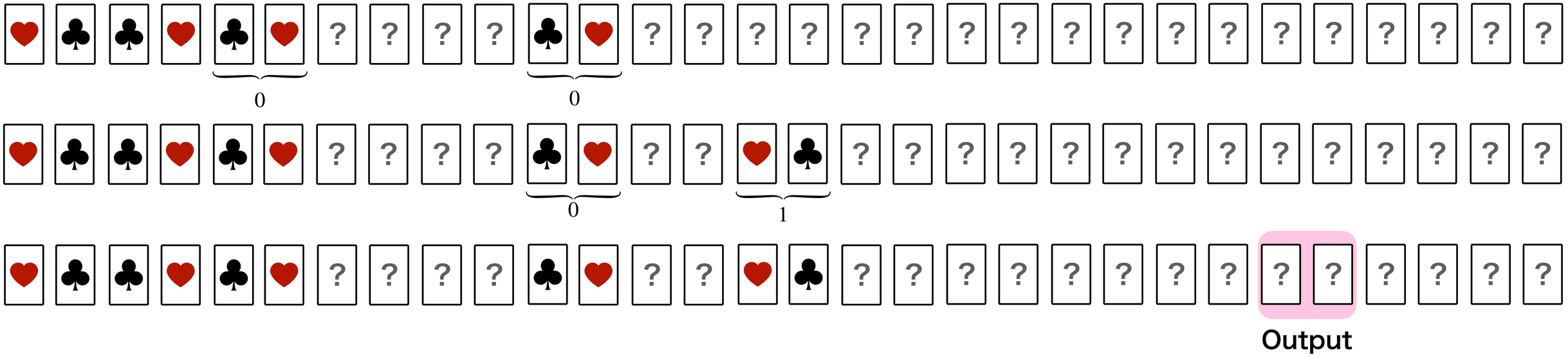
Evaluation Phase



Open the commitment of input wires

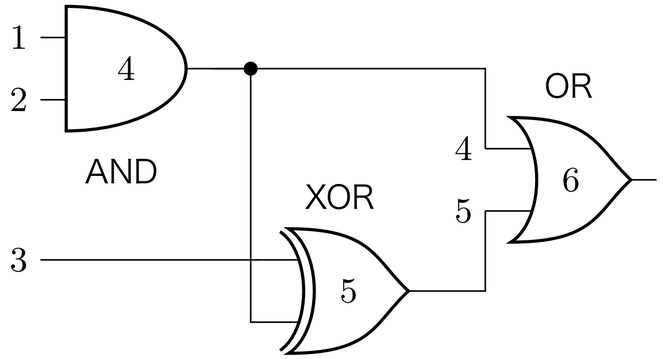


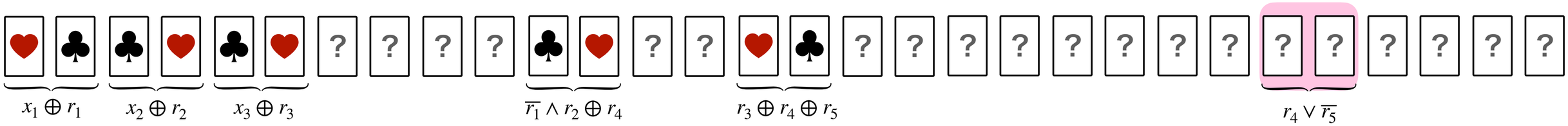
Evaluate each gate



計算例

検算





各 pile-scramble shuffle の定義より以下が成り立つ：

$1 = x_1 \oplus r_1$　　$0 = x_2 \oplus r_2$　　$0 = x_3 \oplus r_3$　　$0 = \overline{r_1} \wedge r_2 \oplus r_4$　　$1 = r_3 \oplus r_4 \oplus r_5$

従って,

$$r_4 \vee \overline{r_5} = r_4 \vee (r_3 \oplus r_4) = (\overline{r_1} \wedge r_2) \vee (r_3 \oplus \overline{r_1} \wedge r_2) = (x_1 \wedge x_2) \vee (x_3 \oplus x_1 \wedge x_2) = f(x_1, x_2, x_3)$$

提案手法

Pile-scramble Shuffle が一回の Shuffle 操作にまとめられること

- $n + g - m$ 回の pile-scramble shuffle は，ある置換群 G_f を用いて一回の shuffle 操作 (shuffle, G_f) で表現できる.

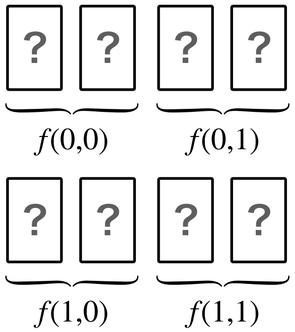
  ワイヤ i の pile-scramble shuffle が置換群 G_i に対応するとき, 置換群 G_f を以下で定義：

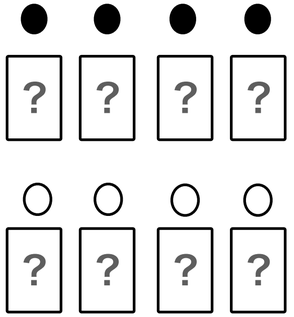
$$G_f := \{g_1 g_2 \cdots g_{n+g-m} \in S_{2n+8g} \mid g_j \in G_j\}$$

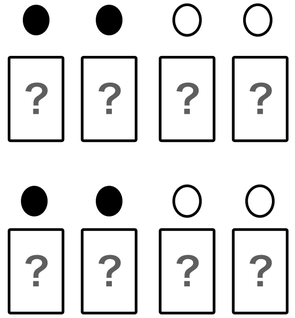
- G_f が群になっていることの証明の概略.

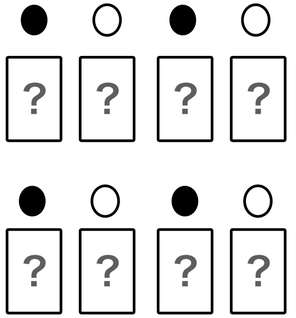
  すべての順列対 $g_i \in G_i$, $g_j \in G_j$ に対し, $g_i g_j = g_j g_i$ であることを示せばよい.

  これは各ゲートに作用する 3 回の pile-scramble shuffle が（定義より）可換であることから従う.







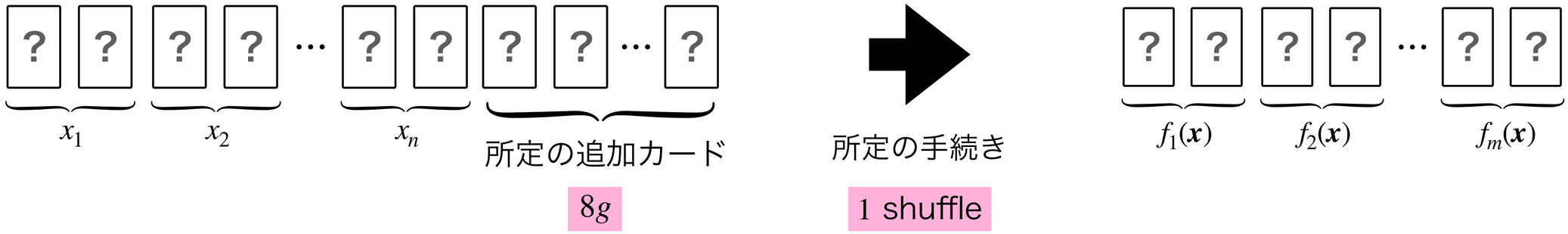


結果 (UCNC ’23)

Efficient Card-based Protocol for Any Boolean Function

f をブール関数，C を入力数 n, ゲート数 g, 出力数 m の論理回路で f を計算するものとする.

任意のブール関数 f を秘匿計算する効率の良いカードベースプロトコルを構成した.



- プロトコル中の shuffle の回数は合計で 1 回にできる. （時間計算量）
- プロトコル実行に必要なカードの枚数は合計で $2n + 8g$. （空間計算量）
- 同様の機能を実現している single-shuffle プロトコル [SN21]と比較して，追加カードの枚数を $\frac{1}{3}$ に削減している. ($2n + 24g$ 枚から $2n + 8g$ 枚)

[SN21] K. Shinagawa and K. Nuida. A single shuffle is enough for secure card-based computation of any boolean circuit. Discrete Applied Mathematics, 289:248–261, 2021.

カードベースガーブルド回路の発展

比較

カードベース ガーブルド回路	ゲート種類	AND	XOR	シャッフル	Batching
[SN21]	非公開	24	24	一様閉	✓
Point-and-permute [Ours @UCNC23]	非公開	8	8	一様閉	
Free-XOR [MS23]	**公開**	**8**	**0**	**一様閉**	
Single-card encoding [OSNWI24]	**公開**	**6**	**6**	**一様**	

[MS23] Y. Manabe and K. Shinagawa. Free-XOR in Card-Based Garbled Circuits. CANS 2023.

[OSNWI24] T. Ono, K. Shinagawa, T. Nakai, Y. Watanabe and M. Iwamoto. A single shuffle is enough for secure card-based computation of any boolean circuit. ICISC 2023.

結果 (Natural Computing ’25)

カードベースガーブルド回路の拡張

UCNC’23 で提案したカードベースガーブルド回路に対し，以下の２つの自然な拡張を行った

(1) **多入力ゲート**への対応

２入力ゲートだけでなく，任意の入力個数のゲートに対応する汎用的構成を与えた

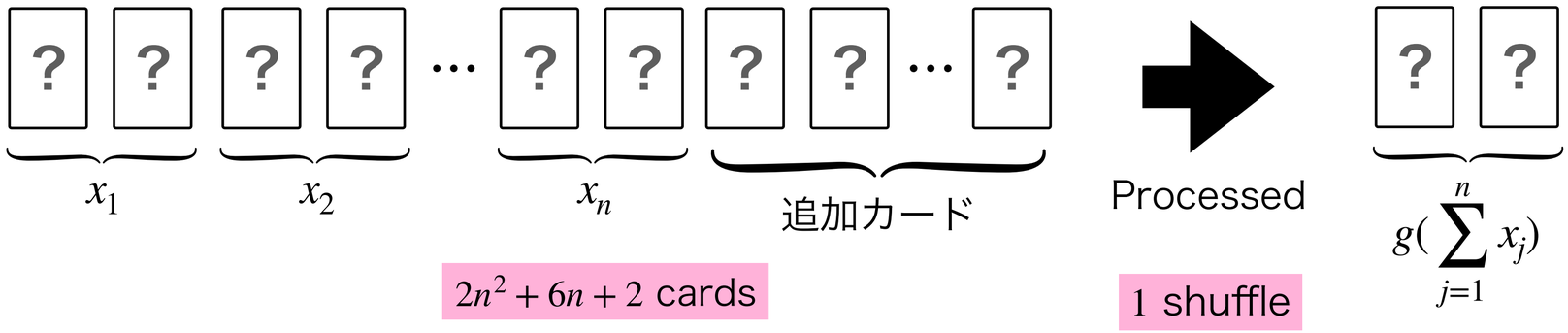
k 入力ゲートは 2^{k+1} 枚のカードで実現可能

(2) **算術ゲート**への対応

論理ゲートだけでなく，加算ゲートなどの算術演算に対応

Free-XOR 手法を一般化することで Free-AND が可能であることを示した

- 応用：**対称ブール関数**を計算するカードベースプロトコル



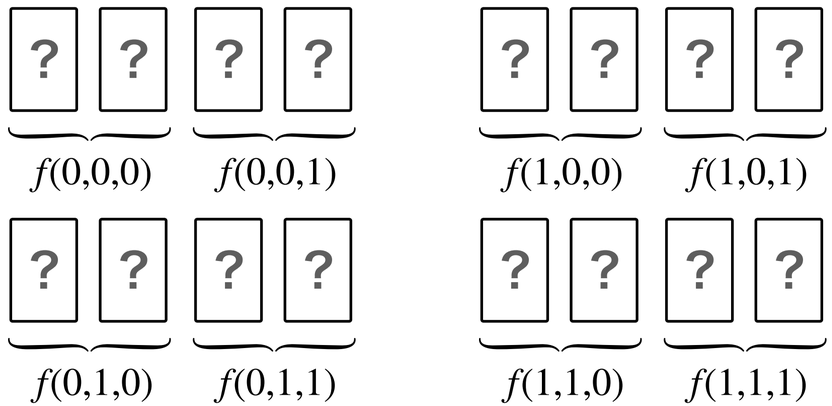
カードベースガーブルド回路の拡張

(1)多入力ゲートのサポート

- **Initialization**: 回路 C をエンコードする. 各ゲートに対応する真理値表の演算結果のコミットメントのみを並べる.

x0	x1	x2	f
0	0	0	f(0,0,0)
0	0	1	f(0,0,1)
0	1	0	f(0,1,0)
0	1	1	f(0,1,1)
1	0	0	f(1,0,0)
1	0	1	f(1,0,1)
1	1	0	f(1,1,0)
1	1	1	f(1,1,1)

encryption

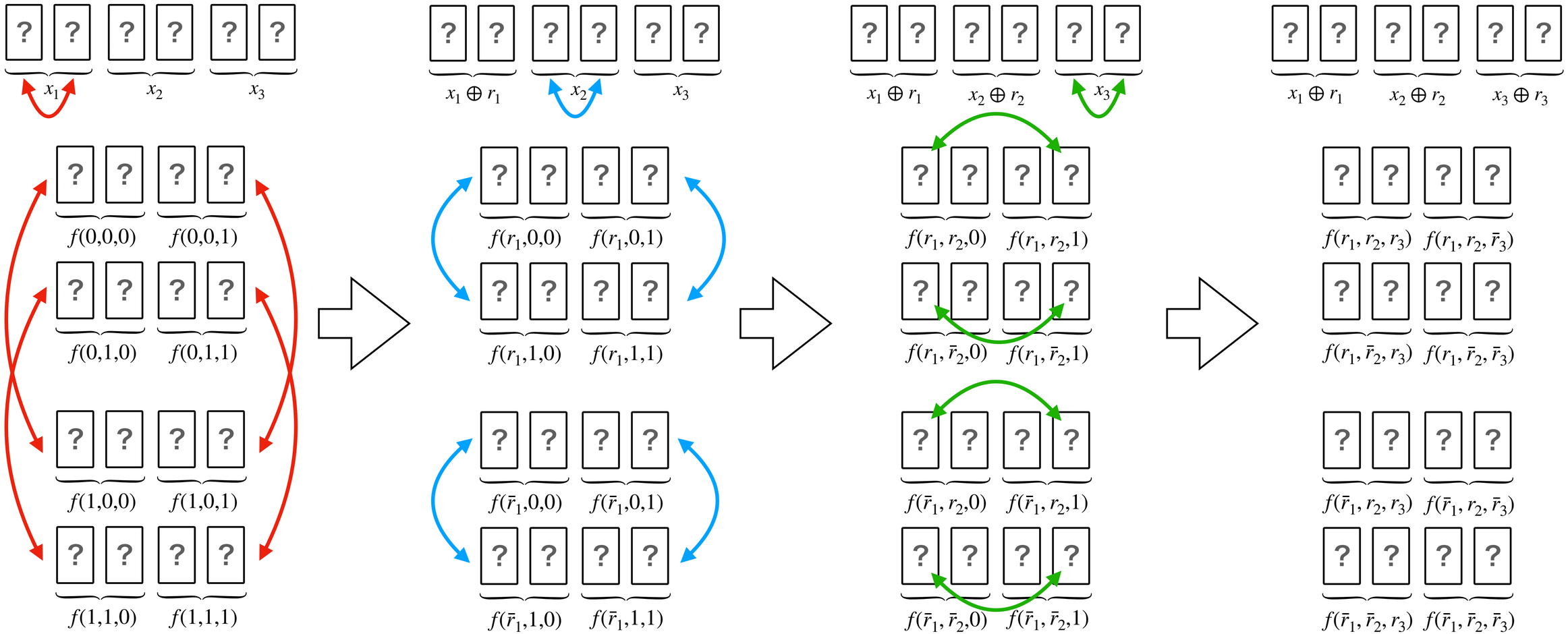


k 入力ゲートごとに追加カード 2^{k+1} 枚

カードベースガーブルド回路の拡張

(1)多入力ゲートのサポート

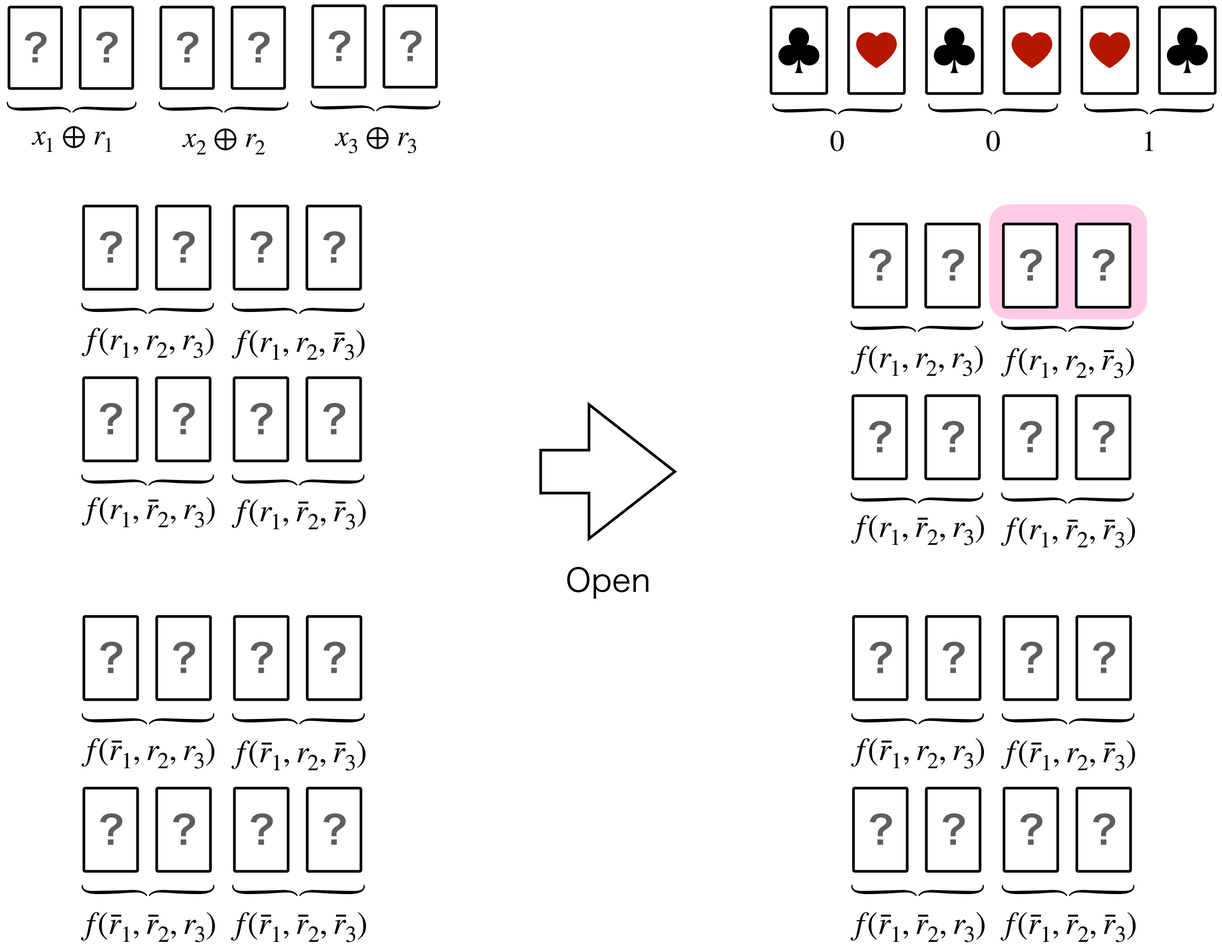
- **Garbling**: 各ワイヤに対し pile-scramble shuffle を一回実行



カードベースガーブルド回路の拡張

(1)多入力ゲートのサポート

- **Evaluation**: garbled circuit の一部を Open し，所望のコミットメントを得る



$(b_A, b_B, b_C) = (0,0,0)$ のとき，1st value
$(b_A, b_B, b_C) = (0,0,1)$ のとき，2nd value
$(b_A, b_B, b_C) = (0,1,0)$ のとき，3rd value
$(b_A, b_B, b_C) = (0,1,1)$ のとき，4th value
$(b_A, b_B, b_C) = (1,0,0)$ のとき，5th value
$(b_A, b_B, b_C) = (1,0,1)$ のとき，6th value
$(b_A, b_B, b_C) = (1,1,0)$ のとき，7th value
$(b_A, b_B, b_C) = (1,1,1)$ のとき，8th value

カードベースガーブルド回路の拡張

(2)算術ゲートのサポート

- **論理ゲート:** 入力・出力がすべてブール値であるゲート

x1	x2	AND(x1,x2)
0	0	0
0	1	0
1	0	0
1	1	1

x1	x2	XOR(x1,x2)
0	0	0
0	1	1
1	0	1
1	1	0

- **算術ゲート:** 入力もしくは出力が数値であるゲート

2 ブール値入力・数値出力のゲート

x1	x2	f(x1,x2) : Int
0	0	3
0	1	0
1	0	0
1	1	2

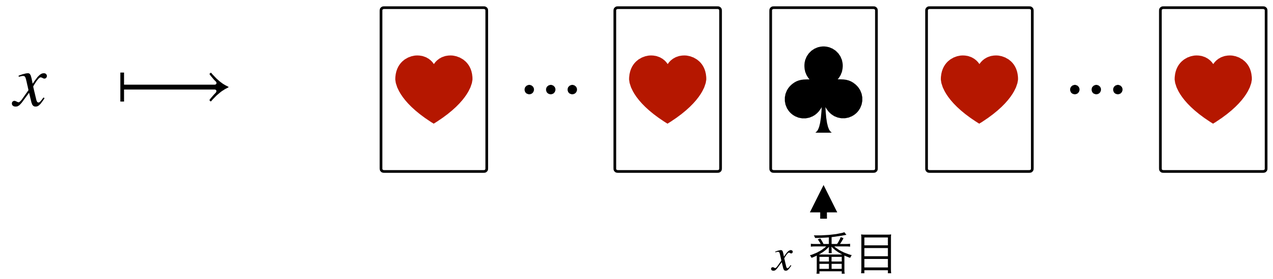
1 数値入力・ブール値出力のゲート

x : Int	f(x)
0	1
1	0
2	1

カードベースガーブルド回路の拡張

(2)算術ゲートのサポート

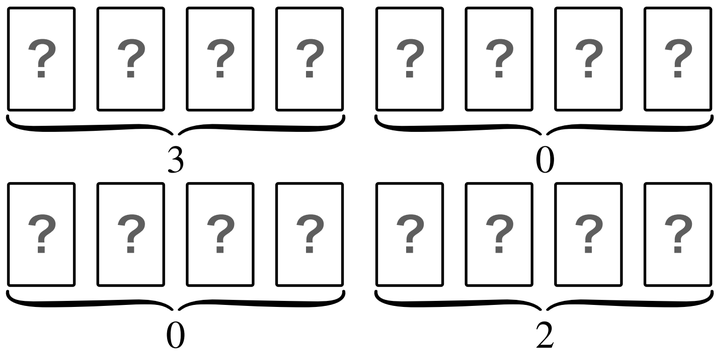
- **position encoding**: 数値 $x \in \mathbb{Z}_N$ を N 枚のカードで表現する.



- **Initialization**: 各ゲートに対応する真理値表の演算結果のコミットメントのみを用いる.

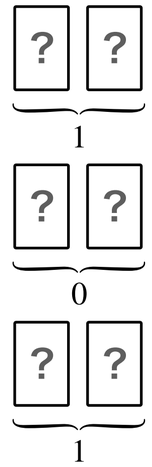
２ブール値入力・数値出力のゲート

x0	x1	f(x0,x1)
0	0	3
0	1	0
1	0	0
1	1	2



１数値入力・ブール値出力のゲート

x0	f(x0) : Bool
0	1
1	0
2	1



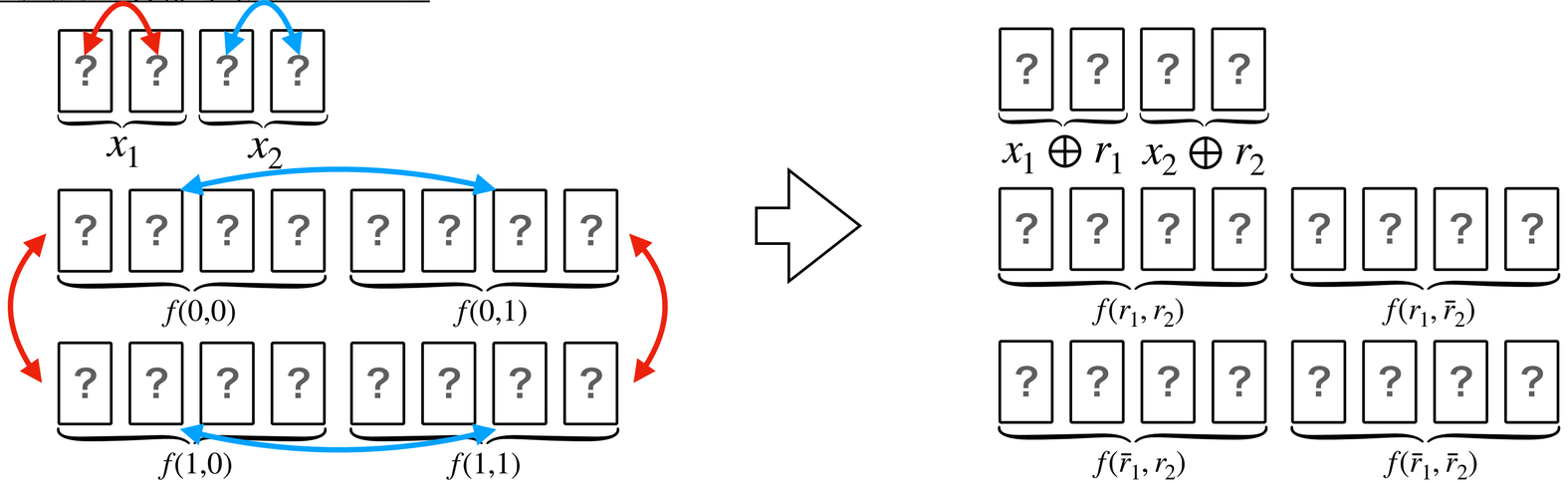
関数 $f\colon \mathbb{Z}_M^k \to \mathbb{Z}_N$を計算するゲートに対し，追加カード $M^k N$ 枚

カードベースガーブルド回路の拡張

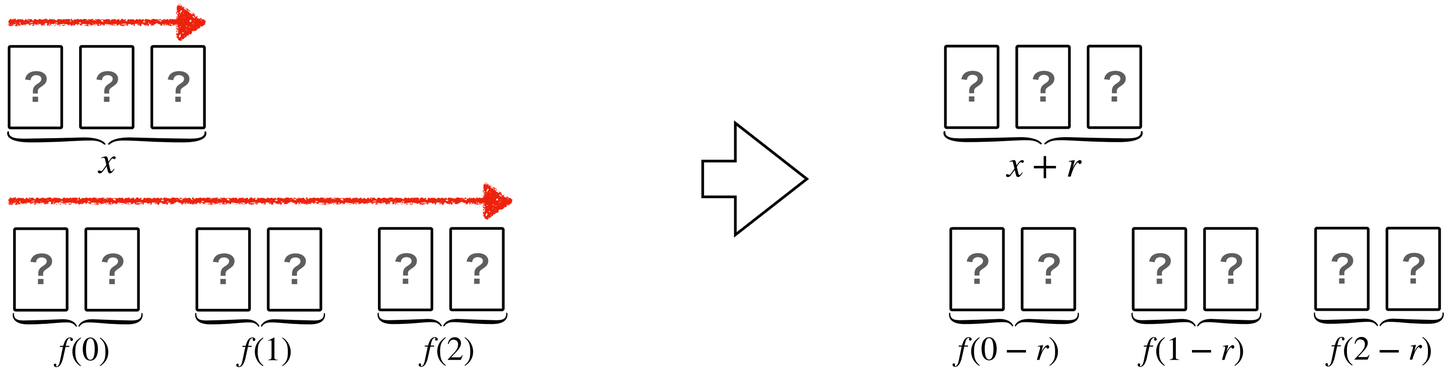
(2)算術ゲートのサポート

- **Garbling**: 各ワイヤについて pile-shifting shuffle を一回実行する

２ブール値入力・数値出力のゲート



１数値入力・ブール値出力のゲート

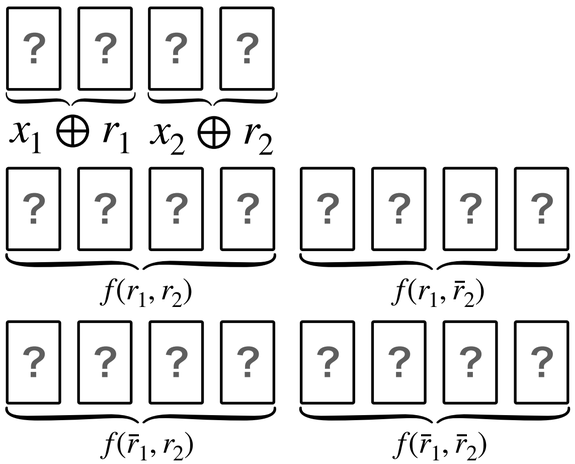


カードベースガーブルド回路の拡張

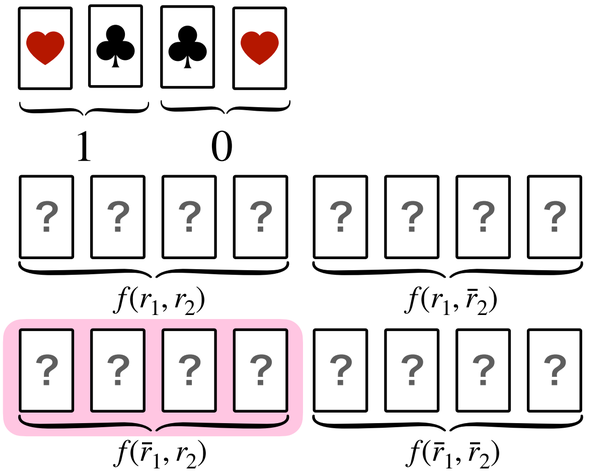
(2)算術ゲートのサポート

- **Evaluation**: garbled circuit の一部を Open し，所望のコミットメントを得る

２ブール値入力・数値出力のゲート

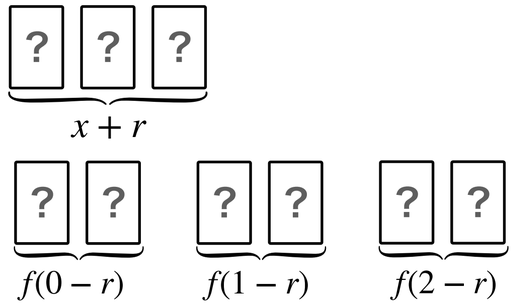


Open

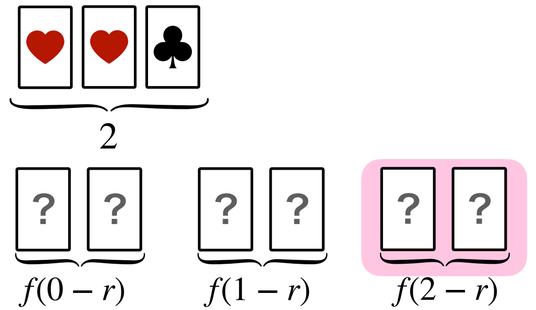


$(b_A, b_B) = (0,0)$ のとき，1st value
$(b_A, b_B) = (0,1)$ のとき，2nd value
$(b_A, b_B) = (1,0)$ のとき，3rd value
$(b_A, b_B) = (1,1)$ のとき，4th value

１数値入力・ブール値出力のゲート



Open



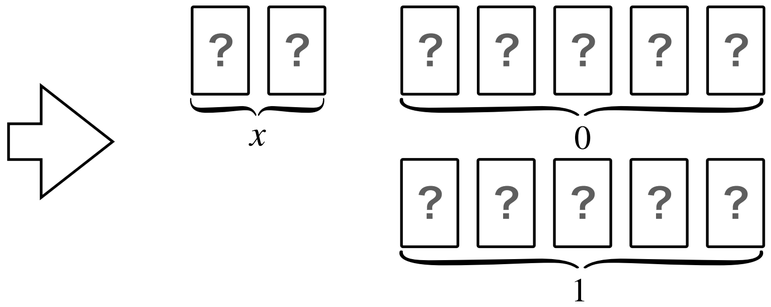
$v = 0$ のとき，1st value
$v = 1$ のとき，2nd value
$v = 2$ のとき，3rd value

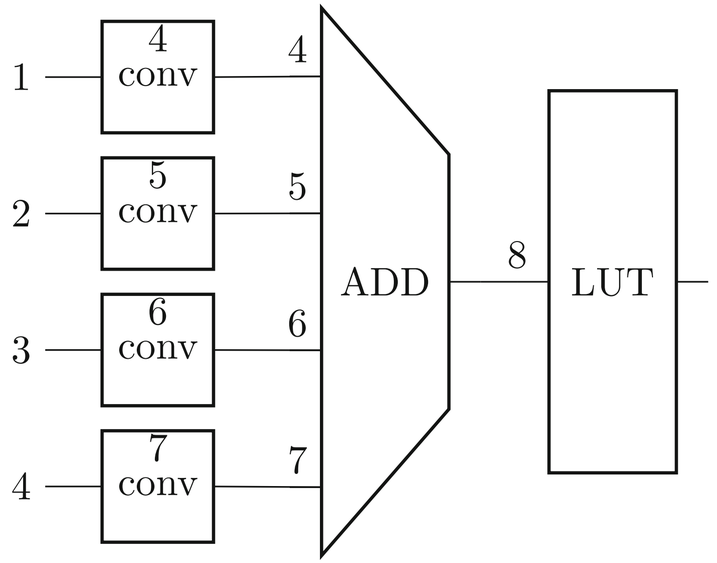
応用：対称ブール関数評価プロトコル

構成の概要

- 対称ブール関数評価は右図の拡張回路で書ける
  - conv はブール値を数値に変換するゲート

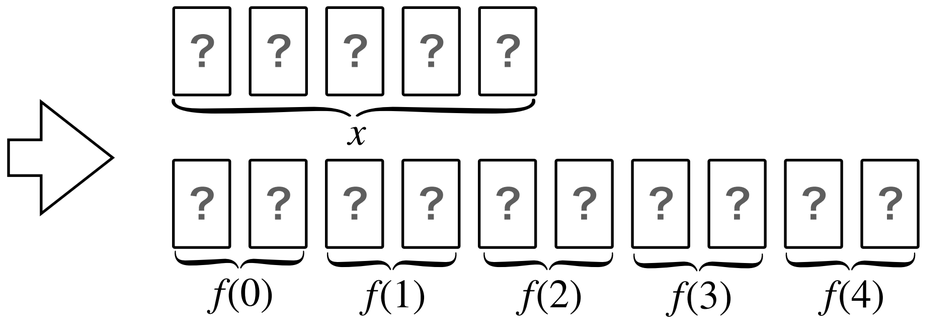
x : Bool	x : Int
0	0
1	1





  - ADD ゲートは Free-XOR 手法[MS23]を一般化することで追加カードなしで実現
  - LUT はテーブル引きのゲート

x : Int	f(x): Bool
0	f(0)
1	f(1)
2	f(2)
3	f(3)
4	f(4)



追加カード $2n^2 + 6n + 2$ 枚

結果 (Natural Computing ’25)

カードベースガーブルド回路の拡張

UCNC’23 で提案したカードベースガーブルド回路に対し，以下の２つの自然な拡張を行った

(1) **多入力ゲート**への対応

２入力ゲートだけでなく，任意の入力個数のゲートに対応する汎用的構成を与えた

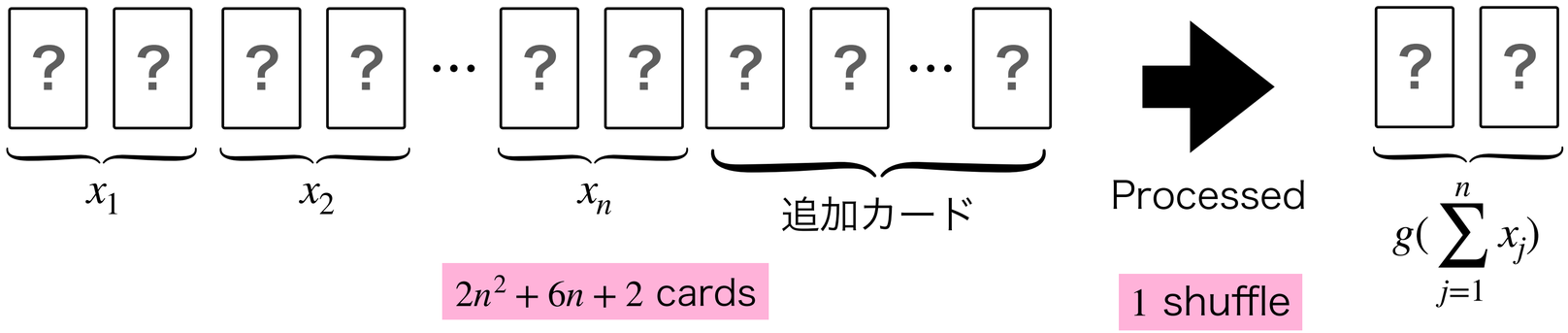
k 入力ゲートは 2^{k+1} 枚のカードで実現可能

(2) **算術ゲート**への対応

論理ゲートだけでなく，加算ゲートなどの算術演算に対応

Free-XOR 手法を一般化することで Free-AND が可能であることを示した

- 応用：**対称ブール関数**を計算するカードベースプロトコル



今後の課題

- Garbled circuit の他の最適化手法はカードベース暗号で表現できる？
  - GRR3, GRR2, fleXOR, half gates, garbled gadgets, three-halves garbling, one-hot garbling …
  - カードベースガーブルド回路における garbled AND ゲートのサイズの下限は？

- 対称ブール関数プロトコルは効率化できる？
  - 現在のカードベースプロトコル：カード枚数 $O(n^2)$
  - 一方で，対称ブール関数を評価する回路の複雑性は $\Omega(n)$
    - ビット列のソート：回路サイズ $\Omega(n)$ [LSX19]
    - ソート列を使ってLUT：回路サイズ $O(n)$

[LSX19] W. Lin, E. Shi, and T. Xie. Can We Overcome the nlogn Barrier for Oblivious Sorting? SODA 2019.