Computer Security Symposium 2018
22 - 25 October 2018

プレゼント交換に適したシンプルなカードベース置換生成

齋藤 敬宏[1]　千田 栄幸[2]　水木 敬明[3]

概要：n 人のプレーヤーがいて，プレゼント交換を行いたい場面を考える．すなわち，不動点を持たない置換をランダムに生成したいとする．物理的なカード組を用いると，そのような置換生成ができることが知られている．例えば，2016 年に Ibaraki と Manabe は，巡回シャッフル等を用いる置換生成プロトコルを提案している．本稿では，Ibaraki と Manabe のプロトコルは不動点を持たない置換を一様ランダムに生成するわけではなく，不動点を持たない長さ n のサイクルを一様ランダムに出力すること，及びプレゼント交換の実用上はこの置換生成で問題がないことを指摘するとともに，彼らのプロトコルを改良し，より少ないシャッフル回数で同じ機能をシンプルに実現できることを示す．

キーワード：カードベースプロトコル，不動点，ランダム置換，プレゼント交換

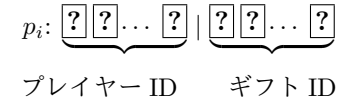
A Simple Card-based Generation of a Random Permutation Suitable for Exchange of Gifts

Takahiro Saito[1]　Eikoh Chida[2]　Takaaki Mizuki[3]

1. はじめに

n 人のプレイヤーがいて，プレゼント交換を行いたい場面を考える．各々のプレイヤーが自分自身の贈り主になるのは避けたいので，不動点 (fixed point) を持たない置換をランダムに生成したい．

この問題に対して 2016 年に Ibaraki と Manabe [5] は裏面が同一の模様である 2 色のカードを用いる置換生成プロトコルを提案した．そのプロトコルは各プレイヤー $p_i (1 \leq i \leq n)$ に対して $2\lceil \log n \rceil$ 枚[*1]のカードからなるプレイヤー ID，同じく $2\lceil \log n \rceil$ 枚のギフト ID を用意する．



1 人分のカードの束が合計 $4\lceil \log n \rceil$ 枚であるのでプレイヤー人数が n 人の場合にはカードの枚数は $4n\lceil \log n \rceil$ となる．またプロトコルの途中で不動点チェックを行うため使用するカードの総数は $4n\lceil \log n \rceil + 6$ 枚となる．シャッフルは Pile-Scramble シャッフルと巡回シャッフルが用いられているが詳細は後節で説明する．

1.1 本稿の内容

Ibaraki と Manabe が提案したプロトコルによって生成される置換は一様ランダムではない．すなわち出力される置換は，不動点を持たない置換からなる集合に一様分布しているわけではない．なぜなら，詳細は後述するが彼らのプロトコルはシャッフルに巡回シャッフルを用いており，長さ n のサイクルを一様ランダムに出力しているためである．しかし，プレゼント交換の実用上は，このような置換生成でも問題がない，あるいはむしろ好ましいことを本稿で指摘する．また彼らのプロトコルを改良し，より少ない

1 一関高専 専攻科生産工学専攻
Advanced Course of Prouduction Engineering，National Institute of Technology, Ichinoseki College.
2 一関高専 未来創造工学科情報・ソフトウェア系
Department of Engineering for Future Inovation，Division of Computer Engineering and Informatics，National Institute of Technology Ichinoseki College.
3 東北大学 サイバーサイエンスセンター
Cyberscience Center，Tohoku University.

*1 本文で用いる log の底は全て 2 とする．

カード枚数とシャッフル回数でシンプルに，不動点を持たない長さ n のサイクルを一様ランダムに生成できることを示す．

1.2 関連研究

本稿では，不動点を持たない長さ n のサイクルを一様ランダムに出力することを目的にしているが，不動点を持たない置換をランダムに生成する手法が文献 [1] [2] において確立されている．

2. カードベースプロトコル

本節では本稿で取り扱うカードとプロトコルについて説明する．

2.1 カード

カードには表面が ♣ と ♡ の 2 種類のものを用いる．裏面の模様はどちらのカードも ? とし，裏面のみでは表面の模様がどちらかはわからないものとする．またプロトコルに用いるカードは全て同じサイズと重さで傷等はついておらず模様以外ではカードの特定は困難であるとする．

2.2 エンコーディング

2 枚の組のカードに対して，2 進数の 0 と 1 を対応させるため次のように符号化を行う．

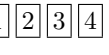
また裏返された 2 枚組のカードを**コミットメント**と呼ぶ．

コミットメント 1 つが 2 進数 1 ビットに対応する．またカードの左右をいれかえる操作を行うと，NOT 演算を行うことができる．

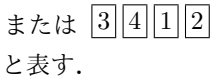
2.3 ランダム二等分割カット [3]

偶数枚のカード列に対して行うことができるシャッフルである．カード列を二等分し，左右をランダムに入れ替える．

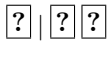
例えば次のような 4 枚のカード列に対して実行すると



結果は確率 1/2 で



となる．以降は [· | ·] と表す．



2.4 Pile-Scramble シャッフル [5]

m は ℓ で割り切れるとする．全部で m 枚のカードに対して，それを同枚数づつの束に分ける．分けた束の数が ℓ 個である時，$\ell!$ 通りの束の置換からランダムに 1 つの置換が選ばれるシャッフルのことを指す．クリップや封筒を用いることで容易に実装することができる．

2.5 巡回シャッフル

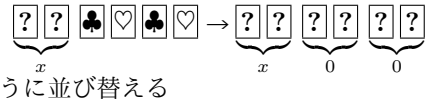
ℓ 個のカードの束を巡回させるシャッフルである．置換の総数は ℓ 通りである．例えばカードの束 $k = (k_0, k_1, \cdots, k_5)$ があるとき置換のパターンは 6 とおりである．

$(k_0, k_1, k_2, k_3, k_4, k_5)$，$(k_1, k_2, k_3, k_4, k_5, k_0)$，
$(k_2, k_3, k_4, k_5, k_0, k_1)$，$(k_3, k_4, k_5, k_0, k_1, k_2)$，
$(k_4, k_5, k_0, k_1, k_2, k_3)$，$(k_5, k_0, k_1, k_2, k_3, k_4)$．

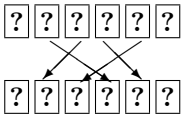
2.6 コピープロトコル

1 つのコミットメントに 4 枚のカードを追加してコミットメントのコピーを得るプロトコル [3] である．

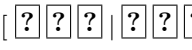
(1) カードを次のように置き，すべて裏返す



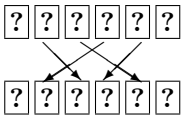
(2) 次のように並び替える



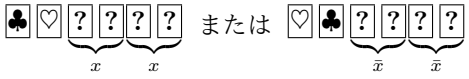
(3) カード列に対してランダム二等分割カットを適用する．



(4) 次のように並び替える



(5) 左側の 2 枚を表にする．

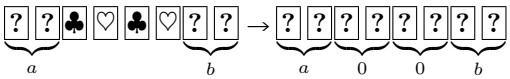


このように x のコミットメントをコピーすることができる．もし左側の 2 枚が ♡ ♣ だった場合にはコミットメントの左右を入れ替える NOT 演算により x のコミットメントを得ることができる．

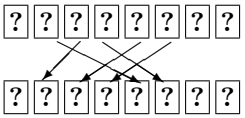
2.7 1 入力保存の AND プロトコル

2 つのコミットメントの AND 演算をしつつ，入力の 1 つが保存できるプロトコルである．これは既存の AND プロトコル [3] と半加算器プロトコル [4] を応用することで実現できることが知られている．入力 a と b に対して，a を保存する場合として紹介する．

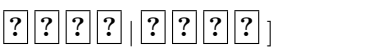
(1) a と b のコミットメント，さらに 4 枚のカードを次のように置く．



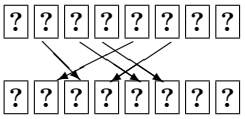
(2) 次のように並び替える．



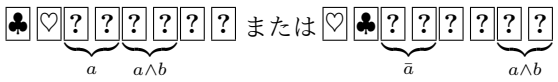
(3) カード列に対してランダム二等分割カットを実行する．



(4) 次のように並び替える．



(5) 左側の 2 枚を表側にする．左側のコミットメントの値によって出力は以下のようになる．

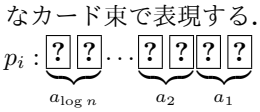


これにより $a \wedge b$ のコミットメントとともに a のコミットメントも得られる．なお残っている裏になっている 2 枚のカードはシャッフルしてから開くことにより，以降自由なカード ♣ ♡ として使える．

2.8 不動点チェック

これまで紹介したプロトコルを利用して，与えられた置換に不動点がないかチェックする方法 [2] を紹介する．

まず，カードを使ってどのように置換を表現するのかについて説明する．置換を π とし，$\pi(i)$ をプレイヤー p_i に対応させ次のようなカード束で表現する．



ただし，

$$(\pi(i)-1)_{10} = (a_{\log n}, \cdots a_2 a_1)_2$$

とし，添え字の $\log n$ は $\lceil \log n \rceil$ である．いま，i の二進表現を

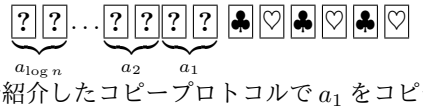
$$(i-1)_{10} = (b_{\log n}, \cdots b_2 b_1)_2$$

とするとき，$\pi(i)$ が不動点でないためには次の式を満たす必要がある．

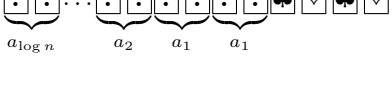
$$(a_1 \oplus \overline{b_1}) \wedge (a_2 \oplus \overline{b_2}) \wedge \cdots \wedge (a_{\log n} \oplus \overline{b_{\log n}}) = 0$$

この式を a の値を秘密にしたまま計算するための具体的な手順を紹介する．

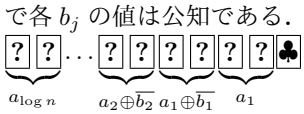
(1) 6 枚のカードを追加する．



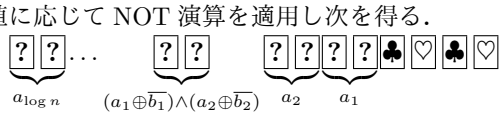
(2) 2.6 節で紹介したコピープロトコルで a_1 をコピーする．



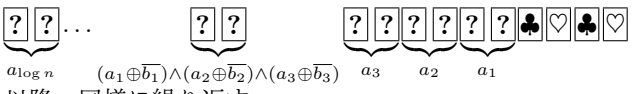
(3) b_1 と b_2 の値に応じて NOT 演算を行い，次の結果を得る．ここで各 b_j の値は公知である．



(4) 次に 1 入力保存の AND プロトコルを用いて，$(a_1 \oplus \overline{b_1}) \wedge (a_2 \oplus \overline{b_2})$ と $(a_2 \oplus \overline{b_2})$ を得る．後者に対して b_2 の値に応じて NOT 演算を適用し次を得る．



(5) 同様にして，$(a_1 \oplus \overline{b_1}) \wedge (a_2 \oplus \overline{b_2}) \wedge (a_3 \oplus \overline{b_3})$ を得る．



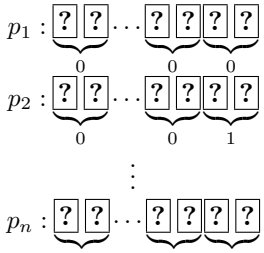
以降，同様に繰り返す．

(6) 最後に 1 番左側のコミットメントを表にする．そのとき，コミットメントが ♡ ♣ の場合は不動点である．♣ ♡ の場合は不動点ではない．どちらの場合でも $a_1, a_2, \cdots, a_{\log n}$ のコミットメントは失われない．

3. 既存プロトコル

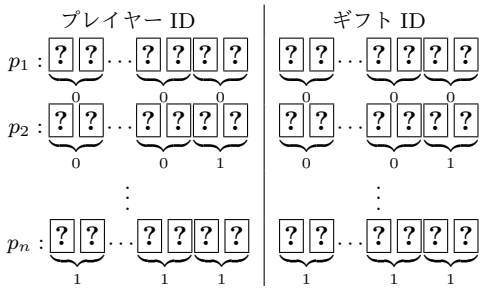
Ibaraki と Manabe [5] はプレイヤー ID とギフト ID，巡回シャッフルを導入し不動点を持たないランダム置換を効率よく生成する方法を提案している．本節では彼らのプロトコルを紹介するとともに生成される置換についても言及する．

(1) 各プレイヤー p_i に対応して $(i-1)_2$ のコミットメントの束を用意する．



これをギフト ID と呼び，プレイヤー p_i のギフト ID を α_i とする．

(2) ギフト ID のコミットメントと同じものを用意し，これをプレイヤー ID と呼び，プレイヤー p_i のプレイヤー ID を β_i とする．



(3) プレイヤー ID とギフト ID をペアにし，それを 1 つの束として Pile-Scramble シャッフルを実行する．この Pile-Scramble シャッフルにより，次のように置換 π で並べ替えが起こったとする．

$$\begin{array}{ll} p_1 & \beta_{\pi(1)} : \alpha_{\pi(1)} \\ p_2 & \beta_{\pi(2)} : \alpha_{\pi(2)} \\ p_3 & \beta_{\pi(3)} : \alpha_{\pi(3)} \\ \vdots & \vdots \\ p_n & \beta_{\pi(n)} : \alpha_{\pi(n)} \end{array}$$

(4) プレイヤー ID とギフト ID のペアを切り離す．ギフト ID のみに対して巡回シャッフルを実行する．同様に，c を長さ n の一様ランダムな置換とすると，次のように書ける．

$$\begin{array}{ll} p_1 & \beta_{\pi(1)} : \alpha_{c(\pi(1))} \\ p_2 & \beta_{\pi(2)} : \alpha_{c(\pi(2))} \\ p_3 & \beta_{\pi(3)} : \alpha_{c(\pi(3))} \\ \vdots & \vdots \\ p_n & \beta_{\pi(n)} : \alpha_{c(\pi(n))} \end{array}$$

(5) 2.8 節で紹介した不動点チェックを $(\beta_{\pi(1)}, \alpha_{c(\pi(1))})$ のペアに対して行う．この時プレイヤー ID $\beta_{\pi(1)}$ をコピーする必要がある．もし不動点であった場合は（4）の手順からやり直す．

(6) 不動点ではないことが確認できたなら，もう一度プレイヤー ID とギフト ID をペアにして Pile-Scramble シャッフルを行う．プレイヤー ID を表側にする．プレイヤー ID に対応するプレイヤーがペアになっているギフト ID のプレゼントを得る．

巡回シャッフルを導入することによって，不動点チェックの回数が 1 回で済む．なぜなら巡回シャッフルの性質によって結果が次の 2 つに大別できるからである．（1）全てのプレイヤーが不動点ではない．（2）全てのプレイヤーが不動点である．つまり 1 人のプレイヤーが不動点でなければ必然的に他のプレイヤーも不動点ではない．そのため不動点チェックを行うのは代表プレイヤー（上述の場合は $p_{\pi(1)}$）だけでよい．

ただし，このプロトコルによって生成される置換は $n!$ 通りの置換から一様にランダムに生成されるわけではない．置換の生成に巡回シャッフルを用いているため，$n!$ 通りの置換のうち生成されない置換が存在する．

具体的には n 人の中で部分的な巡回を含む置換が生成されることはない．例えば 5 人でこのプロトコルを手順（3）まで実行し，次のような結果となったとする．

$$\begin{array}{ll} p_1 & \beta_4 : \alpha_4 \\ p_2 & \beta_2 : \alpha_2 \\ p_3 & \beta_5 : \alpha_5 \\ p_4 & \beta_3 : \alpha_3 \\ p_5 & \beta_1 : \alpha_1 \end{array}$$

上の 3 人のプレイヤー，つまり p_4，p_2，p_5 でプレゼントを巡回させようとする．巡回シャッフルの置換の数は 5 通りであるがうち 1 つは不動点を含むため，不動点を含まないものは以下の 4 通りである．

$$\begin{array}{llllll} p_1 & \beta_4 : \alpha_1 & \beta_4 : \alpha_3 & \beta_4 : \alpha_5 & \beta_4 : \alpha_2 \\ p_2 & \beta_2 : \alpha_4 & \beta_2 : \alpha_1 & \beta_2 : \alpha_3 & \beta_2 : \alpha_5 \\ p_3 & \beta_5 : \alpha_2 & \beta_5 : \alpha_4 & \beta_5 : \alpha_1 & \beta_5 : \alpha_3 \\ p_4 & \beta_3 : \alpha_5 & \beta_3 : \alpha_2 & \beta_3 : \alpha_4 & \beta_3 : \alpha_1 \\ p_5 & \beta_1 : \alpha_3 & \beta_1 : \alpha_5 & \beta_1 : \alpha_2 & \beta_1 : \alpha_4 \end{array}$$

p_4，p_2，p_5 でプレゼントを巡回させるためには β_4，β_2，β_5 の隣に α_3 と α_1 がこないことが条件である．これらの置換の中で α_3 に注目する．置換の中で α_3 が取れる位置は自身のプレイヤー ID β_3 を除いた次の 4 つである．さらに前述した条件を考えると，残る置換は

$$\begin{array}{ll} p_1 & \beta_4 : \alpha_1 \\ p_2 & \beta_2 : \alpha_4 \\ p_3 & \beta_5 : \alpha_2 \\ p_4 & \beta_3 : \alpha_5 \\ p_5 & \beta_1 : \alpha_3 \end{array}$$

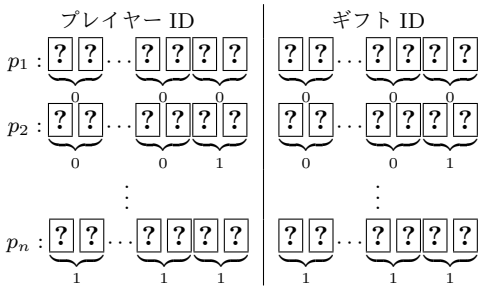
上記の 1 つのみとなる．しかしこの置換も α_1 が β_4 の隣にきているため条件を満たしていない．生成されるどの置換も条件を満たしていない．つまりこの置換生成では特定のプレイヤー間でプレゼントを巡回させる置換は生成されない．

このように，このプロトコルが出力する置換は長さ n のサイクルになっており，そのような置換のうち不動点を持たないものを一様ランダムに生成している．したがって，不動点を持たない任意の置換をランダムに生成しているわけではない．しかし，本稿ではプレゼント交換に用いる置換生成を考えている．大人数で集まった状態でのプレゼント交換であるのでその内の少人数でプレゼント交換が起こることは望ましくない．したがって，既存プロトコルで得られる置換は必ず最長のサイクルになるので，プレゼント交換の実用上は問題がない，あるいはむしろ好ましいといえる．

4. 提案プロトコル

本節では前節で紹介した既存のプロトコルを改良し，より少ないカードの枚数と手順でプレゼント交換に適した置換を生成するプロトコルを提案する．

(1) 既存プロトコルと同様に各プレイヤー p_i に対して，$(i-1)_2$ のプレイヤー ID とギフト ID を生成する．



(2) プレイヤー ID とギフト ID をペアにし，それを 1 つの束として Pile-Scramble シャッフルを実行する．シャッフルにより置換 π によって並べ替えが起こったとする．

$$\begin{array}{ll} p_1 & \beta_{\pi(1)} : \alpha_{\pi(1)} \\ p_2 & \beta_{\pi(2)} : \alpha_{\pi(2)} \\ p_3 & \beta_{\pi(3)} : \alpha_{\pi(3)} \\ \vdots & \vdots \\ p_n & \beta_{\pi(n)} : \alpha_{\pi(n)} \end{array}$$

(3) プレイヤー ID とギフト ID のペアを切り離す．ギフト ID のみに対して 1 つだけずらす巡回シフトを実行する．

$$\begin{array}{ll} p_1 & \beta_{\pi(1)} : \alpha_{\pi(n)} \\ p_2 & \beta_{\pi(2)} : \alpha_{\pi(1)} \\ p_3 & \beta_{\pi(3)} : \alpha_{\pi(2)} \\ \vdots & \vdots \\ p_n & \beta_{\pi(n)} : \alpha_{\pi(n-1)} \end{array}$$

(4) もう一度プレイヤー ID とギフト ID をペアにして Pile-Scramble シャッフルを行う．

(5) プレイヤー ID を表側にする．プレイヤー ID に対応するプレイヤーがギフト ID に対応するプレゼントを得る．

既存のプロトコルとくらべると，巡回シャッフルを簡略化し，1 つずらすだけの巡回シフトを行っている．そのため，不動点が発生することがなく，不動点チェックを省略することができる．

このプロトコルは

$$(\pi_{(1)}\ \pi_{(2)}\ \cdots\ \pi_{(n)})$$

という巡回置換を生成しているとみなすことができる．π が一様ランダムであるため，この得られる置換は不動点を持たない長さ n の一様ランダムなサイクルになっている．すなわち，プロトコルの出力は，そのような $(n-1)!$ 個の置換からなる集合の上に一様分布している．

5. プロトコルの評価

本節では既存プロトコルと提案プロトコルを比較しプロトコルの評価を行う．

5.1 カード枚数

両プロトコルの使用するカード枚数を表にすると以下のようになる．

表 1 両プロトコルの使用カード枚数
Table 1 The number of cards used in each protocol

	使用カード枚数
既存プロトコル	$4n\lceil \log n \rceil + 6$
提案プロトコル	$4n\lceil \log n \rceil$

どちらのプロトコルも置換生成に用いるカードの枚数は $4n\lceil \log n \rceil$ 枚と変わらない．しかし既存プロトコルは不動点チェックを行う必要があり，そのため 2.8 節で述べたようにカードを 6 枚追加している．提案プロトコルは不動点チェックが不要であるため，そのようなカードの追加の必要は発生していない．

5.2 計算コスト

はじめにシャッフル回数について比較を行う．ランダム二等分割カット，巡回シャッフル，Pile-Scramble シャッフルの 3 つのシャッフルの計算コストは同一であるとする．

既存プロトコルでは置換生成に必要なシャッフル回数は 3 回である．しかし既存プロトコルでは不動点チェックを行う必要があり，不動点チェックの際にプレイヤー ID をコピーする必要がある．そのためコピープロトコルを $\lceil \log n \rceil$ 回実行しなければならず不動点チェック 1 回あたり，ランダム二等分割カットの回数が $\lceil \log n \rceil$ 回増加する．また，AND プロトコルを $\lceil \log n \rceil - 1$ 回実行することになるので，同じ回数のシャッフルが必要である．

さらに既存プロトコルで不動点が発生した場合，巡回シャッフルと不動点チェックをやり直すため不動点が 1 回発生する度に，$2\lceil \log n \rceil$ 回増加してしまう．

一方，提案プロトコルでは置換生成が 2 回のシャッフルと 1 回の巡回シフトで可能になり計算コストを削減している．また不動点チェックが不要であるため，シャッフル回数の増加が発生せずシャッフル以外の計算コストの削減にも成功している．

さらに，既存プロトコルは最悪の場合，置換生成と不動点チェックを無限に繰り返してしまいプロトコルが終了しない可能性がある．一方，提案プロトコルではその問題を解決し，有限のシャッフル回数と実行時間でプロトコルが終了する．

5.3 安全性

既存プロトコルではプレゼントの贈り主を示すギフト ID と贈り先であるプレイヤー ID を別々に用意している．そのうち公開するのはプレイヤー ID のみであるので，贈り先のプレイヤー以外にはプレゼントの贈り主はわからな

い．このようにして安全性を得ている．

提案プロトコルにおいても，既存プロトコルと同様にプレイヤー ID とギフト ID を用意し，そのうちギフト ID のみ公開する．そのため提案プロトコルは既存プロトコルのカード枚数と計算コストを削減しつつ，安全性は維持している．

6. むすび

本稿では 2016 年に Ibaraki と Manabe が提案したプロトコルを紹介し，生成される置換が一様ランダムではないことを指摘した．しかしプレゼント交換の実用上は問題がなく，このプロトコルを改良し，安全性を保ちながらシンプルに同様の置換を得られるプロトコルを示した．

参考文献

[1] Claude Crépeau and Joe Kilian, “Discreet solitary games,” Proc. CRYPTO ’93, Lecture Notes in Computer Science, vol. 773, pp. 319–330, Springer-Verlag, 1994.

[2] Rie Ishikawa, Eikoh Chida, and Takaaki Mizuki, “Efficient Card-Based Protocols for Generating a Hidden Random Permutation without Fixed Points,” Unconventional Compulation and Natural Computation (UCNC 2015), Lecture Notes in Computar Science, Springer-Verlag, vol.9252, pp.215-226, 2015.

[3] Takaaki Mizuki and Hideaki Sone, “Six-card secure AND and four-card secure XOR,” Proc. Frontiers in Alogorithmics(FAW 2009), Lecture Notes in Computer Science, vol.5598, pp358-369, Springer-Verlag, 2009.

[4] Takaaki Mizuki, Isaac Kobina Asiedu, and Hideaki sone, “Voting with a logarithmic number of cards,” Proc. Unconventional Computation and Natural Computation (UCNC 2013), Lecture Notes in Computer Science, Springer-Verlag, vol. 7956, pp.162-173, 2013.

[5] Takuya Ibaraki and Yoshifumi Manabe, “A More Efficient Card-Based Protocol for Generating a Random Permutation Without Fixed Points,” Third International Conference on Mathematics and Computers in Sciences and in Industry, pp.252-256, 2016.