Card-based Cryptographic Protocols Using a Minimal Number of Cards

Alexander Koch, Stefan Walzer, and Kevin Härtel

Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
alexander.koch@kit.edu, {stefan.walzer, kevin.haertel}@student.kit.edu

Abstract. Secure multiparty computation can be done with a deck of playing cards. For example, den Boer (EUROCRYPT '89) devised his famous "five-card trick", which is a secure two-party AND protocol using five cards. However, the output of the protocol is revealed in the process and it is therefore not suitable for general circuits with hidden intermediate results. To overcome this limitation, protocols in committed format, i.e., with concealed output, have been introduced, among them the six-card AND protocol of (Mizuki and Sone, FAW 2009). In their paper, the authors ask whether six cards are minimal for committed format AND protocols. We give a comprehensive answer to this problem: there is a four-card AND protocol with a runtime that is finite in expectation (i.e., a Las Vegas protocol), but no protocol with finite runtime. Moreover, we show that five cards are sufficient for finite runtime. In other words, improving on (Mizuki, Kumamoto and Sone, ASIACRYPT 2012) "The Five-Card Trick can be done with four cards", our results can be stated as "The Five-Card Trick can be done in committed format" and furthermore it "can be done with four cards in Las Vegas committed format".

By devising a Las Vegas protocol for any k-ary boolean function using $2k$ cards, we address the open question posed by (Nishida et al., TAMC 2015) on whether $2k+6$ cards are necessary for computing any k-ary boolean function. For this we use the shuffle abstraction as introduced in the computational model of card-based protocols in (Mizuki and Shizuya, Int. J. Inf. Secur., 2014). We augment this result by a discussion on implementing such general shuffle operations.

Keywords: card-based protocols · committed format · boolean AND · secure computation · cryptography without computers

1 Introduction

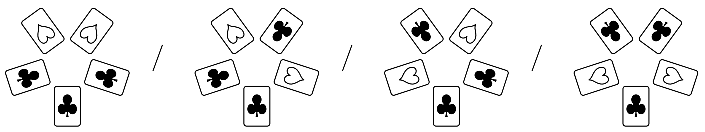
The most well known card-based cryptographic protocol uses five cards showing two different types of symbols, ♡ and ♣, which are otherwise assumed to be physically indistinguishable. Let us quickly describe the elegant "five-card trick" of den Boer [B89] for computing a logical AND operation on the bits of two players. For this, the players input their bits as a commitment, which is two face-down cards either as ♡♣ or ♣♡, encoding 1 or 0, respectively, with a separating ♣ card in between, so that the possible input sequences look like this:

$$\underbrace{\heartsuit\clubsuit}_{a=1}\clubsuit\underbrace{\heartsuit\clubsuit}_{b=1}\ /\ \underbrace{\heartsuit\clubsuit}_{a=1}\clubsuit\underbrace{\clubsuit\heartsuit}_{b=0}\ /\ \underbrace{\clubsuit\heartsuit}_{a=0}\clubsuit\underbrace{\heartsuit\clubsuit}_{b=1}\ /\ \underbrace{\clubsuit\heartsuit}_{a=0}\clubsuit\underbrace{\clubsuit\heartsuit}_{b=0}$$

Now, the second player inverts his bit by swapping his cards, leading to the following situation:

$$\underbrace{\heartsuit\clubsuit\clubsuit\clubsuit\heartsuit}_{a\wedge b=1}\ /\ \underbrace{\heartsuit\clubsuit\clubsuit\heartsuit\clubsuit}_{a\wedge b=0}\ /\ \underbrace{\clubsuit\heartsuit\clubsuit\clubsuit\heartsuit}_{a\wedge b=0}\ /\ \underbrace{\clubsuit\heartsuit\clubsuit\heartsuit\clubsuit}_{a\wedge b=0}$$

Observe that only in the case of $a = b = 1$, the three ♣s are consecutive. The following cyclic arrangement of cards as seen from below a "glass table" makes it obvious that this property is preserved under cyclic shifts of the cards:



By applying a cyclic shift by a random offset, the correspondence of the positions to the players is obscured. This "shuffling" of the cards can be done by the players taking turns in applying a cyclic shift of a random offset, without letting the other players observe the permutation that has been applied to the cards. By revealing all cards afterwards, the players can check whether the three ♣s are consecutive and deduce that the output is 1 if this is the case, and 0 otherwise.

This example illustrates that a deck of cards can be used to securely evaluate functions, without the players giving away anything about their inputs that cannot be deduced from the result of the execution of such a card-based cryptographic protocol. The utility of these protocols is evident from their use in classrooms and lectures to illustrate secure multiparty computation to non-experts to the field of cryptography, or in an introductory course. Moreover, the possibility of performing these protocols without the use of computers is an interesting distinctive feature.

In their ASIACRYPT 2012 paper, Mizuki, Kumamoto, and Sone [MKS12] were able to reduce the number of cards to the best possible of 4, which is already necessary to encode the inputs. However, both protocols have an important caveat: They unavoidably reveal the final result during the computation. This makes them inadequate for use in larger protocols, for instance when evaluating complex logical circuits.

Therefore, starting with [NR98; S01; CK93], several researchers came up with so-called committed format protocols, which output a commitment encoding the result by two cards, as described above. This allows for using the output commitment of the protocol as an input to another protocol and for having a fine-grained control on who learns what about the result.

So far, the protocols using the least number of cards for computing AND in committed format are

- the six-card protocol of Mizuki and Sone [MS09], which has a deterministic runtime (cf. Fig. 2), and
- the five-card Las Vegas protocol of [CHL13], as described in Example 1. Note that this protocol may end in a configuration which needs restarting with probability $1/2$ and utilizes a rather complex shuffle operation. (These operations will be discussed in Section 8).

This leads to the natural question on the minimality of cards needed for a secure committed format AND, which has been posed in several places in the literature, see, e.g., [MS09; MS14a; MKS12]. Moreover in [CHL13], the authors ask whether there is a "deterministic" five-card variant of their protocol. In this paper, we answer these questions comprehensively.

To cope with these questions, [MS14a] defined a formal computational model stating the possible operations that a card-based protocol can make. To allow for strong impossibility results, the authors give a rather wide palette of possible operations that can be applied to the cards, e.g., shuffling with an arbitrary probability distribution on the set of permutations. Our paper shows that this yields rather strong possibility results by utilizing "non-closed" shuffles, as defined in Section 8.

Note that all protocols are in the *honest-but-curious* setting (although some analysis of malicious behavior has been done in [MS14b]), i.e., the players execute the protocol according to its description, but gather any information they can possibly obtain.

Contribution. In this paper, we

- introduce a four-card Las Vegas protocol for the AND of two players' bits,
- give a five-card variant, which has an a priori bound on the number of execution steps, i.e., a finite-runtime protocol,
- show that this is optimal, as four-card finite-runtime protocols computing AND in committed format are impossible,
- define a method of enriching the description of a protocol, that makes correctness and security transparent and gives a good understanding of how these protocols work, which can be used as a leverage to devise impossibility results. We therefore believe that this method is of general interest for research in card-based cryptography,
- state a general $2k$-card protocol for any k-ary boolean function, which can be seen as a touchstone for the practicability of the underlying computational model,
- discuss the computational model of [MS14a] briefly.

For comparison with other protocols, we refer the reader to Tables 1 and 2. For the former, we have three key parameters in describing the properties of protocols: whether it is committed format, whether it is a finite-runtime or a Las Vegas algorithm, and whether "non-closed" or "non-uniform" shuffles are used in the protocols, for which it is not yet apparent how they can be run in practice,

Table 1. Minimal number of cards required by protocols computing AND of two bits, subject to the requirements specified in the first three columns.

format	runtime	shuffles	#cards	reference
committed	exp. finite	non-uniform closed	4	Theorem 1
non-committed	finite	uniform closed	4	[MKS12]
committed	finite	non-uniform non-closed	5	Theorems 2 and 3
committed	exp. finite	uniform non-closed	≤ 5	[CHL13]
committed	finite	uniform closed	≤ 6	[MS09]

Table 2. Comparison of protocols for k-ary boolean functions.

#cards	success probability	shuffles	#steps	reference
$2k$	2^{-k}	uniform non-closed	constant	Theorem 4
$2k+6$	1	uniform closed	large	[N+15]

cf. Section 8 for a discussion. Table 1 states the minimal number of cards for protocols with the given parameters and gives the corresponding references.

In Table 2 we compare our $2k$-card protocol of Section 7 with the best protocol for general boolean functions in the literature, with respect to the number of cards, namely [N+15]. While our protocol reduces the number of cards by six, it is a Las Vegas protocol with a substantial probability to end in a state which requires to restart the protocol. Moreover, it uses the non-closed shuffles mentioned above. Even though the expected number of restarts until a successful run is of order $O(2^k)$, each run of our protocol requires only a constant number of steps. This result can also be interpreted as a touchstone of the plausibility of the computational model for card-based protocols.

Outline. In Section 2 we introduce the basic computational model of card-based protocols and a strong information-theoretic security definition. We describe a method for the analysis of protocols in Section 3. We give a description of our four- and five-card protocols in Section 4 and Section 5, respectively. In the subsequent Section 6 we show that five cards are necessary for finite-runtime protocols. In Section 7 we state a Las Vegas protocol for general boolean functions using a strong shuffle operation that the computational model allows. We discuss these shuffle operations in Section 8. Finally, we conclude the paper in Section 9.

Notation. In the paper we use the following notation.

– *Cycle Decomposition.* For $n \in \mathbb{N}$ and numbers $a_1, a_2, \ldots, a_k \leq n$ we write $\pi = (a_1\ a_2\ \ldots\ a_k)$ for the permutation $\pi \in S_n$ that maps a_i to a_{i+1} for $1 \leq i \leq k-1$ and a_k to a_1 and all other $x \leq n$ to themselves. We call this a *cycle*. Cycles are maps, so they can be composed with $\circ$, which we will omit in the following, e.g. $(1\ 3\ 5)(2\ 4)$ maps $1 \mapsto 3$, $3 \mapsto 5$, $5 \mapsto 1$, $2 \mapsto 4$ and $4 \mapsto 2$.

– *Drawing from a Probability Distribution.* If $\mathcal{F}$ is a probability distribution on a set X, we write $x \leftarrow \mathcal{F}$ to indicate that $x \in X$ should be randomly chosen from X according to $\mathcal{F}$.
– *Sequence Indices.* Given a sequence $x = (\alpha_1, \dots, \alpha_l)$ and an index i with $1 \leq i \leq l$, we denote by $x[i]$, the ith entry of the sequence, namely α_i.

2 Machine Model and Security of Card-based Protocols

Mizuki and Shizuya [MS14a] came up with an elegant framework to model a computation with card-based cryptographic protocols. We adopt their setting to our needs and quickly review the important definitions in the following.

A *deck* $\mathcal{D}$ is a finite multiset of symbols, its elements are *cards*. We will restrict ourselves to the case where $\mathcal{D}$ contains two types of symbols, depicted by $\heartsuit$ and $\clubsuit$. For a symbol $c \in \mathcal{D}$, $\frac{c}{?}$ denotes a *face-up card* and $\frac{?}{c}$ a *face-down card* with symbol c, respectively. Here, '?' is a special backside symbol, not contained in $\mathcal{D}$. For a face-up or face-down card α, $\mathsf{top}(\alpha)$ and $\mathsf{atom}(\alpha)$ denote the symbol in the "numerator" and the symbol distinct from '?', respectively.

Cards are lying on the table in a *sequence.* A sequence is obtained by permuting $\mathcal{D}$ and choosing face-up or face-down for each card. For example, $(\frac{?}{\clubsuit}, \frac{\clubsuit}{?}, \frac{?}{\heartsuit}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit})$ is a sequence of $\mathcal{D} = [\clubsuit, \clubsuit, \clubsuit, \heartsuit, \heartsuit]$. We extend $\mathsf{top}(\cdot)$ and $\mathsf{atom}(\cdot)$ from single cards to sequences of cards in the canonical way. For a sequence Γ, $\mathsf{top}(\Gamma)$ is the *visible sequence* of Γ. For example, $\mathsf{top}\left(\frac{?}{\clubsuit}, \frac{\clubsuit}{?}, \frac{?}{\heartsuit}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit}\right) = (?, \clubsuit, ?, ?, ?)$. We denote the set of all visible sequences of $\mathcal{D}$ by $\mathsf{Vis}^{\mathcal{D}}$, or Vis for short. Furthermore, we define the set of *atomic sequences* $\mathsf{AtSeq}^{\mathcal{D}}$, or AtSeq for short, as the set of all permutations of $\mathcal{D}$.

A *protocol* $\mathcal{P}$ is a quadruple $(\mathcal{D}, U, Q, A)$, where $\mathcal{D}$ is a deck, U is a set of input sequences, Q is a set of states with two distinguished states q_0 and q_{f}, being the initial and the final state. Moreover, we have a (partial) action function

$$A\colon (Q \setminus \{q_{\mathrm{f}}\}) \times \mathsf{Vis} \to Q \times \mathsf{Action},$$

depending only on the current state and visible sequence, specifying the next state and an operation on the sequence from Action that contains the following actions:

– (perm, π) for a permutation $\pi \in S_{|\mathcal{D}|}$ from the symmetric group $S_{|\mathcal{D}|}$ on elements $\{1, \dots, |\mathcal{D}|\}$. This transforms a sequence $\Gamma = (\alpha_1, \dots, \alpha_{|\mathcal{D}|})$ into

$$\mathsf{perm}_\pi(\Gamma) \coloneqq (\alpha_{\pi^{-1}(1)}, \dots, \alpha_{\pi^{-1}(|\mathcal{D}|)}),$$

i.e., it permutes the cards according to π.
– (turn, T) for $T \subseteq \{1, \dots, |\mathcal{D}|\}$. This transforms a sequence $\Gamma = (\alpha_1, \dots, \alpha_{|\mathcal{D}|})$ into

$$\mathsf{turn}_T(\Gamma) \coloneqq (\beta_1, \dots, \beta_{|\mathcal{D}|}), \text{ where } \beta_i = \begin{cases} \mathsf{swap}(\alpha_i), & \text{if } i \in T, \\ \alpha_i, & \text{otherwise,} \end{cases}$$

i.e., it turns over all cards from a *turn set* T. Here $\mathsf{swap}(\frac{c}{?}) \coloneqq \frac{?}{c}$ and $\mathsf{swap}(\frac{?}{c}) \coloneqq \frac{c}{?}$, for $c \in \mathcal{D}$.

– $(\mathsf{shuffle}, \Pi, \mathcal{F})$ for a probability distribution $\mathcal{F}$ on $S_{|\mathcal{D}|}$ with support Π. This transforms a sequence Γ into the random sequence

$$\mathsf{shuffle}_{\Pi,\mathcal{F}}(\Gamma) \coloneqq \mathsf{perm}_{\pi}(\Gamma), \text{ for } \pi \leftarrow \mathcal{F},$$

i.e., $\pi \in \Pi$ is drawn according to $\mathcal{F}$ and then applied to Γ. Note that the players do not learn the chosen permutation when executing the protocol (unless they can derive it from $\mathcal{F}$ and the visible sequence after the operation). If $\mathcal{F}$ is the uniform distribution on Π, we may omit it and write $(\mathsf{shuffle}, \Pi)$.

– $(\mathsf{rflip}, \Phi, \mathcal{G})$ for a probability distribution $\mathcal{G}$ on $2^{\{1,\ldots,|\mathcal{D}|\}}$ with support Φ. This transforms a sequence Γ into

$$\mathsf{rflip}_{\Phi,\mathcal{G}}(\Gamma) \coloneqq \mathsf{turn}_{T}(\Gamma), \text{ for } T \leftarrow \mathcal{G},$$

i.e., $T \subseteq \{1, \ldots, |\mathcal{D}|\}$ is drawn according to $\mathcal{G}$ and then the corresponding cards of Γ are turned.

– $(\mathsf{restart})$. This transforms a sequence into the start sequence. This special operation requires that the first component of A's output, i.e., the next state, is q_0. This allows for Las Vegas protocols that "fail" and start over with a certain probability. Protocols with a (deterministic) finite runtime do not need this operation.

– $(\mathsf{result}, p_1, \ldots, p_l)$ for a list of positions $p_1, \ldots, p_l \in \{1, \ldots, |\mathcal{D}|\}$. This special operation occurs if and only if the first component of A's output is q_{f}. This halts the protocol and specifies that $(\alpha_{p_1}, \ldots, \alpha_{p_l})$ is the *output*, where $\Gamma = (\alpha_1, \ldots, \alpha_{|\mathcal{D}|})$ is the current sequence.

A tuple $(\Gamma_0, \Gamma_1, \ldots, \Gamma_t)$ of sequences such that $\Gamma_0 \in U$ and Γ_{i+1} arises from Γ_i by an operation as specified by the action function in a protocol run is a *sequence trace*; in that case $(\mathsf{top}(\Gamma_0), \mathsf{top}(\Gamma_1), \ldots, \mathsf{top}(\Gamma_t))$ is a *visible sequence trace.*[1]

A protocol *terminates* when entering the final state q_{f}. A protocol is called *finite-runtime*[2] if there is a fixed bound on the number of steps, and in contrast *Las Vegas*, if it terminates almost surely (i.e., with probability 1) and in a number of steps that is *only expectedly* finite.

Next we describe a canonical form for protocols computing boolean functions. For this we interpret two cards with distinct symbols as 1, if their symbols are arranged $\heartsuit\clubsuit$, and 0, if they are arranged as $\clubsuit\heartsuit$.

Definition 1. *Let* $f\colon \{0,1\}^k \to \{0,1\}$ *be a boolean function. Then we say a protocol* $\mathcal{P} = (\mathcal{D}, U, Q, A)$ computes f*, if the following holds:*

– *the deck* $\mathcal{D}$ *contains at least* k *cards of each symbol,*

[1] Note that traces in our sense also capture prefixes of complete protocol runs.

[2] We avoid the term "deterministic" here, as, for their security, card-based protocols use randomness as an intrinsic property, albeit not necessarily as a speedup of the protocol.

– *there is a one-to-one correspondence between inputs and input sequences, with the convention that for* $b \in \{0,1\}^k$ *we have that* U *contains* $\Gamma^b = (\alpha_1, \ldots, \alpha_{|\mathcal{D}|})$*, where*

$$(\alpha_{2i-1}, \alpha_{2i}) = \begin{cases} (\frac{?}{\heartsuit}, \frac{?}{\clubsuit}), & \textit{if } b[i] = 1, \\ (\frac{?}{\clubsuit}, \frac{?}{\heartsuit}), & \textit{if } b[i] = 0, \end{cases}$$

for $1 \leq i \leq k$*. The remaining* $|\mathcal{D}| - 2k$ *"helping" cards are arranged in some canonical way (their arrangement does not depend on* b*). In this paper we assume that the helping* $\clubsuit$*s are to the left of the helping* $\heartsuit$*s.*

– *it terminates almost surely,*

– *for an execution starting with* Γ^b *for* $b \in \{0,1\}^k$ *the protocol ends with the action* $(\mathsf{result}, p_1, p_2)$*, such that*

$$\mathsf{atom}\,(\beta_{p_1}, \beta_{p_2}) = \begin{cases} (\heartsuit, \clubsuit), & \textit{if } f(b) = 1, \\ (\clubsuit, \heartsuit), & \textit{otherwise,} \end{cases}$$

where $\Gamma = (\beta_1, \ldots, \beta_{|\mathcal{D}|})$ *is the final sequence.*

Example 1. Let us describe, as an example, the Las Vegas five-card AND protocol of Cheung, Hawthorne, and Lee [CHL13]. Here, the deck is $\mathcal{D} = [\heartsuit, \heartsuit, \clubsuit, \clubsuit, \clubsuit]$ and the set of inputs is given by $U = \{\Gamma^{11}, \Gamma^{10}, \Gamma^{01}, \Gamma^{00}\}$, where $\Gamma^{11} = (\frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{?}{\clubsuit})$, $\Gamma^{10} = (\frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit})$, $\Gamma^{01} = (\frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{?}{\clubsuit})$, and $\Gamma^{00} = (\frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit}, \frac{?}{\heartsuit}, \frac{?}{\clubsuit})$. The protocol $\mathcal{P} = (\mathcal{D}, U, \{q_0, q_1, q_2, q_3, q_{\mathrm{f}}\}, A)$ is then described by A as follows:

1. $A(q_0, v) = (q_1, (\mathsf{perm}, (2\ 3\ 4\ 5)))$, i.e., insert the helping card at position 2.[3]
2. $A(q_1, v) = (q_2, (\mathsf{shuffle}, \Pi))$, where $\Pi = \{\mathsf{id}, (1\ 4\ 2\ 5\ 3)\}$.
3. $A(q_2, v) = (q_3, (\mathsf{turn}, \{1\}))$, i.e., turn the first card.
4. $A(q_3, v) = \begin{cases} (q_{\mathrm{f}}, (\mathsf{result}, 2, 3)), & \text{if } v[1] = \clubsuit, \\ (q_0, (\mathsf{restart})), & \text{otherwise.} \end{cases}$

Here, v denotes the current visible sequence in each step. Note that there is no obvious way to implement the shuffle in step 2 efficiently, as it is non-closed. See Section 8 for discussion.

Definition 2 (secure, committed format). *Let* $\mathcal{P} = (\mathcal{D}, U, Q, A)$ *be a protocol. Let* Γ_0 *be a random variable with values in the set of input sequences* U *and a distribution* $\mathcal{M}$ *on* U*. Let* V *be a random variable for the visible sequence trace of the protocol execution.*

$\mathcal{P}$ *is* secure *or* private *if* Γ_0 *and* V *are stochastically independent.*

Moreover, let R *be a random variable that encodes the output of the protocol. Then* $\mathcal{P}$ *is said to be in* committed format*, if* $\mathsf{atom}(R)$ *and* V *are stochastically independent. (In particular, this implies that an index occurring in the* result *action points to a face-down card, unless this part of the output is constant.)*

[3] Note that this step is only needed because our input convention from Definition 1 differs from the input convention of [CHL13].

From this definition it is apparent that if there is a functional dependency between the inputs and the output, then security implies committed format. Note that it is stronger than other security definitions in the literature that were defined to also capture non-committed format protocols, such as the five-card trick of [B89].

When the input is provided by players, each of them have a partial knowledge on Γ_0. The definition then implies that, even given this partial knowledge, Γ_0 and V are still independent. Therefore the players cannot learn anything about the inputs of the other players, as the result is not part of V.

3 A Calculus of States

From a specification of a protocol it is not immediately obvious whether it is correct and private. We describe a new method to obtain a rich description of possible protocol runs, from which correctness and privacy can be more easily recognized. We use this method in later sections to describe our constructions and prove the impossibility of finite-runtime four-card AND in Section 6. We believe this method is of general interest for researchers in the field of card-based cryptography.

When describing all possible executions of a protocol we obtain a tree which branches when the visible sequence differs. The nodes of this tree correspond to the visible sequence traces that can occur during the run of the protocol. Each node has an action associated to it, namely the action that the protocol prescribes for that situation. In the following, this action is a label on the outgoing edges.

Take for instance the six-card AND protocol of [MS09], as shown in Fig. 1. We hope that it will soon become clear why the protocol works.

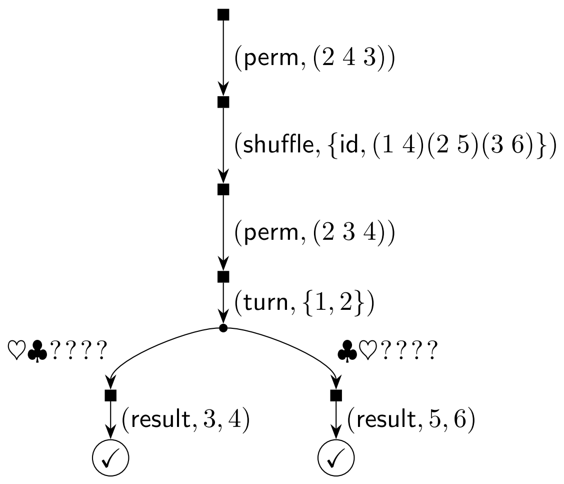


Fig. 1. Six-card AND protocol in committed format of [MS09].

Until the fourth step (the turn step) there is no observable difference, i.e., all visible sequences contain only '?'. After the turn, there are two types of executions that can be distinguished by players. If security was violated, i.e., players can deduce information about the input, then this is because some inputs are more likely to lead to a specific visible sequence than other inputs.

While the actual sequence on the table and the actual input of the players is typically unknown, knowledge about the former implies knowledge about the latter and vice versa. To facilitate the privacy analysis, we annotate the nodes of the tree with this dependent knowledge. A state in our sense captures the probability distribution of atomic sequences conditioned on the input sequence.

Definition 3. *Let $\mathcal{P}$ be a secure protocol computing $f\colon \{0,1\}^k \to \{0,1\}$ and V be a visible sequence trace of $\mathcal{P}$. The* state S of $\mathcal{P}$ belonging to V *is the map $S\colon \mathsf{AtSeq} \to \mathbb{X}_k$, with $s \mapsto \Pr[s\,|\,V]$, where:*

- *$\mathbb{X}_k$ denotes the polynomials over the variables X_b for $b \in \{0,1\}^k$ of the form $\sum_{b\in\{0,1\}^k} \beta_b X_b$, for $\beta_b \in [0,1] \subseteq \mathbb{R}$. We interpret these polynomials as probabilities which depend on the probabilities of the inputs b, symbolized by the variables X_b for $b \in \{0,1\}^k$.*
- *for $s \in \mathsf{AtSeq}$, $\Pr[s\,|\,V]$ denotes the (symbolic) probability that the current atomic sequence is s given that current visible sequence trace is V. (It will later be apparent that the probability $\Pr[s\,|\,V]$ is indeed in $\mathbb{X}_k$.)*

We say a state S contains *an atomic sequence s (or s* is in *S for short) if $S(s)$ is not the zero polynomial. For $k \geq 2$, we introduce the additional shorthands $X_0 := \sum_{f(b)=0} X_b$ and $X_1 := \sum_{f(b)=1} X_b$.*

Let S be a state. Given a probability distribution $\mathcal{M}$ on the inputs, then substituting each variable X_b with the probability of the input b, yields a probability distribution on the atomic sequences in S. In particular, if s is an atomic sequence in S and $S(s)$ the corresponding polynomial, substituting 1 for the variable X_b and 0 for the other variables in $S(s)$, yields the probability that s is the current atomic sequence, given the input b and any information observed so far. Accordingly, we can use our notions to analyze player knowledge in multiparty computations where an agent has partial information about the input.

As an illustration of our method, consider the states of the six-card AND protocol from above, see Fig. 2 on page 10, where states are represented by a box with atomic sequences on the left and the associated polynomials on the right. In such a 2-ary protocol, a state maps each atomic sequence to a polynomial of the form $\beta_{11}X_{11} + \beta_{10}X_{10} + \beta_{01}X_{01} + \beta_{00}X_{00}$, where $\beta_{11}, \beta_{10}, \beta_{01}, \beta_{00} \in [0,1]$.

- In the start state, each input $b \in \{00, 01, 10, 11\}$ is associated with a unique input sequence $\Gamma^b \in U$, which, by our conventions in Definition 1, are $\Gamma^{11} = (\heartsuit, \clubsuit, \heartsuit, \clubsuit, \clubsuit, \heartsuit)$, $\Gamma^{10} = (\heartsuit, \clubsuit, \clubsuit, \heartsuit, \clubsuit, \heartsuit)$, $\Gamma^{01} = (\clubsuit, \heartsuit, \heartsuit, \clubsuit, \clubsuit, \heartsuit)$ and $\Gamma^{00} = (\clubsuit, \heartsuit, \clubsuit, \heartsuit, \clubsuit, \heartsuit)$. The probability of $\mathsf{atom}(\Gamma^b)$ being the current atomic sequence is therefore exactly X_b, i.e., the probability that b is the input. The remaining $\binom{6}{3} - 4$ atomic sequences are mapped to zero and omitted in the presentation.

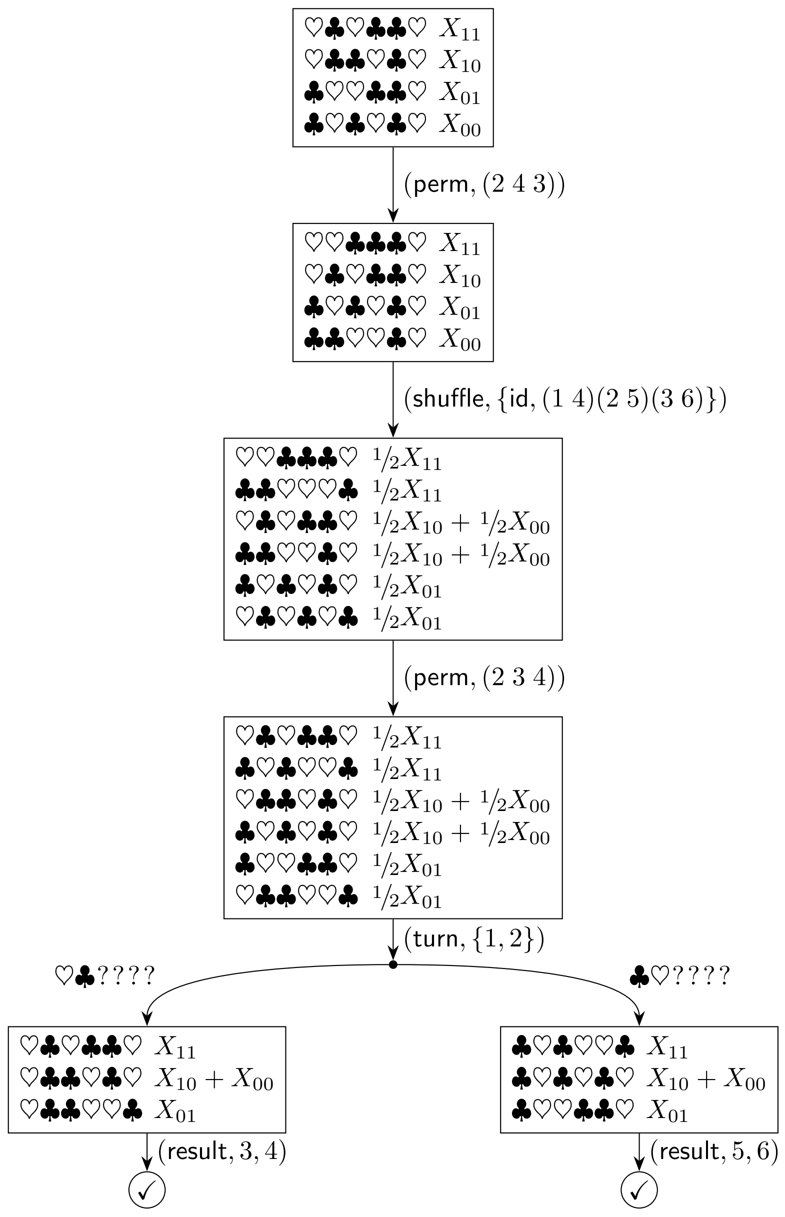


Fig. 2. Six-card AND protocol in committed format of [MS09] augmented with state information as in Definition 3.

- The first (and third) action is a permutation. Mathematically, nothing interesting happens here: If an atomic sequence s had its probability captured by $S(s)$, then after permuting with a permutation π, these probabilities are then assigned to the atomic sequence $\pi(s)$.
- The shuffle introduces uncertainty. Consider for instance the case that the input was "10". Then, before the shuffle, we must have had the atomic sequence $s = (\heartsuit, \clubsuit, \heartsuit, \clubsuit, \clubsuit, \heartsuit)$. It was either permuted by id or by $\pi = (1\ 4)(2\ 5)(3\ 6)$, yielding either s itself or $s' = (\clubsuit, \clubsuit, \heartsuit, \heartsuit, \clubsuit, \heartsuit)$, both with probability $1/2$. This explains the coefficients of X_{10} in the polynomials for s and s'.
- The turn step can yield two possible visible sequences: $(\heartsuit, \clubsuit, ?, ?, ?, ?)$ and $(\clubsuit, \heartsuit, ?, ?, ?, ?)$. Crucially, the probability of observing $(\clubsuit, \heartsuit, ?, ?, ?, ?)$ is the same for each possible input, so no information about the actual sequence is leaked: If $(\clubsuit, \heartsuit, ?, ?, ?, ?)$ would be observed slightly more frequently for, say, the input "01" than for the input "10", then observing $(\clubsuit, \heartsuit, ?, ?, ?, ?)$ would be weak evidence that the input was "01". In the case at hand, however, the probability for the right branch is $1/2$ for each input, as the sum of the polynomials of the atomic sequences branching right is $1/2(X_{11} + X_{10} + X_{01} + X_{11})$.

  After the turn our knowledge has changed, for instance, if we have observed $(\heartsuit, \clubsuit, ?, ?, ?, ?)$ and know that the input was "11" then we know beyond doubt that the atomic sequence must then be $(\heartsuit, \clubsuit, \heartsuit, \clubsuit, \clubsuit, \heartsuit)$, explaining the coefficient 1 of X_{11}.
- The output given by the result actions is correct: For all polynomials containing X_{11} with non-zero coefficient, the corresponding atomic sequence has $(\heartsuit, \clubsuit)$ at the specified positions and for all polynomials containing one of the other variables with non-zero coefficient, the corresponding atomic sequence has $(\clubsuit, \heartsuit)$ there.

  Note that "mixed" polynomials with non-zero coefficients of both types cannot occur in a final state of a protocol.

Derivation Rules for States. To compute the states we first identify the start state and then specify how subsequent states arise from a given state when performing an action. The rules of our calculus can also be seen as an inductive proof that our definition of a state is sound in secure protocols, as the probabilities are in $\mathbb{X}_k$ as claimed.

The *start state* S_0 with initial visible sequence trace V_0 contains exactly the input sequences in U. Each $\Gamma_b \in U$ of input $b \in \{0,1\}^k$ is mapped to the probability $\Pr[\mathsf{atom}(\Gamma_b) \,|\, V_0] = X_b$.

An action $\mathsf{act} \in \mathsf{Action}$ on a state S belonging to a visible sequence trace V can result in visible sequences $v_1, \dots, v_n$. In the following, we state the rules for the derivation of these subsequent states $S_1, \dots, S_n$ belonging to the extended visible sequences traces $V \,\|\, v_i$, obtained by appending the new visible sequence v_i to the trace, for $1 \leq i \leq n$. We restrict the presentation to shuffle and randomized flip operations, as the permutation and turn operations are special cases. For an illustration, we refer to Fig. 3.

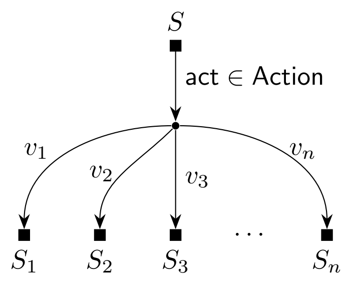


Fig. 3. Performing an action on a state can result in different visible sequences corresponding to a state each.

Shuffle Action. Let $\mathsf{act} = (\mathsf{shuffle}, \Pi, \mathcal{F})$. If all cards are face-down before the shuffle, act can result in only one visible sequence, but in general let Π_v be the subset of Π that leads to some visible sequence v with corresponding state S'. If $\mathcal{F}_{|v}$ denotes the probability distribution on Π_v conditioned on the fact that v is observed, we have that

$$S'(s) = \sum_{\pi \in \Pi_v} \mathcal{F}_{|v}(\pi) \cdot S(\pi^{-1}(s)).$$

In other words, the probability for the atomic sequence s in the new state S' is obtained by considering all atomic sequences $\pi^{-1}(s)$ from which s may have originated through some $\pi \in \Pi_v$ and summing the probability of those atomic sequences in the old state, weighted with the probabilities that the corresponding π is chosen.

Randomized Flip Action. Let $\mathsf{act} = (\mathsf{rflip}, \Phi, \mathcal{G})$. Consider the state S' belonging to the visible sequence trace $V' := V \parallel v$ for the new visible sequence v, resulting from a flip of some turn set $T \in \Phi$. We say that v is *compatible* with an atomic sequence s from S if v and s agree in all positions that are not '?' in v. The set of all atomic sequences compatible with v is denoted by C_v.

Let $P_v := \sum_{s \in C_v} S(s)$. This polynomial represents the probability of observing v if T is turned in state S. Let β_b be the coefficients of P_v, i.e., $P_v = \sum_{b \in \{0,1\}^k} \beta_b X_b$. If the coefficients differ, i.e., $\beta_{b_1} \neq \beta_{b_2}$ for two inputs b_1 and b_2, then the probability of observing v when turning T in state S depends on the input. This must not be the case in secure protocols where visible sequences and inputs are independent.

In secure protocols, we therefore know that

$$P_v = \sum_{b \in \{0,1\}^k} \beta_v X_b = \beta_v \sum_{b \in \{0,1\}^k} X_b,$$

for some $\beta_v \in \mathbb{R}$. In our interpretation as probabilities, we have $\sum_{b \in \{0,1\}^k} X_b = 1$, i.e., the sum over all input probabilities is 1. From this, we obtain $P_v = \beta_v$.

Then, using Bayes' formula yields

$$\begin{aligned} S'(s) &= \Pr[s \,|\, V'] = \Pr[s \,|\, (V \parallel v)] = \Pr[v \,|\, V, s] \cdot \frac{\Pr[s \,|\, V]}{\Pr[v \,|\, V]} \\ &= \Pr[v \,|\, V, s] \cdot \frac{S(s)}{P_v} = \begin{cases} S(s)/\beta_v, & \text{if } s \in C_v, \\ 0, & \text{otherwise,} \end{cases} \end{aligned}$$

where $\Pr[v \,|\, V, s]$ denotes the probability that v occurs, given that the visible sequence trace is V and the actual atomic sequence is s, and $\Pr[v \,|\, V]$ denotes the probability that v occurs, given that the visible sequence trace is V. Note that the actual atomic sequence s determines the visible sequence of the turn action, so $\Pr[v \,|\, V, s]$ is either 0 or 1.

Checking Correctness and Security. Since we keep track of the set of possible atomic sequences for any state of the protocol, we can decide for any result action whether it yields the correct output in all cases.

To check privacy, first note that shuffle actions never reveal new critical information: When shuffling with face-up cards, the shuffle may reveal information about which permutation was used to shuffle, but this information is a fresh random variable independent of all previous information. Considering turns or randomized flips, we already identified the condition before: A turn does not violate privacy if for every visible sequence v that may result from the turn, the set C_v of atomic sequences that are compatible with v must fulfill $\sum_{s \in C_v} S(s) = \beta_v \in [0, 1]$ since this exactly means that the probability to observe a visible sequence does not depend on the inputs. As this was a precondition for the derivation rule of randomized flips, being able to construct a diagram by the rules above is a witness to the security of the protocol. (In this sense, Fig. 2 is an alternative proof for the security of the six-card AND protocol of [MS09].)

Las Vegas vs Finite-Runtime. In our formalism, the states of a finite-runtime protocol form a finite tree without restart actions. A Las Vegas protocol, in contrast, makes use of restart actions, or its states form a cyclic or infinite diagram.

4 A Four-Card Las Vegas AND Protocol

We present a secure protocol to compute AND on two bits in committed format and without restarts. An algorithmic description is given in Protocol 1 and a representation in the state calculus of Section 3, from which correctness and privacy can be deduced, is given in Fig. 4.

Note that the state diagram contains a cycle, i.e., it is possible to return to a state that was encountered before. This implies that the protocol is not finite-runtime. However, on the cycle there are two turn operations each of which have a chance of $1/3$ to yield a final state and therefore leave the cycle. The probability to return to a state on the cycle is therefore $(\frac{2}{3})^2 = \frac{4}{9}$ and the probability to take the cycle k times is $(\frac{4}{9})^k$. The expected number of times the cycle is taken is therefore $\sum_{k\geq 0}(\frac{4}{9})^k = (1 - \frac{4}{9})^{-1} = \frac{9}{5}$. In particular, the expected runtime of the protocol is bounded. We summarize our result in the following theorem.

Theorem 1. *There is a secure Las Vegas protocol to compute AND on two bits in committed format and without restarts.*

In contrast to the protocol for general boolean functions presented in Section 7 the shuffle operations are "closed", a circumstance we discuss more closely in Section 8.

Protocol 1. Protocol to compute AND in committed format using four cards. Note that, because of the **goto** operations, no bound on the number of steps can be given.

```
  (shuffle, $\{\mathsf{id}, (1\ 3)(2\ 4)\}$)
  (shuffle, $\{\mathsf{id}, (2\ 3)\}$)
  (turn, $\{2\}$)
  if $v = (?, ♣, ?, ?)$ then
      (turn, $\{2\}$) // turn back
      (shuffle, $\{\mathsf{id}, (1\ 3)\}$)
1     (shuffle, $\{\mathsf{id}, (1\ 2)(3\ 4)\}, \mathcal{F}\colon \mathsf{id} \mapsto 1/3, (1\ 2)(3\ 4) \mapsto 2/3$)
      (turn, $\{4\}$)
      if $v = (?, ?, ?, ♣)$ then
          (result, $1, 2$)
      else if $v = (?, ?, ?, ♡)$ then
          (turn, $\{4\}$) // turn back
          (shuffle, $\{\mathsf{id}, (1\ 3)\}$)
          (perm, $(1\ 3\ 4\ 2)$)
          goto 2
  else if $v = (?, ♡, ?, ?)$ then
      (turn, $\{2\}$) // turn back
      (shuffle, $\{\mathsf{id}, (3\ 4)\}$)
2     (shuffle, $\{\mathsf{id}, (1\ 3)(2\ 4)\}, \mathcal{F}\colon \mathsf{id} \mapsto 1/3, (1\ 3)(2\ 4) \mapsto 2/3$)
      (turn, $\{1\}$)
      if $v = (♡, ?, ?, ?)$ then
          (result, $2, 4$)
      else if $v = (♣, ?, ?, ?)$ then
          (turn, $\{1\}$) // turn back
          (shuffle, $\{\mathsf{id}, (3\ 4)\}$)
          (perm, $(1\ 2\ 4\ 3)$)
          goto 1
```

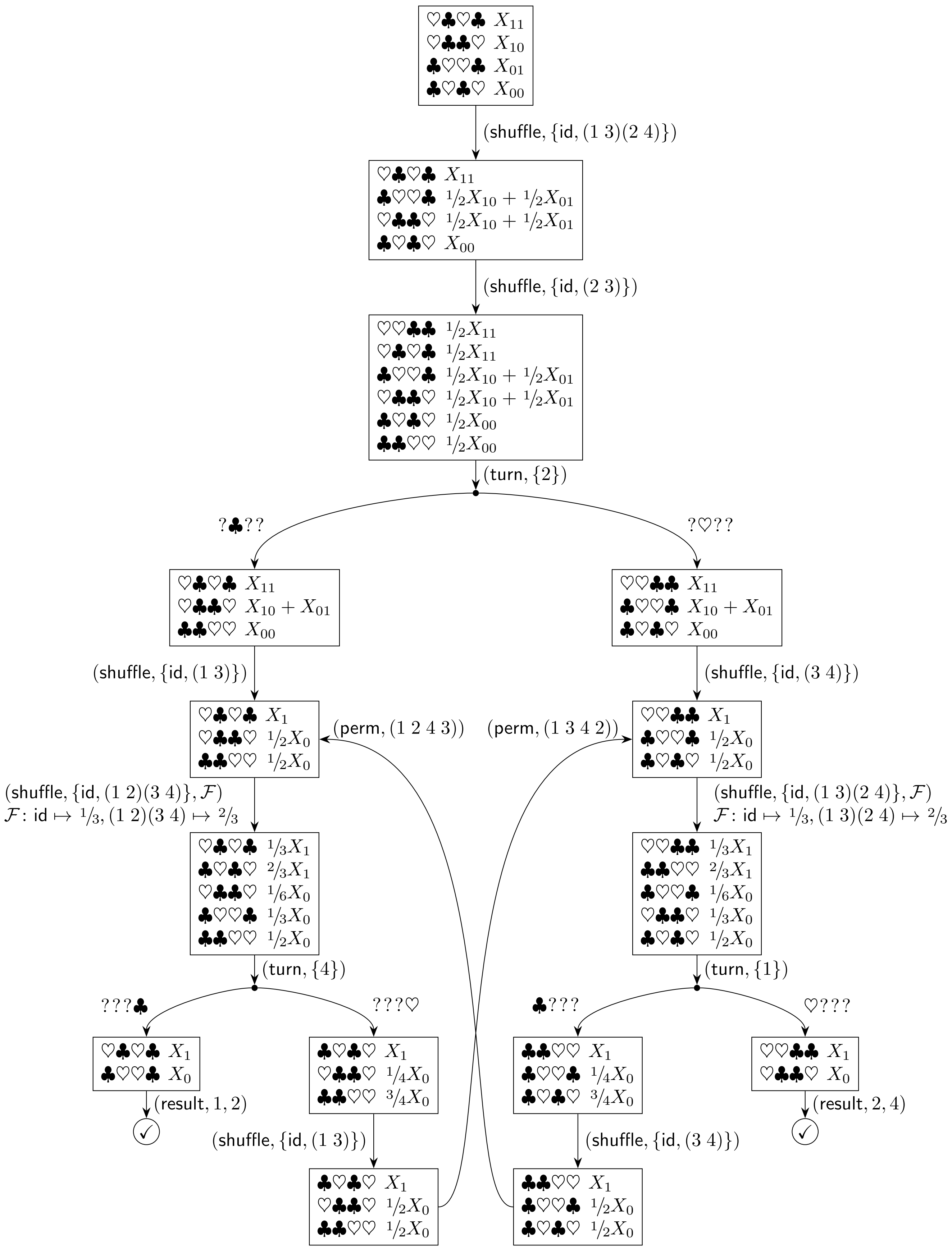


Fig. 4. The four-card Las Vegas AND protocol without restart operations from Protocol 1. Note that we make use of the shorthands $X_1 := X_{11}$ and $X_0 := X_{00} + X_{10} + X_{01}$ and omit the turn actions that merely turn cards back to face-down. Starting at certain points the tree becomes self-similar, which we represent by drawing backwards edges.

5 A Five-Card Finite-Runtime AND Protocol

In the presentation of our five-card finite-runtime AND protocol in committed format, we reuse part of our four-card protocol from Section 4. We just have to show that we can "break out" of the cycle of the four card protocol by using the fifth card. This yields a finite-runtime protocol with at most 12 steps in every execution. Here, the fifth card is chosen to have symbol ♡.

An algorithmic description is given in Protocol 2 and a representation of the crucial component in the state calculus of Section 3, from which correctness and privacy can be deduced, is given in Fig. 5. We summarize our result in the following theorem.

Theorem 2. *There is a secure five-card finite-runtime protocol to compute AND on two bits in committed format.*

Protocol 2. A five-card finite-runtime AND protocol. It proceeds as in Protocol 1 (ignoring card 5) until reaching the line marked as 1, when instead of executing the line, an alternative path is taken using the fifth card.

```
  (shuffle, {id, (1 3)(2 4)})
  (shuffle, {id, (2 3)})
  (turn, {2})
  if $v = (?, ♣, ?, ?, ?)$ then
      (turn, {2}) // turn back
      (shuffle, {id, (1 3)})
⋆     (perm, (1 5 2 4)) // sort in the fifth card
      (shuffle, {id, (5 4 3 2 1)}, $\mathcal{F}$: id $\mapsto 1/3$, (5 4 3 2 1) $\mapsto 2/3$)
      (turn, {5})
      if $v = (?, ?, ?, ?, ♣)$ then
          (result, 4, 3)
      else if $v = (?, ?, ?, ?, ♡)$ then
          (result, 3, 1)
  else if $v = (?, ♡, ?, ?, ?)$ then
      (turn, {2}) // turn back
      (shuffle, {id, (3 4)})
      (shuffle, {id, (1 3)(2 4)}, $\mathcal{F}$: id $\mapsto 1/3$, (1 3)(2 4) $\mapsto 2/3$)
      (turn, {1})
      if $v = (♡, ?, ?, ?, ?)$ then
          (result, 2, 4)
      else if $v = (♣, ?, ?, ?, ?)$ then
          (turn, {1}) // turn back
          (shuffle, {id, (3 4)})
          (perm, (1 2 4 3))
          goto ⋆
```

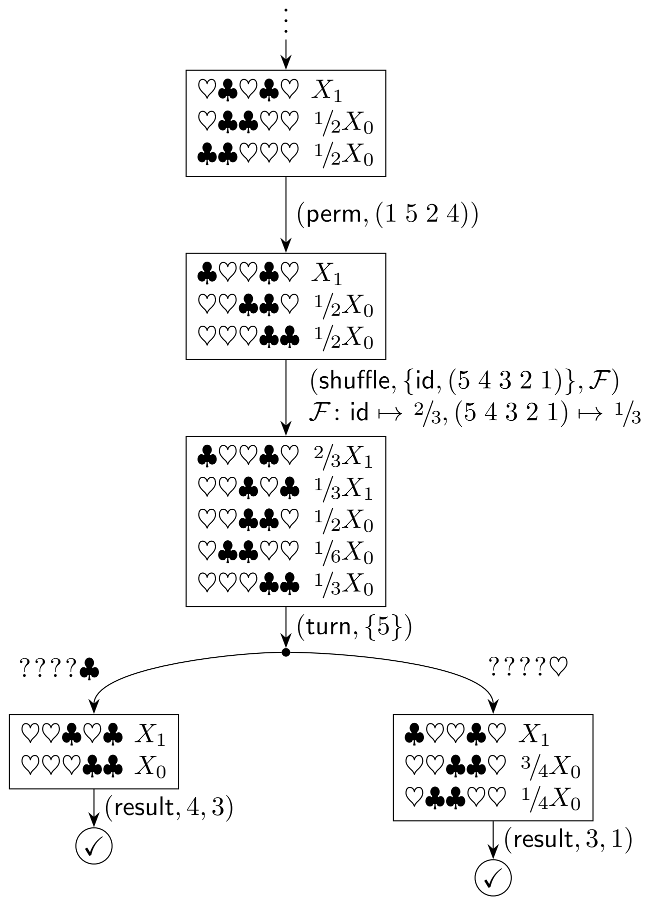


Fig. 5. The crucial part of a five-card finite-runtime AND protocol that allows to "break out" of the cycle in the four-card Las Vegas AND protocol.

6 Finite-Runtime AND Requires Five Cards

There are secure protocols with four cards computing AND in committed format using either the restart operation (see Section 7) or running in cycles for a number of iterations that is finite only in expectation (see Section 4). However, it would be nice to have a protocol that is finite-runtime, i.e., is guaranteed to terminate after a finite number of steps. In the following we show that this is impossible.

To this end, we distinguish several different types of states and later analyze which state transitions are possible. We need the following definitions and observations only for the deck $\mathcal{D} = [\heartsuit, \heartsuit, \clubsuit, \clubsuit]$, but choose to state some of them in a more general form to better convey the underlying ideas.

Definition 4. *Let $\mathcal{P}$ be a protocol with deck $\mathcal{D}$ computing a boolean function f. Let s be an atomic sequence, S a state of $\mathcal{P}$ and $P = S(s)$ the polynomial representing the probability of s in S.*

1. *If P contains only variables X_b with $f(b) = 1$ or $f(b) = 0$, then s is called a* 1-sequence *or* 0-sequence*, respectively.*
2. *If P contains variables of both types, then s is called a* $\bot$-sequence.
3. *We say that S is* of type i/j*, or an* i/j-state*, if its number of 0-sequences and 1-sequences is i and j, respectively, and it does not contain any $\bot$-sequences.*
4. *We call a state S* final *if it does not contain a $\bot$-sequence and there are indices $m, n \in \{1, \ldots, |\mathcal{D}|\}$, such that all 1-sequences have $\heartsuit$ at position m, all 0-sequences have $\clubsuit$ at position m, and the other way round at position n. In that case (result, m, n) is a correct output operation.*

Note that a protocol that produces a $\bot$-sequence cannot be finite-runtime: once the $\bot$-sequence is lying on the table, it is impossible to decide whether the output should be 0 or 1. Thus, any protocol that proceeds to output something without restarting in between produces an incorrect result with positive probability; and any protocol that may use a restart, may take this execution path an unbounded number of times.

Since we are interested in the existence of finite-runtime protocols, we restrict our attention to protocols that never produce $\bot$-sequences. We now bundle a few simple properties about i/j-states in the following lemma.

Lemma 1. Given a secure protocol computing a non-constant boolean function with deck $\mathcal{D}$, consisting of n $\heartsuit$s and m $\clubsuit$s where $n, m \geq 1$, the following holds.

1. In a state of type i/j, we have $i, j \geq 1$, otherwise players could derive the the result, contradicting the committed format property.
2. If a turn in a state S of type i/j can result in two different successor states S_1 and S_2 of type i_1/j_1 and i_2/j_2, respectively, then $i = i_1 + i_2$ and $j = j_1 + j_2$. In particular, $i \geq 2$ and $j \geq 2$.
3. In a state of type i/j resulting from a turn that revealed a $\heartsuit$ or $\clubsuit$ we have $i + j \leq \binom{n+m-1}{n-1}$ or $i + j \leq \binom{n+m-1}{m-1}$, respectively.
4. Let S be a state of type i/j and S' a state of type i'/j' resulting from S via a shuffle operation. Then we have $i' \geq i$, $j' \geq j$.
5. If S is a final state of type i/j, then $i, j \leq \binom{n+m-2}{n-1}$.
6. Two atomic sequences differ in an even number of positions, i.e., have even *distance.*
7. Given an atomic sequence $s \in \mathsf{AtSeq}$, there are
$$\binom{n}{\frac{d}{2}}\binom{m}{\frac{d}{2}}$$
atomic sequences of (even) distance d to s.
8. Any two sequences have distance at most $\min\{2m, 2n\}$.
9. After a single-card turn revealing $\heartsuit$ or $\clubsuit$, any two sequences of the state have distance at most $2n - 2$ or $2m - 2$, respectively.

Theorem 3. *There is no secure finite-runtime four-card AND protocol in committed format.*

Proof. Let P be a secure protocol computing AND with four cards in committed format.

We will define a set of *good states*, denoted by $\mathcal{G}$, that contain all final states but not the starting state and show that any operation on a non-good state will produce at least one non-good state as a successor. From this it is then clear by induction that P is not finite-runtime.

A state S is *good* iff it fulfills one of the following properties:

- S is a 1/1-state,
- S is a 2/2-state,
- S is a 1/2- or 2/1-state containing two atomic sequences of distance 4.

We first observe which state types i/j can occur with our deck: Since there are $6 = \binom{4}{2}$ atomic sequences in total, we need $i + j \leq 6$. By Lemma 1, item 1, states with $i = 0$ or $j = 0$ cannot occur.

Final States are Good. From item 5 in Lemma 1 we know that final states fulfil $i, j \leq 2$ so the only candidate for final states are 1/1, 2/2, 1/2 and 2/1. We need to show that they are good which is true by definition for 1/1 and 2/2. Consider a final 1/2-state (the argument for the 2/1-state is symmetric). Its 0-sequence differs from both 1-sequences in the two positions used for the output. Since the two 1-sequences are distinct, at least one of them must differ from the 0-sequence in another position, meaning they must have distance at least 3 and therefore distance 4 (item 6 in Lemma 1).

Therefore, all final sequences are good, but the start state, which is a 3/1-state, is non-good. Consider an action $\mathsf{act} \in \mathsf{Action}$ that acts on a non-good state. We show that act has a non-good successor state by considering all cases for the type of act:

Non-trivial Single-card Turns. Let S be a non-good state of type i/j, and $S_\heartsuit$ and $S_\clubsuit$ the two possible states after a turn of a single card. From item 2 in Lemma 1, we know that S has to be of type i/j, with $i, j \geq 2$, excluding the case of 2/2, as S is non-good. This leaves the following possible types for S: 2/3, 3/3, 2/4 where we assume without loss of generality that $i \leq j$. The turn partitions the sequences onto the two branches in one of the following ways:

$$\begin{array}{ccccccccccccccc}
 & 2/3 & & & & 3/3 & & & & 3/3 & & & & 2/4 & & & & 2/4 & \\
 / & & \backslash & & / & & \backslash & & / & & \backslash & & / & & \backslash & & / & & \backslash \\
1/1 & & 1/2 & & 1/1 & & 2/2 & & 1/2 & & 2/1 & & 1/2 & & 1/2 & & 1/3 & & 1/1
\end{array}$$

From item 3 in Lemma 1, we know that a state resulting directly from a turn contains at most 3 atomic sequences, thereby ruling out turn-transitions that lead to a 2/2- or 1/3-state. Moreover, any 2/1- or 1/2-state occurring after a turn has the property that all atomic sequences have pairwise distance 2 by item 9 in Lemma 1. By definition, such 2/1-states are non-good. Note that a turn action on a 2/3-state – while producing a good and even final 1/1-state – produces a non-good 1/2-state on the other branch.[4]

Non-branching Shuffles. Now consider a shuffle that produces a unique subsequent state S' of type i'/j'. We want to show that S' is non-good. Using item 4 in Lemma 1 and the fact that a good S' would require $i', j' \leq 2$, we only need to consider the case that S is a non-good state with $i, j \leq 2$, i.e., S is of type 1/2 or 2/1 with pairwise distance 2 – without loss of generality of type 1/2 and with a 0-sequence s_0 and two 1-sequences s_1 and s_1'. We argue that without loss of generality S is of the form

s_0: ♡♡♣♣
s_1: ♡♣♡♣
s_1': ♡♣♣♡

This is because

– S contains a constant column: Let k and l be the positions where s_0 differs from s_1, and m, n the positions where s_0 differs from s_1'. If $\{k, l\}$ and $\{m, n\}$ are disjoint, then s_1 and s_1' have distance 4 – a contradiction. Otherwise $\{k, l, m, n\}$ has size at most 3 so there is one position where all atomic sequences agree.
– The constant column can be assumed to be in position 1 and to contain ♡s. This completely determines the atomic sequences occurring in S. Our choice to pick the 0-sequence is arbitrary, but inconsequential.

If all permutations in the shuffle map 1 to the same $i \in \{1, 2, 3, 4\}$, then S' will have a constant column in position i. Then S' is still of type 1/2 with sequences of pairwise distance 2, so non-good. If there are two permutations in the shuffle that map 1 to different positions $i \neq j$, then S' will contain all three atomic sequences with ♡ in position i and all three atomic sequences with ♡ in position j. There is only one atomic sequence with ♡ in both positions. So S' contains at least $3 + 3 - 1 = 5$ atomic sequences and is therefore non-good.

Other Actions. The hard work is done, but for completeness, we need to consider the remaining actions as well:

Restart. This action is not allowed in our finite-runtime setting.
Result. Since non-good states are non-final this action cannot be applied.
Permutation. This is just a special case of a non-branching shuffle.

[4] Moreover, this is the only way to produce a good state from a non-good state via a turn action. We make use of such a turn in our four-card protocol in Section 4, which did not require finite-runtime. (In contrast to our protocol in Section 7 this allows us to avoid restart actions.)

Trivial turn. If act is a turn operation that can only result in a single visible sequence (the turn is *trivial*), then the outcome of the turn was known in advance and the state does not change.

Multi-card turn. If act turns more than one card, then act can be decomposed into single-card turn actions, turning the cards one after the other. We already know that a single-card turn from a non-good state yields a non-good subsequent state, so following a "trail" of non-good states shows act produces a non-good state as well.

Randomized flip. If act is a randomized flip then consider any turn set T that act might be picked. We already know that turning T yields a non-good subsequent state and this is also a subsequent state of act.

Branching shuffle. If act is a shuffle that produces several subsequent states (this requires shuffling with a face-up card), then restricting the set of allowed permutations to those corresponding to one of the visible sequences yields an ordinary shuffle that therefore yields a single subsequent non-good state. This state is also a subsequent state of act.

This conclusion the proof. $\square$

7 A $2k$-Card Protocol For Any k-ary Boolean Function

The following protocol will compute a k-ary boolean function with $2k$ cards and success probability 2^{-k} in three steps: One shuffle, one turn and one result or restart action. The "hard work" is done in an "irregularly complex" shuffle operation, which may pose practical problems we expand upon in Section 8.

Theorem 4. *For any boolean function $f\colon \{0,1\}^k \to \{0,1\}$ there is a secure Las Vegas protocol in committed format using $2k$ cards. The expected number of* restart *actions in a run is $2^k - 1$.*

Proof. Note first that all unary boolean functions can easily be implemented: The identity and not-function is simple (just output the input or the inversed input) and for the constant functions we may shuffle the two cards (to obscure the input), then turn the cards over, arrange them to represent the constant and then return the positions of the corresponding cards, via result.

We now assume $k \geq 2$. For each input $b = (b_1, b_2, \ldots, b_k) \in \{0,1\}^k$ we define the permutation:

$$\pi_b := (2\ 3)^{1-f(b)} \circ (1\ 2)^{b_1} (3\ 4)^{b_2} \cdots (2k-1\ 2k)^{b_k}.$$

In other words, when applied to an input sequence, π_b first swaps the i-th input bit for each i such that $b_i = 1$. Afterwards, it swaps the second and third card if $f(b) = 0$.

We can now describe the steps of our protocol:

1. (shuffle, $\{\pi_b \colon b \in \{0,1\}^k\}$), i.e., pick $b \in \{0,1\}^k$ uniformly at random and permute the cards with π_b.

2. $(\mathsf{turn}, \{1, 4, 6, 8, \dots, 2k\})$, i.e., turn over the first card and all cards with even indices except 2.
3. If the turn revealed $\clubsuit$ in position 1 and $\heartsuit$ everywhere else, i.e., the visible sequence is $(\clubsuit, ?, ?, \heartsuit, ?, \heartsuit, \dots, ?, \heartsuit)$, then perform $(\mathsf{result}, 2, 3)$. Otherwise, $(\mathsf{restart})$.

For a deeper understanding of what is actually going on, we suggest contemplating on Fig. 6 (which is, admittedly, somewhat intimidating), but correctness and privacy are surprisingly easy to show directly:

Correctness. Assume the input is $b \in \{0,1\}^k$ and a result action is performed. Then the visible sequence after the turn was $(\clubsuit, ?, ?, \heartsuit, ?, \heartsuit, \dots, ?, \heartsuit)$. This means the permutation π done by the shuffle must have first transformed the input sequence to $(\clubsuit, \heartsuit, \clubsuit, \heartsuit, \clubsuit, \heartsuit, \dots, \clubsuit, \heartsuit)$ (before potentially flipping the cards in position 2 and 3). This can be interpreted as the sequence encoding only 0s, therefore π has flipped exactly the card pairs, where the input sequence had $(\heartsuit, \clubsuit)$ encoding 1. This implies $\pi = \pi_b$. From the definition of π_b it is now clear that the output is $(\heartsuit, \clubsuit)$ if $f(b) = 1$ and $(\clubsuit, \heartsuit)$ if $f(b) = 0$.

Privacy. Let v be a visible sequence after the turn step. Consider an input sequence Γ_b belonging to the input $b \in \{0,1\}^k$. The probability that Γ_b yields the visible sequence v in the turn is exactly 2^{-k} since exactly one of the 2^k permutations in the shuffle action swaps the appropriate set of pairs of positions. This means the probability to observe v is 2^{-k} – and thus independent of the input sequence.

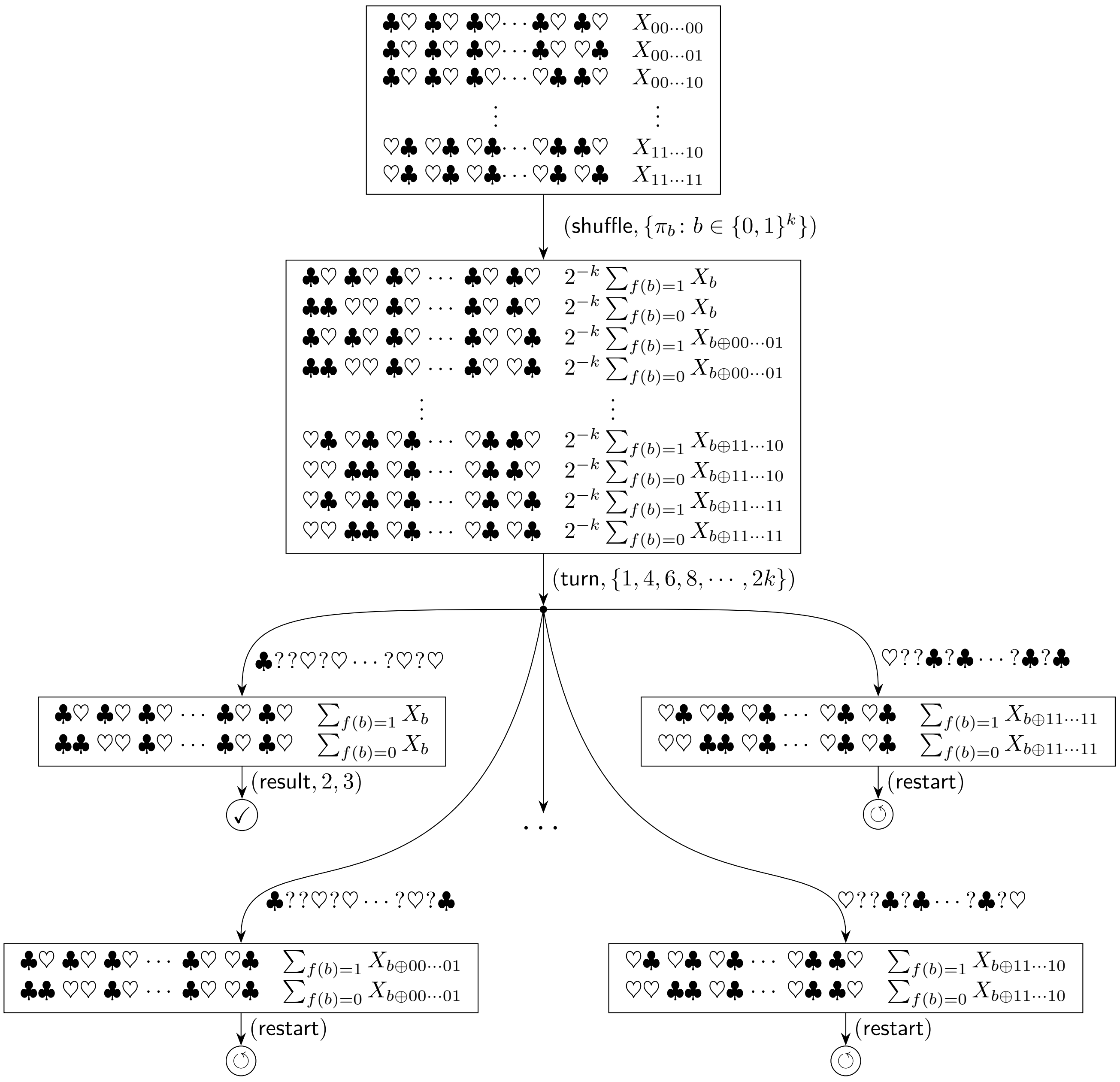
Runtime. The probability to observe $(\clubsuit, ?, ?, \heartsuit, \dots, ?, \heartsuit)$ in the turn step is 2^{-k}, the probability to restart is therefore $1 - 2^{-k}$. This yields a runtime that is finite in expectation – of order $O(2^k)$. □

8 On the Implementation of Shuffle Operations

The shuffle used in the protocol in Section 7, while allowed in the formalism by [MS14a], is of questionable practicality: in general there is no obvious way to perform it in a real world situation with actual people and actual cards such that the players do not learn anything about the permutation that was done in the shuffle. In a weaker form this also applies to the protocols in Sections 4 and 5.

Other shuffle operations, such as $(\mathsf{shuffle}, \{\mathsf{id}, (1\ 2)\})$ that either perform a swap or do nothing, both with probability $\frac{1}{2}$, are unproblematic to implement with two players Alice and Bob: first let Alice perform the shuffle while Bob is looking away and then have Bob perform the shuffle while Alice is looking away. Provided they do not tell each other what they did, to both of them the cards seem to be swapped with probability $1/2$. Here, it is crucial that performing the swap twice yields the identity: one of the allowed permutations.

In general, a shuffle action $\mathsf{act} = (\mathsf{shuffle}, \Pi, \mathcal{F})$ can be implemented in this way if act is *closed*, i.e., $\Pi^2 := \{\pi_1 \circ \pi_2 \mid \pi_1, \pi_2 \in \Pi\} = \Pi$ and *uniform*, i.e., $\mathcal{F}$ is



$X_{00\cdots00}$
$X_{00\cdots01}$
$X_{00\cdots10}$
$X_{11\cdots10}$
$X_{11\cdots11}$
(shuffle, $\{\pi_b\colon b \in \{0,1\}^k\}$)
$2^{-k}\sum_{f(b)=1} X_b$
$2^{-k}\sum_{f(b)=0} X_b$
$2^{-k}\sum_{f(b)=1} X_{b\oplus00\cdots01}$
$2^{-k}\sum_{f(b)=0} X_{b\oplus00\cdots01}$
$2^{-k}\sum_{f(b)=1} X_{b\oplus11\cdots10}$
$2^{-k}\sum_{f(b)=0} X_{b\oplus11\cdots10}$
$2^{-k}\sum_{f(b)=1} X_{b\oplus11\cdots11}$
$2^{-k}\sum_{f(b)=0} X_{b\oplus11\cdots11}$
(turn, $\{1,4,6,8,\cdots,2k\}$)
♣??♡?♡···?♡?♡
♡??♣?♣···?♣?♣
(result, 2, 3)
(restart)
♣??♡?♡···?♡?♣
♡??♣?♣···?♣?♡
(restart)
(restart)

Fig. 6. The $2k$-card protocol for an arbitrary boolean function f of Theorem 4. We use the notation $b_1 \oplus b_2$ to denote the bitwise exclusive-or operation, e.g. $0011 \oplus 0101 = 0110$.

the uniform distribution on Π. Note that our protocols in Sections 4, 5 and 7 use shuffles that are not uniform and/or not closed, see Tables 1 and 2. Therefore, it may be worthwhile to continue studying shuffles in several directions:

- Restrict the computational model to uniform closed shuffles and examine the properties of the new model.
- Replace the action $\mathsf{shuffle}$ of the computational model by an alternative action $\mathsf{playerPerm}$ executed by a single player, while other players are not allowed to look on the table. Here, $(\mathsf{playerPerm}, p, \Pi, \mathcal{F})$ is like $(\mathsf{shuffle}, \Pi, \mathcal{F})$, with the difference that the executing player p learns which permutation has been chosen. As argued above, this at least as powerful as allowing uniform closed shuffles.
- Search for a more clever way to implement shuffles with everyday objects.
- Weaken the honest-but-curious assumption and discuss implementations of shuffles with respect to, e.g., robustness against active attacks.

9 Conclusion

To summarize our results, we have extensively considered the question on tight lower bound on the number of cards for AND protocols, which has been open for several years. We believe that our answer to this question is satisfactory, as we do not only give two concrete AND protocols with different properties, we also show an impossibility result. Apart from the impossibility for perfect copy of a single card in [MS14a], we are the first to give such a type of result. This may be because of the sparsity of good ways to speak about card-based protocols. We believe to have overcome this problem by introducing an elegant "calculus of protocol states" in Section 3. Finally, we give a protocol for evaluating a k-ary boolean function with the theoretical minimum of cards, i.e., the $2k$ cards which are already necessary for encoding the input.

Open Problems. Our paper identifies a number of open problems in the field of card-based cryptographic protocols. This is, for example, how to implement non-closed or non-uniform shuffles and in consequence back up the current computational model with more evidence that its definition is rooted in reality. In the same way, we ask whether there is a finite-runtime five-card protocol using only closed and/or uniform shuffles.

The same set of questions which have been answered in Table 1 can also be asked for general boolean functions: What is the minimal number of cards for finite-runtime protocols with and without closed shuffles. Analogously, a tight lower bound on the number of cards in Las Vegas protocols using only uniform closed shuffles would be interesting.

Acknowledgments. We would like to thank the anonymous reviewers and Gunnar Hartung for helpful comments.

References

[B89] B. den Boer. “More Efficient Match-Making and Satisfiability: The Five Card Trick”. In: *EUROCRYPT ’89, Proceedings*. Ed. by J. Quisquater and J. Vandewalle. LNCS 434. Springer, 1989, pp. 208–217. DOI: 10.1007/3-540-46885-4_23.

[CHL13] E. Cheung, C. Hawthorne, and P. Lee. “CS 758 Project: Secure Computation with Playing Cards”. 2013. URL: https://cs.uwaterloo.ca/~p3lee/projects/cs758.pdf (visited on 02/10/2015).

[CK93] C. Crépeau and J. Kilian. “Discreet Solitary Games”. In: *CRYPTO ’93, Proceedings*. Ed. by D. R. Stinson. LNCS 773. Springer, 1993, pp. 319–330. DOI: 10.1007/3-540-48329-2_27.

[MKS12] T. Mizuki, M. Kumamoto, and H. Sone. “The Five-Card Trick Can Be Done with Four Cards”. In: *ASIACRYPT 2012, Proceedings*. Ed. by X. Wang and K. Sako. LNCS 7658. Springer, 2012, pp. 598–606. DOI: 10.1007/978-3-642-34961-4_36.

[MS09] T. Mizuki and H. Sone. “Six-Card Secure AND and Four-Card Secure XOR”. In: *Frontiers in Algorithmics, FAW 2009, Proceedings*. Ed. by X. Deng, J. E. Hopcroft, and J. Xue. LNCS 5598. Springer, 2009, pp. 358–369. DOI: 10.1007/978-3-642-02270-8_36.

[MS14a] T. Mizuki and H. Shizuya. “A formalization of card-based cryptographic protocols via abstract machine”. In: *International Journal of Information Security* 13.1 (2014), pp. 15–23. DOI: 10.1007/s10207-013-0219-4.

[MS14b] T. Mizuki and H. Shizuya. “Practical Card-Based Cryptography”. In: *Fun with Algorithms, FUN 2014, Proceedings*. Ed. by A. Ferro, F. Luccio, and P. Widmayer. LNCS 8496. Springer, 2014, pp. 313–324. DOI: 10.1007/978-3-319-07890-8_27.

[N$^+$15] T. Nishida, Y. Hayashi, T. Mizuki, and H. Sone. “Card-Based Protocols for Any Boolean Function”. In: *Theory and Applications of Models of Computation, TAMC 2015, Proceedings*. Ed. by R. Jain, S. Jain, and F. Stephan. LNCS 9076. Springer, 2015, pp. 110–121. DOI: 10.1007/978-3-319-17142-5_11.

[NR98] V. Niemi and A. Renvall. “Secure Multiparty Computations Without Computers”. In: *Theoretical Computer Science* 191.1-2 (1998), pp. 173–183. DOI: 10.1016/S0304-3975(97)00107-2.

[S01] A. Stiglic. “Computations with a deck of cards”. In: *Theoretical Computer Science* 259.1-2 (2001), pp. 671–678. DOI: 10.1016/S0304-3975(00)00409-6.