Foundations for Actively Secure Card-Based Cryptography

Alexander Koch
Karlsruhe Institute of Technology (KIT), Germany
alexander.koch@kit.edu

Stefan Walzer
Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de

Abstract

Card-based cryptography, as first proposed by den Boer [4], enables secure multiparty computation using only a deck of playing cards. Many protocols as of yet come with an "honest-but-curious" disclaimer. However, modern cryptography aims to provide security also in the presence of *active attackers* that deviate from the protocol description. In the few places where authors argue for the active security of their protocols, this is done ad-hoc and restricted to the concrete operations needed, often using additional physical tools, such as envelopes or sliding cover boxes. This paper provides the first systematic approach to *active security* in card-based protocols.

The main technical contribution concerns *shuffling* operations. A shuffle randomly permutes the cards according to a well-defined distribution but hides the chosen permutation from the players. We show how the large and natural class of *uniform closed* shuffles, which are shuffles that select a permutation *uniformly* at random from a permutation *group*, can be implemented using only a linear number of helping cards. This ensures that any protocol in the model of Mizuki and Shizuya [17] can be realized in an actively secure fashion, as long as it is secure in this abstract model and restricted to uniform closed shuffles. Uniform closed shuffles are already sufficient for securely computing any circuit [19]. In the process, we develop a more concrete model for card-based cryptographic protocols with two players, which we believe to be of independent interest.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques; Security and privacy → Usability in security and privacy; Theory of computation → Models of computation

Keywords and phrases Card-Based Protocols, Card Shuffling, Secure Multiparty Computation, Active Security, Cryptography without Computers

Related Version This is the full version of [11] at FUN 2021 with DOI: 10.4230/LIPIcs.FUN.2021.17.

Acknowledgements We would like to thank the anonymous reviewers for helpful comments.

1 Introduction

The elegant "five-card trick" of den Boer [4] allows two players – here called Alice and Bob – to compute a logical AND of two private bits, using five playing cards. For instance, if the bit of a player encodes whether they have romantic interest for the other player, the protocol will result in a "yes"-output if and only if there is mutual interest, sparing a party with an unrequited crush the embarrassment of having this information revealed.

More generally, using a deck of playing cards (usually with symbols ♡, ♣), Alice and Bob can jointly compute an arbitrary Boolean function on multiple secret inputs such that neither player learns anything about the input, except, possibly, what can be learned from looking at the output. One distinctive feature is that these protocols do not need a computer, which makes their security tangible. For this reason, they have become popular for introducing secure multiparty computation in lectures and to non-experts.

The key operations that introduce randomness in a controlled manner are shuffles. A shuffle operation causes a sequence of cards to be rearranged according to random permutation such that observers cannot tell which permutation was chosen. The formal computational model of Mizuki and Shizuya [17] permits shuffles with arbitrary distributions on permutations. The model is useful when showing impossibility results and *lower bounds* on cards, cf. [12], but it seems unlikely that all shuffle operations permitted in the model have a convincing real world implementation. This spawned some formal protocols with apparently good parameters, but unclear real-world implementations, especially if active security is a concern [12, Sect. 7]. There is to this day still no *positive* account of what shuffles can be done with playing cards beyond the justification of individual protocols, and even then, most make "honest-but-curious" assumptions, with no guarantees when one of the players deviates from the protocol. In several places in the literature, e.g. [2, Sect. 8] and [12, Sect. 9], the need for achieve actively secure shuffles and protocols has been recognized.

Our Contribution

As security guarantees in the physical world are harder to formalize than in the digital domain[1], we *introduce* a suitable notion of active security. It is slightly non-standard in that we exclude attackers that are too strong. For instance, there is no possible defense against attackers that can arbitrarily turn over cards, cf. Section 6 for a discussion. Moreover, we show how any card-based protocol (in the model of [17]) that is restricted to *uniform closed shuffles* can be transformed into an actively secure protocol that increases the number of cards only by a constant factor. Uniform closed shuffles, namely those that rearrange the cards according to a *uniform* distribution on a permutation *group*, have already been identified in [12, Sect. 8] as a natural class of operations. More importantly, they *suffice to compute any function*[2].

Along the way, we define a new model for card-based cryptography, which we call *two-player protocols*. These, in turn, use *permutation protocols* that allow Alice to apply a $\pi \in \Pi$ of her choosing to a sequence of face-down cards, such that Bob learns nothing about her choice. We believe this to be of independent interest, e.g. as an approach to formalize protocols such as the 3-card AND protocol in [13] that does not fit into the model of Mizuki and Shizuya.

The idea of using "private permutations" as base operations instead of shuffles was first mentioned in [12, Sect. 8]. Independently from our work, these operations are used in [23] to more efficiently perform an instance of the millionaires problem with cards and in [22] for the case of a three-input voting protocol. To formalize security correctly however, we have to distinguish between private permutation in the role of introducing uncertainty, and those which should serve as input (and need special protection), which we discuss in Section 8. There, we also discuss active attacks against two majority protocols from the literature, that have their inputs given by the users's choice of permutation.

---

[1] See also `https://xkcd.com/538/` for a humorous illustration of this fact.

[2] Almost all existing Mizuki–Shizuya protocols, e.g. [3, 4, 6, 15, 14, 16, 19, 18, 25, 24, 31, 1], use only these. This list contains protocols for AND and COPY, hence allowing arbitrary circuits. More general shuffles appear in [2, 12, 29] for the purpose of using less cards. For example, for committed-format AND, restricting to uniform closed shuffles needs exactly one additional card, both in the case of finite runtime and Las Vegas protocols, as shown in [19, 12, 1, 7, 8].

Related Work

The feasibility of general secure multiparty computation with cards was shown in [4, 3, 24, 31]. Since then, researchers proposed a wide range of protocols with different objectives and parameters. One line of research has been to minimize the number of cards used in protocols. In this regard, [19, 16, 12, 28, 7, 1] try to minimize the number of cards for AND, XOR or bit copy protocols, achieving, for instance, the minimum number of four cards for AND protocols both in committed[3] and non-committed format.

With respect to shuffles, all early protocols relied solely on a uniform *random cut*, which is a shuffle causing a cyclic shift on a pile of cards with uniformly random offset. Niemi and Renvall [24, Sect. 3] and den Boer [4] plausibly argue that random cuts can be performed by repeatedly cutting a pile of cards in quick succession, as players are unable to keep track. Other shuffles were justified, including "dihedral group" shuffles [24], [31, Sect. 7], *random bisection cuts* [19, 32] and unequal division shuffles [2, 28, 27].

Other works have investigated the question of active attacks, albeit with a different focus. Mizuki and Shizuya [18] address active security against adversaries who deviate from the input encoding, e.g. giving input $(\heartsuit, \heartsuit)$ instead of $(\heartsuit, \clubsuit)$. We describe in Section 8 how our results subsume this, using a separate input phase. Moreover, they stress the necessity of non-symmetric backs to avoid marking cards by rotating them. Finally, using a secret sharing-like mechanism, they specify how to avoid security breaches by scuff marks on the backs of the cards. [30] describe a method against injection attacks in their model using polarizing plates. Independently, [32] give an implementation of the special case of random bisection cuts, including experiments showing the real-world security of the shuffle.

Besides short ad-hoc discussions of the shuffle security, we believe that this is an exhaustive list of all investigations into active security so far. In particular, the issue of ensuring that only permutations allowed in the protocol description can be performed during a shuffle has not been addressed for non-trivial cases. Due to our constructions spanning multiple layers of abstractions as depicted in Figure 1, we are able to solve this by giving a transformation of passively secure protocols into an actively secure ones, under certain conditions.

2 Preliminaries

Permutations. A *permutation* of a set $X = \{1, \dots, n\}$ for some $n \in \mathbb{N}$, is a bijective map $\pi : X \to X$. The set S_n of all permutations of $\{1, \dots, n\}$ is called *symmetric group*. It has group structure with the identity map id as neutral element and composition $(\circ)$ as group operation. We apply a permutation π of X to a set $S \subseteq X$ by writing $\pi(S) := \{\pi(s) \mid s \in S\}$. We say that π *respects* S if $\pi(S) = S$. In that case, π also respects the complement $X \setminus S$ and we can define the *restriction* of π to S as the permutation τ with domain S and $\tau(s) = \pi(s)$ for all $s \in S$. For elements $x_1, \dots, x_k$ the *cycle* $(x_1\ x_2\ \dots\ x_k)$ denotes the *cyclic* permutation π with $\pi(x_i) = x_{i+1}$ for $1 \le i < k$ and $\pi(x_k) = x_1$ and $\pi(x) = x$ for all x not occurring in the cycle. If several cycles act on pairwise disjoint sets, we write them next to one another to denote their composition. For instance $(1\ 2)(3\ 4\ 5)$ denotes a permutation with mappings $\{1 \mapsto 2, 2 \mapsto 1, 3 \mapsto 4, 4 \mapsto 5, 5 \mapsto 3\}$. Every permutation can be written in such a *cycle decomposition*.

[3] In a committed-format protocol, input and output bits are encoded by the order of two face-down cards (a "commitment") that hides the value and hence, may be used as intermediary input to another protocol without looking at it, while those not in committed format reveal the output and are hence unsuitable for larger circuits.

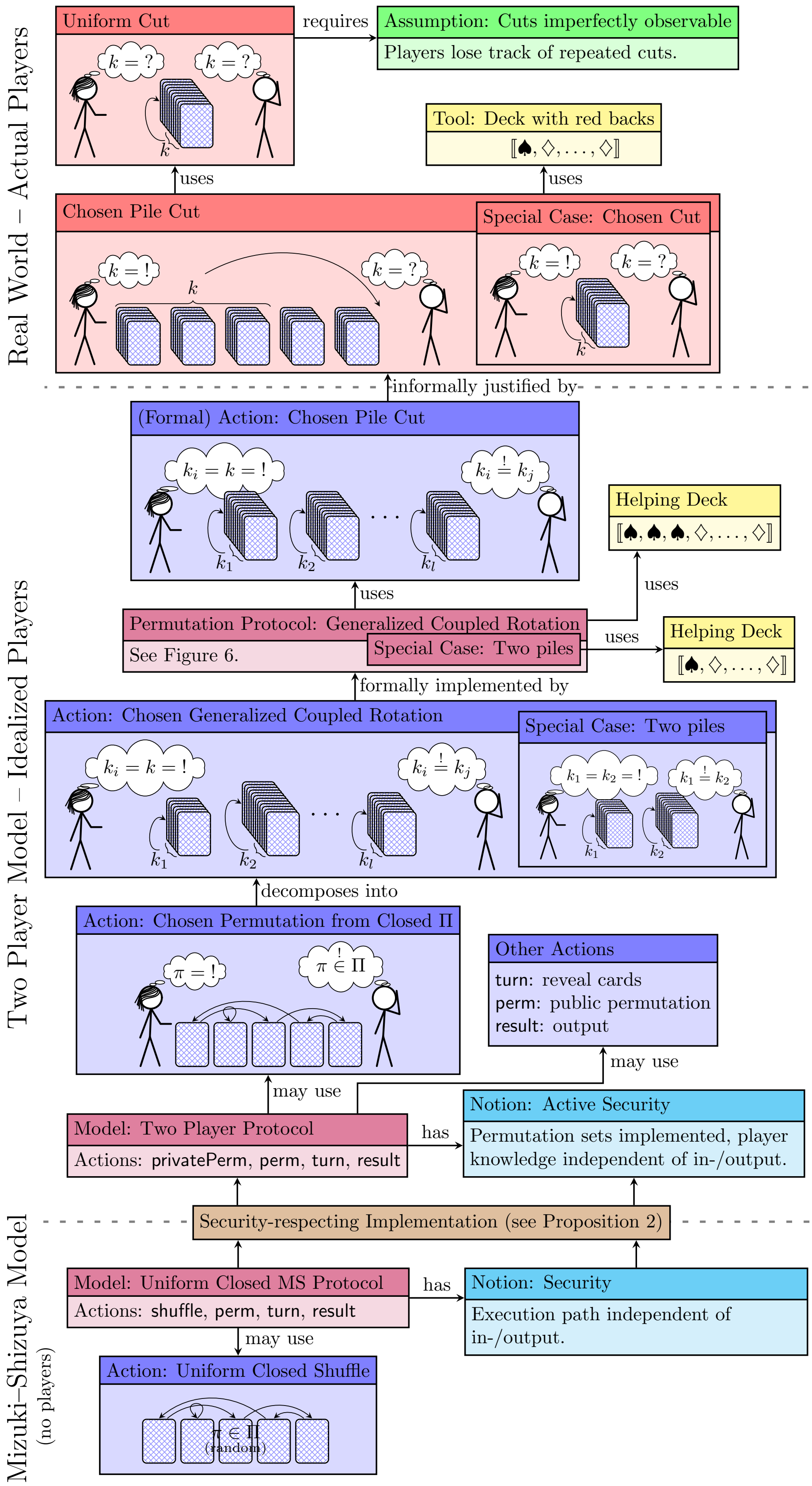


Figure 1 Overview of the content of this paper. The images of Alice and Bob are adapted from xkcd (by Randall Munroe), which is licensed as CC-BY-NC-2.5.

A *uniform cut* (p. 6) rotates a pile of cards by a uniformly random value unknown to Alice and Bob. From this we build *chosen cuts* (p. 6) leaving a pile rotated by a value chosen by Alice but unknown to Bob. When generalized to *chosen pile cuts* (p. 7) and formalized, we obtain a chosen pile cut action that rotates a sequence of equally-sized piles by a value k chosen by Alice. Bob remains oblivious of that value but he can be sure that the cards are not rearranged in any other way. In particular he knows that each pile is rotated by the same amount, even if Alice is dishonest.
With the help of a *permutation protocol* (p. 8) this is extended to the case where piles may have different sizes. This yields *chosen coupled rotations* (p. 8) in the case of two piles and *chosen generalized coupled rotations* (p. 9) in the case of more than two piles.
These are powerful enough to build arbitrary *chosen permutations from a closed permutation set* (p. 10). In that setting, Alice may choose any permutation π from a group of permutations Π. Bob will not learn π but can be sure that no permutation outside the set Π is performed.
A *two player protocol* (p. 11) may make use of these chosen closed permutation actions as well as the *other actions* turn, perm and result.
Uniform closed Mizuki–Shizuya (MS) protocols (p. 16) are a large natural subset of protocols as formalized by Mizuki and Shizuya. Our main result is that for any such protocol there is a two player protocol computing the same function that is *actively secure* (p. 15) if the original protocol is *secure* (p. 13). This *security-respecting implementation* (p. 15) replaces each uniform closed shuffle with two corresponding chosen closed permutations.
Active security is bought with helping cards needed in several places; intuitively to prove the legitimacy of Alice's actions to Bob.

By a *conjugate* of a permutation $\pi \in S_n$ we mean any permutation of the form $\pi' := \tau^{-1} \circ \pi \circ \tau$ where $\tau \in S_n$. For a set $\Pi \subseteq S_n$ of permutations and $\tau \in S_n$ the set $\tau^{-1} \circ \Pi \circ \tau := \{\tau^{-1} \circ \pi \circ \tau \mid \pi \in \Pi\}$ is a *conjugate* of Π. Given an arbitrary sequence of objects $\Gamma = (\Gamma[1], \ldots, \Gamma[n])$ and a permutation $\pi \in S_n$ then *applying* π to Γ yields the sequence $\pi(\Gamma) = (\Gamma[\pi^{-1}(1)], \Gamma[\pi^{-1}(2)], \ldots, \Gamma[\pi^{-1}(n)])$. Intuitively, the object in position i is transported to position $\pi(i)$.

Sets and Groups. If $g_1, g_2, \ldots, g_k \in G$ are group elements, $\langle g_1, \ldots, g_k \rangle$ is the smallest subgroup of G containing $g_1, \ldots, g_k$ and called the *subgroup generated by* $\{g_1, \ldots, g_k\}$. For $g \in G$ the *order* of g is $\mathsf{ord}(g) = |\langle g \rangle| = \min\{k \geq 1 \mid g^k = \mathsf{id}\}$. In the following, a group is implicitly also the set of its elements.

Multisets and Decks. $[\![\diamondsuit, \diamondsuit, \diamondsuit, \spadesuit, \spadesuit]\!]$ is the multiset containing three copies of $\diamondsuit$ and two copies of $\spadesuit$, also written as $[\![3 \cdot \diamondsuit, 2 \cdot \spadesuit]\!]$. If such a multiset represents cards, it is called a *deck*. All cards are implicitly assumed to have the same back, unless stated otherwise. Cards can lie face-up or face-down. When face-down, all cards are indistinguishable (unless they have different backs). When face-up, cards with the same symbol are indistinguishable. Throughout this paper, cards are always face-down with the exception of during a turn operation. To simplify the protocol specification, we immediately turn the card(s) face-down again. Unions of multisets are denoted by $\cup$, disjoint unions are denoted by $+$, e.g. $[\![\diamondsuit, \spadesuit, \spadesuit]\!] \cup [\![\diamondsuit, \heartsuit, \spadesuit, \clubsuit]\!] = [\![\diamondsuit, \heartsuit, \spadesuit, \spadesuit, \clubsuit]\!]$ whereas $[\![\diamondsuit, \spadesuit, \spadesuit]\!] + [\![\diamondsuit, \heartsuit, \spadesuit, \clubsuit]\!] = [\![\diamondsuit, \diamondsuit, \heartsuit, \spadesuit, \spadesuit, \spadesuit, \clubsuit]\!]$.

3 Implementing Cuts and Pile Cuts with Choice

We are interested in procedures that, for a given set $\Pi \subseteq S_n$ of permutations, allow Alice to apply a $\pi \in \Pi$ of her choosing to a sequence of face-down cards, such that Bob learns nothing about her choice, but is certain that Alice did not choose $\pi \notin \Pi$. Also, no player learns anything about the face-down cards if the other player is honest.

In this case we say Π has an *actively secure implementation with choice*, or *is implemented* for short.

Example: Bisection Cut with Envelopes

Mizuki and Sone [19] make use of the following procedure on six cards: The cards in positions 1, 2 and 3 are stacked and put in one envelope and the cards in position 4, 5 and 6 are put into another. Behind her back, Alice then swaps the envelopes or leaves them as they are – her choice. Unpacking yields either the original sequence or the sequence $4, 5, 6, 1, 2, 3$. The bisection cut $\Pi = \{\mathsf{id}, (1\ 4)(2\ 5)(3\ 6)\}$ is therefore implemented (with active security and choice) using two indistinguishable envelopes.

The envelopes ensure that the two groups of cards stay together and their ordering is preserved. The idea is that opening the envelopes behind her back would be impractical and noisy, so even if Alice is malicious, she is limited to the intended options. For a model of secure envelopes, cf. [20, 21].

Example: Unequal Division Shuffle

A bisection cut on n cards can be interpreted as "either do nothing or rotate the sequence by $n/2$ positions". Generalizing this, we now want to "either do nothing or rotate the sequence by l positions" for some $0 < l < n$, i.e. implement $\Pi_l = \{\mathsf{id}, (1\ 2\ \ldots\ n)^l\}$. In [28, 29] a corresponding mechanism is described using two card cases with sliding covers. The card cases behave like envelopes but are heavy enough to mask inequalities in weight caused by different numbers of cards, and support joining the content of two card cases – for details refer to their paper (or Appendix D).

While we are very fond of such creative ideas, in this paper we implement card-based protocols using only one tool: additional cards.

3.1 Cutting the Cards

By the *cut on n cards* we mean the permutation set $\Pi = \langle(1\ \ldots\ n)\rangle$. Alice would *cut* a pile of n cards by taking the top-most k cards (for some $0 < k < n$) from the top of the pile, setting them aside and then placing the remaining $n - k$ cards on top. In this form, Alice can *only approximately* pick k while allowing Bob to approximately observe k. Implementing Π requires fixing both problems.

Uniform Cut

As an intermediate goal we implement a *uniform cut* on n cards, i.e. we perform a permutation $(1\ 2\ \ldots\ n)^k$ for $0 \leq k < n$ chosen uniformly at random and unknown to the players. As proposed in [4], this is done by repeatedly cutting the pile in quick succession until both players lost track of what happened. More formally, under reasonable assumptions, the state of the pile is described by a Markov chain that converges quickly to an almost uniform distribution after a finite number of steps.

Arguably, if the pile is too small, say two cards, the number of cards taken during each cut is perfectly observable. In that case, we put a sufficiently large number c of cards with different backs behind each card, repeatedly cut this larger pile and remove the auxiliary cards afterwards. Note that [32] found it to work well in practice even for $n = 2$ and $c = 3$.[4] We shall not explore this further and use uniform cuts as a primitive in our protocols.

Uniform Cut with Alternating Backs

Later we apply the uniform cut procedure to piles of $n \cdot (\ell + 1)$ cards with n cards of red back, each preceded by ℓ cards of blue back. From a "uniform cut" on such a pile, we expect a cut by $0 \leq k < n \cdot (\ell + 1)$ where $\lfloor k/(\ell + 1)\rfloor$ is uniformly distributed in $\{0, ..., n - 1\}$ and independent of the observable part $k \bmod (\ell + 1)$. We leave it to the reader to verify that the iterated cuts still work under the same assumptions.

Chosen Cut

We now show how to implement $\Pi = \langle(1\ \ldots\ n)\rangle$ with active security and choice. Say Alice wants to rotate the pile of n cards by exactly k positions for a secret $0 \leq k < n$. We propose the process illustrated in Figure 2.

Alice is handed the helping deck $⟦\spadesuit, (n-1) \cdot \heartsuit⟧$ with red backs and secretly rearranges these cards in her hand, putting $\spadesuit$ in position k. The helping cards are put face-down on the table and interleaved with the pile to be cut (each blue card followed by a red card). The $\spadesuit$ is now to the right of the card that was the k-th card in the beginning. To obscure Alice's choice of k, we perform a uniform cut on all cards as described previously. The red helping

---

[4] If not satisfied, the reader may accept some variant of Berry's turntable, cf. [33].

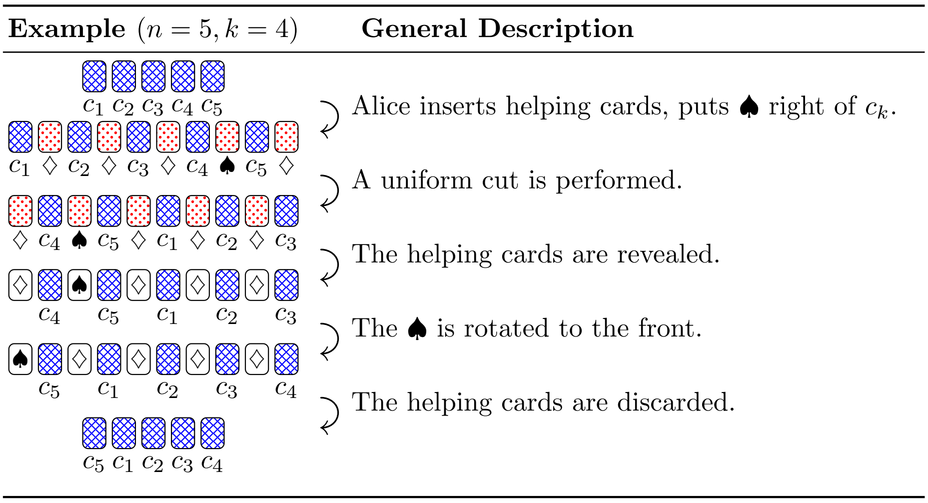


Figure 2 Alice cuts a pile of n cards, here $(c_1, \dots, c_5)$, with back at position k with a helping deck of n helping cards $[\![♠, 4 \cdot ♢]\!]$ with back . In this illustration we annotated face-down cards with the symbol they contain.

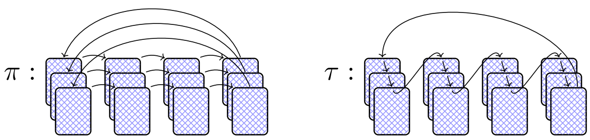


Figure 3 Rotating a sequence of four piles of three cards each by one position *(left)* is described by a permutation π with three cycles of length 4. Alternatively, we can think of π as $\pi = \tau^3$ where τ is the cyclic permutation of length 12 *(right)*.

cards are then turned over. Rotating the sequence so as to put ♠ in front, and removing the helping cards afterward leaves the cards in the desired configuration. Bob is clueless about k since he only observes the position of ♠ *after* the cut, which is independent of the position of ♠ before the cut (which encodes k).

Chosen Pile Cut

Chosen cuts can be generalized in an interesting way. Given n piles of ℓ cards each and $0 \leq k < n$, Alice wants to rotate the sequence of piles by exactly k positions, meaning the i-th pile will end up where pile $i + k$ has been (modulo n). Again, k must remain hidden from Bob and he, on the other hand, wants to be certain that Alice does not tamper with the piles in any other than the stated way. Note that this is equivalent to cutting a pile of $n\ell$ cards where only cutting by multiples of ℓ is allowed, see Figure 3. In that interpretation, the i-th pile is made up of the cards in positions $(i - 1)\ell + 1, \dots, i\ell$.

We apply the same procedure as before with n helping cards, except this time, instead of a single blue card we have ℓ blue cards (a pile) before each of the n gaps that Alice may fill with her red deck $[\![♠, (n-1) \cdot ♢]\!]$. Now the special ♠-card marks the end of the k-th pile and is (after a uniform cut) rotated to the beginning of the sequence, ensuring that after removing the helping cards again we end up having rotated the $n \cdot \ell$ cards by a multiple of ℓ as desired. Note that, uniform (non-chosen) pile cuts have been proposed in [6] as “pile-scramble shuffles”, with an implementation using rubber bands, clips or envelopes.

Summary

If $\Pi = \langle(1\ 2\ \dots\ n \cdot \ell)^\ell\rangle$ for $n, \ell \in \mathbb{N}$, then Π is implemented with active security and choice using the helping deck $[\![\spadesuit, (n{-}1) \cdot \heartsuit]\!]$. For $\ell = 1$ it is called a *cut*, for $\ell > 1$ a *pile cut*. We use the same name for conjugates of Π, i.e. if cards are relabeled. Any subset $\emptyset \neq \Pi' \subset \Pi$ of a (pile) cut is also implemented: Alice places $\spadesuit$ only in some positions, the others are publicly filled with $\heartsuit$.

4 Permutation Protocols for Arbitrary Groups

We introduce a formal concept that allows to compose simple procedures to implement more complicated permutation sets.

▶ **Definition 1.** *A* permutation protocol $\mathcal{P} = (n, \mathcal{H}, \Gamma, A)$ *is given by a number n of* object cards*, a deck of helping cards $\mathcal{H}$ with initial arrangement* $\Gamma\colon \{n+1, \dots, n+|\mathcal{H}|\} \to \mathcal{H}$*, and a sequence A of* actions *where each action can be either*

- $(\mathsf{privatePerm}, \Pi)$ *for* $\Pi \subseteq S_{n+|\mathcal{H}|}$ *implemented with active security and choice, and respecting* $\{1, \dots, n\}$ *(i.e.* $\forall \pi \in \Pi\colon \pi(\{1, \dots, n\}) = \{1, \dots, n\}$*), or*
- (check, p, o) *for a position p of a helping card (i.e.* $n < p \leq n + |\mathcal{H}|$*) and an expected outcome* $o \in \mathcal{H}$*.*

Indeed, consider the following procedure: We start with n object cards lying on a table (positions $1, \dots, n$). We place the sequence Γ next to it, at positions $n+1, \dots, n+|\mathcal{H}|$, and go through the actions of $\mathcal{P}$. Whenever the action $(\mathsf{privatePerm}, \Pi_i)$ is encountered, we use the procedure $\mathcal{P}_i$ implementing Π_i to let Alice apply a permutation on the current sequence. When an action (check, p, o) is encountered, the p-th card is revealed. If its symbol is o, Bob continues, otherwise he aborts, declaring Alice as dishonest. In the end, the helping cards are removed, yielding a permuted sequence of object cards. (All permutations respect $\{1, \dots, n\}$, hence, the helping and the object cards remain separated).

We are interested in the set $\mathsf{comp}(\mathcal{P}) \subseteq S_{n+|H|}$ of permutations *compatible* with $\mathcal{P}$. If there are k $\mathsf{privatePerm}$ actions with permutations sets $\Pi_1, \dots, \Pi_k$ and $\pi_i \in \Pi_i$, then $\pi_k \circ \ldots \circ \pi_1$ is compatible with $\mathcal{P}$ if each check *succeeds*, meaning if (check, p, o) happens after the i-th $\mathsf{privatePerm}$ action (and before the $i+1$st, if $i < k$) then $\Gamma[(\pi_i \circ \ldots \circ \pi_1)^{-1}(p)] = o$. We argue that this implements $\Pi' = \mathsf{comp}(\mathcal{P})|_{\{1,\dots,n\}}$ using $\mathcal{H}$ (and, possibly, helping cards to implement Π_i).

Alice can freely pick any $\pi' \in \Pi'$; using an appropriate decomposition, all checks will succeed. In this case, Bob knows that the performed permutation is from Π'. No player learns anything about the object cards (only helping cards are turned) and conditioned on Alice being honest, the outcome of the checks is determined, so Bob learns nothing about π'.

Coupled Rotations

Let $\varphi = (1\ 2\ \dots\ s)$, $\psi = (s{+}1\ s{+}2\ \dots\ s{+}t)$, and assume $s < t$. For $\pi = \psi \circ \varphi = \varphi \circ \psi$ we call $\Pi = \{\pi^k \mid 0 \leq k < s\}$ the *coupled rotation* with parameters s and t. Note that Π is not a group since $\pi^s \notin \Pi$. We aim to implement Π. We make use of a helping deck $[\![\spadesuit, (t{-}1) \cdot \heartsuit]\!]$ available in positions $H = \{h_0, h_1, \dots, h_{t-1}\}$ with $\spadesuit$ at position h_0. Then define $\hat{\varphi} := \varphi \circ (h_0\ \dots\ h_{s-1})$ and $\hat{\psi} := \psi \circ (h_0\ \dots\ h_{t-1})^{-1}$ and consider the permutation protocol $\mathcal{P}$ in Figure 5 (left), and Figure 4 for illustration. The idea here is that Alice may choose k and k' and perform $\hat{\varphi}^k$ and $\hat{\psi}^{k'}$ to the sequence. However, k is "recorded" in the configuration of a helping sequence and $-k'$ is "added" on top. A check ensures that the

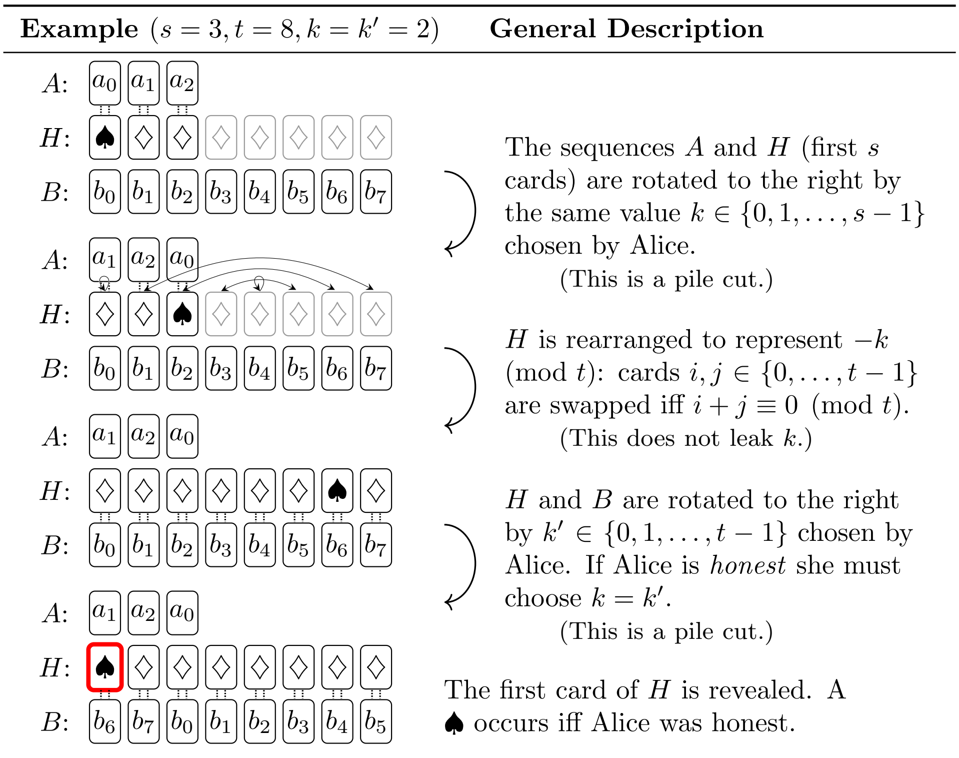


Figure 4 The sequence A of length s and B of length t are to be rotated by the same value k chosen privately by Alice. A helping sequence ensures that the same value is used. All cards are face-down, except for the highlighted card in the last step. The dotted lines indicate that cards are belonging to the same pile in a pile cut, i.e. they maintain their relative position during the cut. The rearrangement of the helping cards is useful in this visualization (so that H and B can be rotated in the same direction) but is not reflected in the formal description.

helping sequence is in its original configuration, implying $k = k'$ as required. Note that $\langle\hat{\varphi}\rangle$ and $\langle\hat{\psi}\rangle$ are pile cuts, which we already know how to implement. In total, we implemented

$$\begin{aligned}\mathsf{comp}(\mathcal{P}) &= \{\hat{\psi}^{k'} \circ \hat{\varphi}^{k} : 0 \le k < s, 0 \le k' < t, \Gamma[(\hat{\psi}^{k'} \circ \hat{\varphi}^{k})^{-1}(h_0)] = \spadesuit\}|_{\{1,\dots,n\}} \\ &= \{\hat{\psi}^{k'} \circ \hat{\varphi}^{k} : 0 \le k < s, 0 \le k' < t, k' = k\}|_{\{1,\dots,n\}} \\ &= \{\psi^{k} \circ \varphi^{k} : 0 \le k < s\}|_{\{1,\dots,n\}} = \Pi.\end{aligned}$$

Products, Conjugates and Syntactic Sugar

The protocol in Figure 5 (middle) implements $\Pi_2 \circ \Pi_1$ using Π_1 and Π_2, showing that if Π_1 is implemented using $\mathcal{H}_1$ and Π_2 is implemented using $\mathcal{H}_2$, then $\Pi_2 \circ \Pi_1$ is implemented using $\mathcal{H}_1 \cup \mathcal{H}_2$. As a corollary, if Π is implemented using $\mathcal{H}$ then so is any conjugate $\Pi' = \{\pi^{-1}\} \circ \Pi \circ \{\pi\}$. Figure 5 (right) uses (perm, π) instead of $(\mathsf{privatePerm}, \{\pi\})$ to emphasize that such deterministic actions can be carried out publicly.

Generalized Coupled Rotations

We generalize the idea of a coupled rotation to more than two sequences. Let $\pi \in S_n$ with cycle decomposition $\pi = \varphi_0 \circ \dots \circ \varphi_\ell$ for $\ell \ge 2$ and increasingly ordered cycle lengths

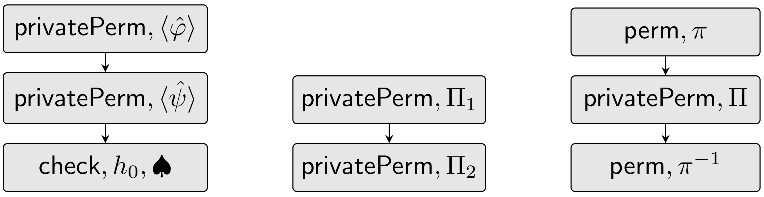


Figure 5 Protocols implementing a coupled rotation (*left*), the product of two permutation sets (*middle*) and the conjugation of a permutation set (*right*).

$t_0 \le t_1 \le t_2 \le \ldots \le t_\ell$. We aim to implement $\Pi = \{\pi^k \mid 0 \le k < t_0\}$ using $t_\ell + 2 \cdot t_0$ helping cards, originally available in the following positions which we label as shown.

$$\underbrace{\overset{\spadesuit}{m_0}\ \overset{\heartsuit}{m_1}\ \overset{\cdots}{\ldots}\ \overset{\heartsuit}{m_{t_\ell-1}}}_{\text{main}}\ \underbrace{\overset{\spadesuit}{x_0}\ \overset{\heartsuit}{x_1}\ \overset{\cdots}{\ldots}\ \overset{\heartsuit}{x_{t_0-1}}}_{\text{temp}}\ \underbrace{\overset{\spadesuit}{s_0}\ \overset{\heartsuit}{s_1}\ \overset{\cdots}{\ldots}\ \overset{\heartsuit}{s_{t_0-1}}}_{\text{store}}$$

We think of the three areas as "registers" *containing* values indicated by the position of ♠ (initially 0). The registers have associated rotations:

$$\psi_{\mathsf{temp}} := (x_0\ \ldots\ x_{t_0-1}), \qquad \psi_{\mathsf{store}} := (s_0\ \ldots\ s_{t_0-1}), \qquad \psi_i := (m_0\ \ldots\ m_{t_i-1}).$$

The protocol's idea is that Alice performs $\varphi_0^{k_0} \circ \cdots \circ \varphi_\ell^{k_\ell}$ and checks will ensure $k_0 = k_1 = \ldots = k_\ell$. To this end, k_0 is recorded in the store register (we use $\langle \varphi_0 \circ \psi_{\mathsf{store}} \rangle$). Then, for each round $i \in \{1, 2 \ldots, \ell-1\}$ the value k_0 is cloned into the main register by first swapping it to the temp register and then moving it to the store *and* main register using $\psi_{\mathsf{copy}} := \psi_{\mathsf{temp}}^{-1} \circ \psi_{\mathsf{store}} \circ \psi_0$. The cloned copy of k_0 in main is consumed when forcing Alice to do $\widehat{\varphi}_i^{k_0}$ where $\widehat{\varphi}_i := \varphi_i \circ \psi_i^{-1}$. The last round is similar. Using the following two swappings, the protocol is formally given in Figure 6.

$$\pi_{\mathsf{store}\leftrightarrow\mathsf{tmp}} := (s_0\ x_0) \ldots (s_{t_0-1}\ x_{t_0-1}), \pi_{\mathsf{store}\leftrightarrow\mathsf{main}} := (s_0\ m_0) \ldots (s_{t_0-1}\ m_{t_0-1}).$$

We now check that this implements the generalized coupled rotation Π using the helping cards $[\![3 \cdot \spadesuit, (t_\ell{+}2t_0{-}3) \cdot \heartsuit]\!]$, cf. Appendix A. The main ingredient is the loop invariant:

If $\pi \in S_{n+2t_0+t_\ell}$ is compatible with the actions until after the i-th execution of the loop and S is the starting sequence then there exists $k \in \{0, \ldots, t_0-1\}$ such that:
- $\pi|_{\{1,\ldots,n\}} = \varphi_i^k \circ \ldots \circ \varphi_1^k \circ \varphi_0^k$,
- *in $\pi(S)$ all registers contain 0 except for store, which contains k.*

We remark that by introducing additional check steps, any subset of a generalized coupled rotation can be implemented as well.

Subgroups of S_n

Generalized coupled rotations are sufficient for:

▶ **Proposition 2.** *Any subgroup Π of S_n can be implemented with active security and choice using only the helping deck $[\![3 \cdot \spadesuit, (n-3) \cdot \heartsuit]\!]$ for (generalized) coupled rotations and the helping deck $[\![\spadesuit, (n{-}1) \cdot \heartsuit]\!]$ for (pile) cuts.*

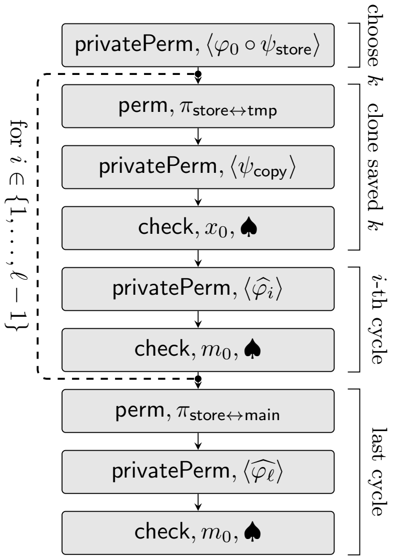


Figure 6 Protocol to implement a generalized coupled rotation with $\ell + 1$ cycles of length $t_0, t_1, \dots, t_\ell$. Notation is explained in the text.

Proof. Note that $\Pi = \prod_{\pi \in \Pi} \langle \pi \rangle$, i.e. Π can be written as the product of cyclic subgroups. Moreover, any cyclic subgroup can be written as $\langle \pi \rangle = \{\pi^0, \dots, \pi^{k-1}\}^\ell$, where k is the length of the shortest cycle in the cycle decomposition of π and $\ell = \lceil \mathsf{ord}(\pi)/(k-1) \rceil$. Hence, Π can be written as the product of rotations and (generalized) coupled rotations, each of which are implemented with the required helping decks. Using the implementation of products (page 9), we are done. ◀

A *simple* decomposition of Π into products of previously implemented permutation sets is desirable to keep the permutation protocol simple. We do not consider this here and merely state that $|\Pi|$ is an upper bound on the number of terms.

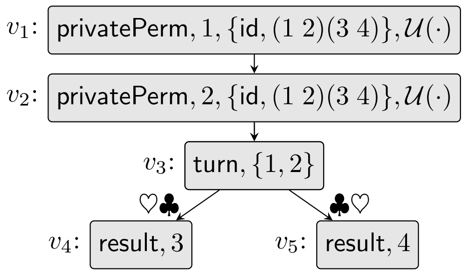
5 Computational Model with Two Players

In the following, two players jointly manipulate a sequence of cards to compute a (possibly randomized) function, i.e. they transform an input sequence into an output sequence. Both have incomplete information about the execution and the goal is to compute with no player learning anything about input or output[5].

Two Player Protocols

A *two player protocol* is a tuple $(\mathcal{D}, U, Q, A)$ where $\mathcal{D}$ is a deck, U is a set of input sequences, Q is a (possibly infinite, computable) rooted tree with labels on some edges, and $A\colon V(Q) \to$ Action is an action function that assigns to each vertex an action which can be perm, turn, result, and privatePerm, with parameters as explained below. All input sequences have the same length n and are formed by cards from $\mathcal{D}$. Vertices with a perm or privatePerm action have exactly one child, vertices with a result action have no children, and those with a turn

[5] An explanation of our security notions follows in Section 6.



$$\mathcal{D} = [\![3 \cdot \clubsuit, 2 \cdot \heartsuit]\!],$$

$$\begin{aligned} U = \{&(\clubsuit, \heartsuit, \clubsuit, \clubsuit), \\ &(\clubsuit, \heartsuit, \heartsuit, \clubsuit), \\ &(\heartsuit, \clubsuit, \clubsuit, \clubsuit), \\ &(\heartsuit, \clubsuit, \heartsuit, \clubsuit)\}, \end{aligned}$$

Q, A: as shown on the left

Figure 7 A protocol example in the two player model, with possible execution trace: $(I = (\heartsuit, \clubsuit, \heartsuit, \clubsuit), O = (\heartsuit), \mathcal{T}_1 = (\mathsf{id}), \mathcal{T}_2 = ((1\ 2)(3\ 4)), W = (v_1, v_2, v_3, v_5))$. This is an actively secure implementation of the AND protocol in [14, Sect. 3.2]. The first two cards encode an input a as $(\clubsuit, \heartsuit) \triangleq 0$, $(\heartsuit, \clubsuit) \triangleq 1$, the third card encodes an input b as $\clubsuit \triangleq 0$, $\heartsuit \triangleq 1$. This encoding is also used for output $a \wedge b$.

action have one child for each possible sequence of symbols the turned cards might conceal, and the edge to that child is annotated with that sequence.

When a protocol is *executed on an input sequence* $I \in U$, we start with the face-down sequence $\Gamma = I$ at the root of Q and empty *permutation traces* $\mathcal{T}_1$ and $\mathcal{T}_2$ for players 1 and 2, respectively. Execution proceeds along a descending path in Q and for each vertex v that is encountered, the action $A(v)$ is executed on the current sequence of cards:

(perm, π) for a permutation $\pi \in S_n$. This replaces the current sequence Γ by the permuted sequence $\pi(\Gamma)$. Execution proceeds at the unique child of v.

(turn, T) for some set $T \subseteq \{1, \ldots, n\}$. For $T = \{t_1 < t_2 < \ldots < t_k\}$, the cards $\Gamma[t_1], \ldots, \Gamma[t_k]$ are turned face-up, revealing their symbols. The vertex v must have an outgoing edge labeled $(\Gamma[t_1], \ldots, \Gamma[t_k])$. Execution proceeds at the corresponding child after the cards are all turned face-down again.

(privatePerm, $p, \Pi, \mathcal{F}(\cdot)$) for a player $p \in \{1, 2\}$, a permutation set $\Pi \subseteq S_n$ and $\mathcal{F}$ being a parameterized distribution on Π. Formally, $\mathcal{F}$ is a function that maps the current permutation trace $\mathcal{T}_p$ of player p to a distribution $\mathcal{F}(\mathcal{T}_p)$ on Π. If $\mathcal{F}(\mathcal{T}_p)$ is the uniform distribution on Π for each $\mathcal{T}_p$ we denote this as $\mathcal{U}(\cdot)$. Player p picks a permutation $\pi \in \Pi$. The current sequence Γ is replaced by the permuted sequence $\pi(\Gamma)$ and π is appended to the player's permutation trace $\mathcal{T}_p$. If player p is *honest* she picks π according to $\mathcal{F}(\mathcal{T}_p)$. Execution proceeds at the unique child of v.

(result, $p_1, \ldots, p_k$) for distinct positions $p_1, \ldots, p_k \in \{1, \ldots, n\}$. Execution terminates with *output* $O = (\Gamma[p_1], \ldots, \Gamma[p_k])$ encoded by face-down cards.

The execution yields an *execution trace* $(I, O, \mathcal{T}_1, \mathcal{T}_2, W)$, containing input, output, permutation traces of the players and the descending path W in Q that was taken, cf. Figure 7. The output of non-terminating protocols is $O = \bot$. Note that we will use permutation protocols from Section 4 in the privatePerm steps, however we use them as black boxes. In particular, the actions specific to permutation protocols (e.g. check) are not part of two player protocols. We say $\mathcal{P}$ is *implemented* using a helping deck $\mathcal{H}$ if each permutation set of a privatePerm action is implemented using $\mathcal{H}$ (as in Section 3). The way we define it, existence, implementability and security of a protocol are separate issues. Security is discussed next.

6 Passive and Active Security

Intuitively, an implemented protocol is (information-theoretically) secure if no player can derive any statistical information about input or output from the choices and observations they make during the execution of the protocol. So the first question is, what information does a player obtain, say Alice, that could potentially be relevant? At first we consider the setting where both players are honest. Surely, Alice knows the public information W, i.e. the execution path of the protocol run, in which the sequence of actions and their parameters are implicit. For each action along W she may have obtained additional information during its execution. To get a complete picture, we go through all types of actions:

- turn actions reveal some card symbols. However, as each outcome corresponds to a unique child vertex where execution continues, this information is already implicit in W.
- perm actions are deterministic and reveal no information. The same is true for result actions. Note that they only *indicate* the position of the output, not reveal it.
- For privatePerm actions, the observations that can be made depend on the implementation. If the protocols are implemented in our sense (see Section 3) and Alice is the active player then Alice learns nothing of relevance except her own choice of permutation (which is recorded in her permutation trace) and, since Alice is honest, Bob learns nothing at all.

So the only potentially relevant information player p has with regards to input and output is W and $\mathcal{T}_p$. Therefore it is adequate to define:

▶ **Definition 3** (Passive Security). *A two player protocol $\mathcal{P} = (\mathcal{D}, U, Q, A)$ is* secure against passive attackers *if for any random variable $I \in U$ the following holds: If $(I, O, \mathcal{T}_1, \mathcal{T}_2, W)$ is the execution trace when executing $\mathcal{P}$ with honest players on input I, then (I, O) is independent of $(\mathcal{T}_p, W)$ for both $p \in \{1, 2\}$.*

Delegated Computation

Passive security implies that if a player has no prior knowledge about in- or output, executing the protocol leaves her in this oblivious state. In particular, by following the protocol the players implement what we call an *oblivious delegated computation* where the computation is performed on secret data (provided by a third party), and the output is not revealed to the executers.

Note that this setting differs from the *standard multiparty computation setting*, where players provide part of the input and usually the output is sent to the players in non-committed (non-hiding) form, i.e., learned by the players. In this case, security means that the players learn nothing *except* what can be deduced from the facts they are permitted to know. It is important to understand that our definition is still adequate for such cases, as *any protocol that is secure in the delegated computation setting is also secure if players have (partial) information about input and output.* The formal reason is the basic fact that for any event E relating only to (I, O), i.e., E is independent of $(\mathcal{T}_p, W)$, conditioning the probability space on E will retain the independence of (I, O) and $(\mathcal{T}_p, W)$.

Moreover, protocols secure in the delegated setting are flexibly applicable in different contexts, making it a very suitable framework. For example, non-delegateable (non-committed input format) protocols which can only be performed by players knowing the input (cf. [13, 23, 22]) cannot be transferred to the delegated setting and are hence unsuitable for use with hidden intermediate results from previous computations. Hence, we protect the output and do not assume knowledge of the inputs. This is a natural setting for card-based cryptography, as all committed-format protocols in the literature achieve this notion, it ensures that the protocols can be used in larger protocols, and it is at least as secure as the other notions, due to the information-theoretic setting.

The above definition of passive security is sufficient if players can be trusted to properly execute the protocol. In that case any privatePerm action can directly be performed by the specified player while the other player looks away. Of course, our main concern here is the situation where looking away is not an option.

Permutation Security and Active Security

To argue about security in the presence of a malicious player, we must first discuss what such a player may do. Doing this rigorously would require to closely model the physical world, which allows for different threats than in the usual cryptographic settings. We certainly have to assume physical restrictions, as otherwise we cannot achieve anything.[6] For example, as our security relies on the possibility of keeping face-down cards, we must assume that an attacker does not resort to certain radical means that immediately and unambiguously identify her as an attacker. (However, note that we can protect against such active attackers which turn over cards by the generic "private circuit" compiler due to Ishai, Sahai, and Wagner [5].) Hence, we can assume that she does not interfere with the correct execution of perm and turn actions, nor does she, in open violation of the protocol, spontaneously seize or turn over some of the cards or mark them in any way.

On the other hand we can plausibly argue that certain mechanisms are sufficient to counter attacks other than those that our paper is concerned with. We may argue that the cards could be put into envelopes, and any attempt to reveal its contents contrary to the protocol will be countered by the cautious other players jumping in to physically abort the protocol in that case.

Concerning an operation $(\mathsf{privatePerm}, \text{Alice}, \Pi, \mathcal{F}(\cdot))$ with implemented Π, there is by definition of *implemented permutation set* no possibility for Alice to perform a permutation $\pi \notin \Pi$. If she causes a permutation protocol to fail, Bob aborts the execution before any sensitive information is revealed. Otherwise, Alice is limited to disrespecting $\mathcal{F}(\cdot)$. This is captured as follows:

▶ **Definition 4.** *Let $\mathcal{P} = (\mathcal{D}, U, Q, A)$ be a two player protocol.*

(i) *A* permutation attack *ξ* on *$\mathcal{P}$* as player *$p \in \{1, 2\}$ specifies for each vertex $v \in V(Q)$ with an action of the form $A(v) = (\mathsf{privatePerm}, p, \Pi, \mathcal{F}(\cdot))$, a permutation $\xi(v) \in \Pi$. Replacing such $\mathcal{F}(\cdot)$ with the (point) distributions that always choose $\xi(v)$, yields the* attacked protocol *$\mathcal{P}^\xi$.*

(ii) *An attack ξ is* unsuccessful *if the following holds. Whenever $I \in U$ is a random variable denoting an input and $(I, O, \mathcal{T}_1, \mathcal{T}_2, W)$ and $(I, O^\xi, \mathcal{T}_1^\xi, \mathcal{T}_2^\xi, W^\xi)$ are the resulting execution traces of $\mathcal{P}$ and $\mathcal{P}^\xi$, then for any values i, o, w:*

$$\Pr[W^\xi{=}w] > 0 \Longrightarrow \Pr[(I, O^\xi){=}(i, o) \mid W^\xi{=}w] = \Pr[(I, O){=}(i, o)]. \qquad (\star)$$

(iii) *We say $\mathcal{P}$ is* secure against permutation attacks *if each permutation attack on $\mathcal{P}$ is unsuccessful.*

In light of our discussion above we finally define:

[6] We do not get ultimately strong guarantees for the physical actions such as in quantum cryptography, where, if (a subset of) quantum theory is true, no adversary can predict a randomness source, no matter what she does physically.

▶ **Definition 5.** *A two player protocol* $\mathcal{P} = (\mathcal{D}, U, Q, A)$ *has an* actively secure implementation *if each permutation set* Π *occurring in a* privatePerm *action is implemented and* $\mathcal{P}$ *is secure against permutation attacks.*

Intuitively, a protocol has permutation security if: No matter what permutations a player chooses ($\forall\xi$), and no matter what the turn actions end up revealing ($\forall W^{\xi}$), the best guess for the in- and output (distribution of (I, O^{ξ}), given W^{ξ}) is no different from what he would have said, had he not been involved in the computation at all (distribution of (I, O)). We make a few remarks.

- Passively secure protocols terminate almost surely, otherwise $O = \perp$ can be recognized from an infinite path W. For similar reasons, a permutation attacker can never cause a protocol with permutation security to run forever.[7]
- In our definition, permutation attackers are deterministic without loss of generality. Intuitively, if an attacker learns nothing *no matter what* ξ *she chooses*, then choosing ξ *randomly* is just a fancy way of determining in what way she is going to learn nothing.
- For similar reasons, permutation security implies passive security, since playing honestly is just a weighted mixture of "pure" permutation attacks.
- We cannot say anything if *both* players are dishonest or if they share their execution traces with one another. We also cannot guarantee that player learns nothing if the *other* player is dishonest.

Permutation Security from Passive Security

There is an important special case in which the powers of a permutation attacker turn out to be ineffective, namely if the distributions $\mathcal{F}(\mathcal{T}_p)$ never assign zero probability to a permutation.

▶ **Proposition 6.** *Let* $\mathcal{P} = (\mathcal{D}, U, Q, A)$ *be a passively secure two player protocol where for each action of form* (privatePerm, $p, \Pi, \mathcal{F}(\cdot)$) *and each permutation trace* $\mathcal{T}_p$ *of player* p*,* $\mathcal{F}(\mathcal{T}_p)$ *has support* Π[8]*. If for each attack* ξ *the attacked protocol* $\mathcal{P}^{\xi}$ *terminates with probability 1*[9]*, then* $\mathcal{P}$ *is secure against permutation attacks.*

Proof. Consider an attack ξ on $\mathcal{P}$ as player $p \in \{1, 2\}$, let $I \in U$ be any random variable denoting an input and $(I, O, \mathcal{T}_1, \mathcal{T}_2, W)$ and $(I, O^{\xi}, \mathcal{T}_1^{\xi}, \mathcal{T}_2^{\xi}, W^{\xi})$ be the execution traces of $\mathcal{P}$ and $\mathcal{P}^{\xi}$. Let w be any path in Q with $\Pr[W^{\xi} = w] > 0$ and t the permutation trace that ξ prescribes for player p along w (whenever $W^{\xi} = w$, then $\mathcal{T}_p^{\xi} = t$). For any i, o we have:

$$\begin{aligned}\Pr[(I, O^{\xi}) = (i, o) \mid W^{\xi} = w] &= \Pr[(I, O^{\xi}) = (i, o) \mid (\mathcal{T}_p^{\xi}, W^{\xi}) = (t, w)] \\ &= \Pr[(I, O\ ) = (i, o) \mid (\mathcal{T}_p\ , W\ ) = (t, w)] \\ &= \Pr[(I, O\ ) = (i, o)].\end{aligned}$$

From the first to the second line, note that firstly, since w is finite, the sequence t of choices is finite as well, so, using the assumption that $\mathsf{supp}(\mathcal{F}(\mathcal{T}_p)) = \Pi$ in all cases, there is some positive probability that an honest player behaves exactly like the attacker with respect to this finite sequence of choices. Therefore, the conditional probability in the second line is well defined. Secondly, the attacked protocol and the original protocol behave alike once we

[7] Protocols that almost surely output $\perp$ are a pathological exception.
[8] Otherwise, active attackers may pick $\pi \in \Pi$ which honest players never choose.
[9] this excludes a pathological case

fix the behavior of player p so we have the stated equality. From the second to the third line we use the passive security of $\mathcal{P}$. ◀

7 Implementing Mizuki–Shizuya Protocols

In [17], Mizuki and Shizuya's self-proclaimed goal was to define a "computational model which captures what can possibly be done with playing cards". Hence, any secure real-world procedure to compute something with playing cards can be formalized as a secure protocol in their model.[10] The other direction is not so clear. Given a secure protocol in the model, can it be implemented in the real world? We believe the answer is probably "no" (or, at least, not clearly "yes"). However, our work of identifying implementable actions in the two player model implies that a very natural subset of actions in Mizuki and Shizuya's model is implementable, even with active security: *uniform closed shuffles* (see below). Note that these shuffles already allow for securely computing any circuit [19].

Mizuki–Shizuya Protocols

We modify Mizuki and Shizuya's model slightly: instead of state machine semantics we stick to a tree of actions as in the two player model. This is an equivalent way of defining protocols, cf. [7, Sects. 3 and 4].

A *Mizuki–Shizuya protocol* is a tuple $\mathcal{P} = (\mathcal{D}, U, Q, A)$ similar to a two player protocol. The actions perm, result and turn are available as before, but instead of privatePerm actions there are shuffle actions of the form $(\mathsf{shuffle}, \Pi, \mathcal{F})$ where Π is a set of permutations and $\mathcal{F}$ is a probability distribution on Π. Executing a protocol works as before, but there are no separate permutation traces for players (there are no players at all), instead there is a single permutation trace $\mathcal{T}$. The actions perm, turn and result work as before. When an operation $(\mathsf{shuffle}, \Pi, \mathcal{F})$ is encountered, a permutation $\pi \in \Pi$ is chosen according to $\mathcal{F}$ (independent from previous choices). This permutation π is applied to the current sequence of cards without anyone learning π and appended to the permutation trace $\mathcal{T}$.

For any input $I \in U$, an execution of a protocol is described by the execution trace $(I, O, \mathcal{T}, W)$ where O is the output ($O = \perp$ if it did not terminate), $\mathcal{T}$ the permutation trace and W the path of the execution in Q. It is assumed that only W is observed, suggesting the following security notion:

▶ **Definition 7** (Security of Mizuki–Shizuya Protocols). *A Mizuki–Shizuya protocol $\mathcal{P}$ is* secure *if for each random variable $I \in U$ and resulting execution trace $(I, O, \mathcal{T}, W)$ of the protocol, (I, O) is independent from W.*

Implementing Uniform Closed Mizuki–Shizuya Protocols

We call a shuffle $(\mathsf{shuffe}, \Pi, \mathcal{F})$ *uniform* if $\mathcal{F}$ is the uniform distribution on Π, and *closed* if Π is a group. We call a Mizuki–Shizuya protocol *uniform closed* if each of its shuffle actions is uniform and closed. We are ready to state our main theorem.

▶ **Main Theorem.** *Let $\mathcal{P} = (\mathcal{D}, U, Q, A)$ be a secure uniform closed Mizuki–Shizuya protocol. Then there is a two player protocol $\hat{\mathcal{P}} = (\mathcal{D}, U, \hat{Q}, \hat{A})$ with actively secure implementation computing the same (possibly randomized) function as $\mathcal{P}$.*

[10] Excluding the use case of non-committed input protocols from [13] and [23], where the input is provided by a choice of privatePerm operations by a player, requiring input awareness/knowledge.

Moreover, the implementation of $\hat{\mathcal{P}}$ uses as helping deck only $[\![3 \cdot \spadesuit, (n-3) \cdot \diamondsuit]\!]$ for (generalized) coupled rotations and $[\![\spadesuit, (n-1) \cdot \diamondsuit]\!]$ for chosen (pile) cuts. Here, n is the length of the input sequences.

We sketch the proof here and give the formal proof in Appendix B. Each uniform closed shuffle $(\mathsf{shuffle}, \Pi, \mathcal{U})$ of $\mathcal{P}$ is replaced by two actions $(\mathsf{privatePerm}, p, \Pi, \mathcal{U})$ for $p \in \{1, 2\}$. For $\pi_2 \circ \pi_1$ to be uniformly random in Π, it suffices if π_1 *or* π_2 is chosen uniformly random in Π (while the other is known). Therefore, the joint permutation applied to the sequence after both privatePerm actions looks uniformly random to both players. Hence, they learn nothing from the execution of $\hat{\mathcal{P}}$ that they would not have also learned from executing $\mathcal{P}$. Since $\mathcal{P}$ is secure, $\hat{\mathcal{P}}$ is passively secure and by Proposition 6 also secure against permutation attacks. Moreover, by Proposition 2 all Π are implemented using the stated helping decks, so $\hat{\mathcal{P}}$ has an actively secure implementation.

8 Active Input Security

In Section 6 we have argued that results for the delegated computation setting are also applicable when players have (partial) knowledge of the input and we narrowed our focus accordingly. In this section we take a second look at protocols where players provide the input themselves. In some cases, this allows for simpler protocols with fewer cards, but it also brings about specific issues regarding active security.

We do *not* attempt a formal definition of active security in this setting, leaving this open for future work. To simplify notation, we restrict the presentation to cases with two possible inputs for each player, denoted by 0 and 1.

Warm Up: Inputs in Standard Encoding

The standard encoding of binary inputs uses the card sequence $\clubsuit\heartsuit$ to represent 0 and $\heartsuit\clubsuit$ to represent 1. When expected to provide an input in this format, a malicious player could provide marked cards or cards with altogether different symbols. Mizuki and Shizuya [18] give special attention to detecting the inputs $\clubsuit\clubsuit$ and $\heartsuit\heartsuit$ that a malicious player might provide when given several copies of otherwise uncompromised cards.

A simple solution is to place, for each player, the sequence $\clubsuit\heartsuit$ on the table and give the player the opportunity to swap the cards with no other player noticing. This is a chosen cut and has an actively secure implementation as discussed in Section 3.1, though, arguably, simpler procedures exist for this special case. After all players have provided their inputs, an ordinary protocol expecting inputs in standard format is started.

Input by Permutation

We now turn to protocols that request the inputs of the players sequentially, and by performing a permutation on the cards. In one case, we even require players to provide their input more than once.

We capture this with an additional formal action of the form $(\mathsf{inputPerm}, p, \pi_0, \pi_1)$. When it is encountered, player p should permute the current card sequence using the permutation π_0, if his input is 0, and using π_1, if his input is 1. His choice should stay hidden from the other players. Note that [9, Sect. 12.9] specifies a more general form of this action (not limited to inputs of one bit), as well as a more general form of the corresponding state diagrams (see below). Here, we chose to use a simplified version for ease of exposition.

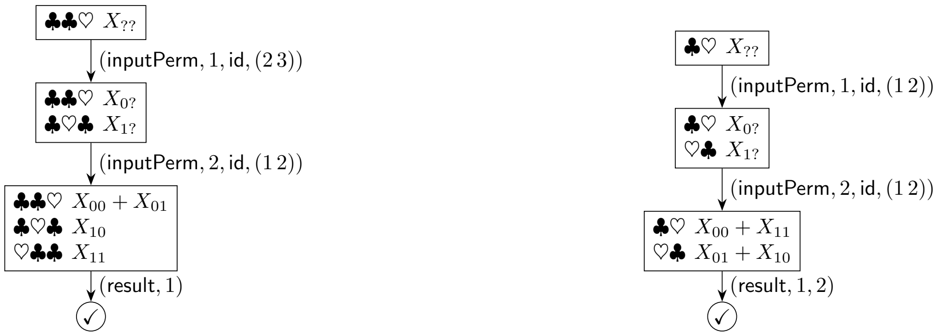


Figure 8 On the *left*, we show the state diagram of a three-card AND protocol [13, Sect. 3.2] with inputs provided by players. The result operation indicates that the first card is the output. Here, ♡ stands for 1 and ♣ for 0. On the *right* is the state diagram of a two-card XOR protocol of [22] in a similar spirit but the output is in standard format.

Considered in isolation, the action is essentially identical to $(\mathsf{privatePerm}, p, \{\pi_0, \pi_1\})$, however, its role in the surrounding protocol and its relation to security notions is fundamentally different: In the case of inputPerm, the player's choice corresponds to her input and may affect the output, while in the case of privatePerm, the choice is (in a secure protocol) independent of input and output and may be (indirectly) leaked in subsequent actions.

During a protocol execution, let the *input trace* $\mathcal{I}_p$ of a player p be the sequence of permutations performed by p during her inputPerm actions encountered so far. To simplify some diagrams in the following, we write ? for the empty input trace and 0 or 1 for non-empty input traces of players that have (so far) always chosen the permutation corresponding to the same input, i.e., have always chosen π_0 or always π_1.

Two Simple Examples: AND and XOR

We consider two very simple protocols for computing AND and XOR from [13, Sect. 3.2] and [22], respectively, shown in Figure 8. The AND protocol starts with the sequence ♣♣♡. The first player is expected to perform id or (2 3) if his input is 0 or 1, respectively. The second player acts similarly, but on the first two cards. In total, the ♡ is moved to the first position if and only if both players choose the permutation corresponding to input 1. The card in the first position therefore encodes the AND of the two player's input bits. Active security can be achieved since inputPerm actions correspond to chosen cuts. The XOR protocol is even simpler and easily generalizes to more than two players.

The shown state diagrams are adapted from [12]. Roughly speaking, each state shows which combination of input traces give rise to which card sequence. For the purposes of this section, an intuitive understanding is sufficient. For instance, in the initial state of the AND protocol in Figure 8, the card sequence is ♣♣♡ and the input traces of both players are empty, i.e. they are $() = ?$. This is represented by $X_{??}$. In the second state the input trace of player 1 could be $0 = (\mathsf{id})$ or $1 = ((2\,3))$ while the second player's input trace is still empty. The two possibilities are represented by $X_{0?}$ and $X_{1?}$ and are given next to the corresponding possibilities for the card sequence. In the last state, we see that two possibilities for the input traces may lead to the same card sequence. That card sequence is correspondingly annotated with the sum of the two possibilities.

Majority Protocols and Two Types of Attacks

The majority function with an odd number of bits as input computes the value (0 or 1) that makes up at least half of the inputs. Recently, Nakai et al. [22] proposed a protocol for computing the majority of three bits using four cards with non-standard input and output format. For comparison, note that among protocols where inputs and outputs are given in standard format, the known protocol using the fewest cards for three-input majority is [26] with eight cards. In our terminology the protocol is given as the left state diagram of Figure 9. The authors plausibly claim security in the honest-but-curious setting. It is, however, unclear how active security could be achieved due to the permutation set $\{\pi_0 = (2\,3), \pi_1 = (3\,4)\}$ in the inputPerm action of player 2. We cannot think of a simple mechanism that allows the player to perform π_0 and π_1 but prevents him from doing id or (2 4), and possibly also (2 3 4) and (2 4 3).[11]

On the right of Figure 9, we depict the state diagram in the case where player 2 can (illegally) perform id or (2 4) without being detected. In this attack scenario, player 2 can, e.g., force the result to be 0 via applying id, when both other player's inputs are 1.

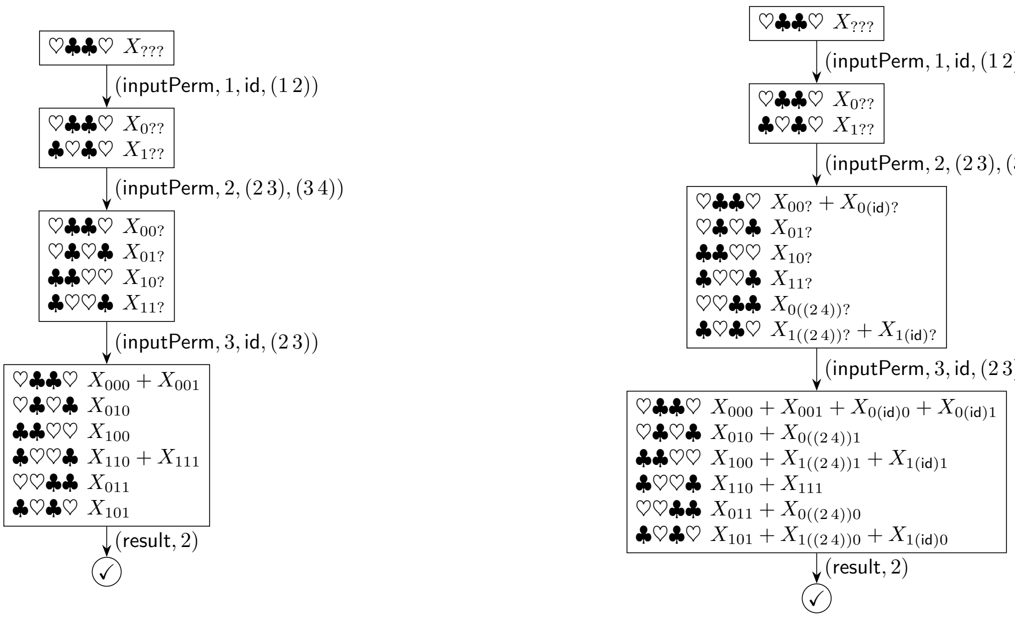


Figure 9 State diagram of the three-inputs majority protocol from [22] on the *left*. The second card encodes the result with ♡ standing for 1 and ♣ for 0. On the *right* we track the same protocol when player 2 is an active attacker who can illegally perform id or (2 4) during his inputPerm action.

In Figure 10 we give an alternative four-card majority protocol, which is conceptually very simple – similar to the AND protocol we saw before. Here each player cyclically rotates the so-far relevant cards by one for input 1 and does nothing otherwise. If the majority of the players did input 1, then the ♡ is in the first two positions. A shuffle of these two cards then conceals which one it was. The protocol has the advantage of only using inputPerm operations that are simple to implement with active security. The output is, however, encoded in an

[11] We can implement any inputPerm action if the two permutations are encoded as in [10], and a sort protocol is used to apply a chosen one of the two to the sequence of cards. For the present case this would, however, require at least 6 helping cards.

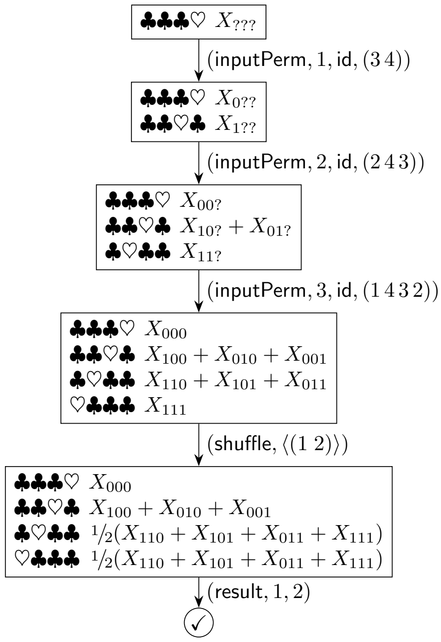


Figure 10 State diagram of a three-inputs majority protocol. The idea is that players rotate the sequence by one, if their input is 1, or do nothing otherwise. In the end, if more than one player rotated, a heart ends up in the first or second position. An additional shuffle obscures which of both is the case. The first two cards encode the output with ♡♣ and ♣♡ standing for 1, while ♣♣ stands for 0.

unusual way with ♣♣ representing 0 and ♡♣ and ♣♡ both representing 1. Note that there is a straightforward generalisation of the protocol to more than three inputs.

Finally, consider the three-card three-input majority protocol from [34] in Figure 11. All permutation sets $\{\pi_0, \pi_1\}$ of inputPerm actions can arguably be implemented with active security. However, since players 1 and 2 each have *two* inputPerm actions assigned to them, these players could choose their permutations incoherently, for instance, π_0 in the first inputPerm action and π_1 in the second. In the state diagram we have tracked this possible attack for player 1, assuming the other players are honest. The occurrence of $X_{(\mathsf{id},\mathsf{id})00}$ at the outcome ♡♣♣ indicates that an output of 0 can occur even though players 2 and 3 have both input 0 if player 1 (illegally) chooses the identity permutation in both his inputPerm actions. It seems unlikely that there is a meaningful defense against such an attack that does not substantially alter the protocol.

The case of 3-bit majority protocols shows that the question of how many cards are required does not have a straightforward answer as it depends on the desired input and output formats, the security requirements, and, if active security is desired also on whether or not helping cards are counted that might be used in the *implementation* of the inputPerm operations.

9 Conclusion

Central to our notion of active security is the concept of a *permutation set implemented with active security and choice*, indicating that a player Alice can choose to perform a permutation from the set while Bob can know that Alice did not cheat, but nothing else. We argued that

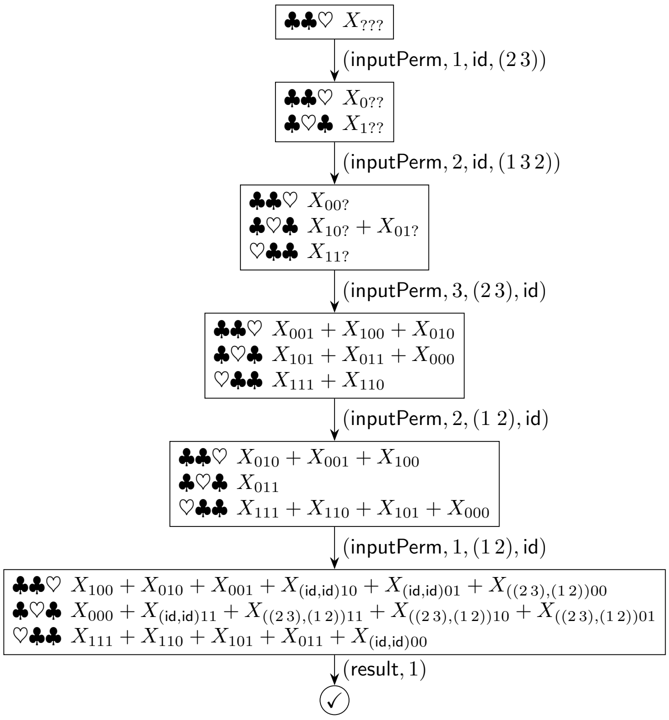


Figure 11 State diagram of the three-inputs majority protocol from [34]. We track the case that player 1 is an active attacker who may make incoherent choices during his two inputPerm actions.

cuts and *pile cuts* have such an implementation and we used *permutation protocols* to build more sophisticated procedures handling any group of permutations. Moreover, we defined security for Mizuki–Shizuya protocols, active and passive security for our own *two player protocols* and showed how secure Mizuki–Shizuya protocols using only uniform closed shuffles can be transformed into actively secure two player protocols. This is a solid foundation for actively secure card-based cryptography.

Finally, we discuss protocols where input is given by the players via choosing a permutation, including a corresponding adaptation of the state tree formalism, and present active attacks on two majority protocols from the literature.

Open Problems

Some card-minimal protocols, e.g. the general k-ary boolean function protocol of [12], use non-closed shuffles, with no evidence yet that this is necessary. As we have determined that uniform closed shuffles are a natural shuffle class, which can be done actively secure, it is interesting to find card-minimal protocols using only uniform closed shuffles.

Another natural problem is to implement more general shuffles, and even to characterize the shuffles which are possible with (a linear number of) helping cards, and the assumption of the security of a uniform random cut. To give one non-trivial example, we show in Appendix D how any subset of a cut can be implemented.

References

1 Yuta Abe, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Five-card and protocol in committed format using only practical shuffles. In Keita Emura, Jae Hong Seo, and Yohei

Watanabe, editors, *APKC@AsiaCCS 2018*, pages 3–8. ACM, 2018. doi:10.1145/3197507.3197510.

2 Eddie Cheung, Chris Hawthorne, and Patrick Lee. CS 758 project: Secure computation with playing cards. 2013. URL: https://cdchawthorne.com/writings/secure_playing_cards.pdf.

3 Claude Crépeau and Joe Kilian. Discreet solitary games. In Douglas R. Stinson, editor, *CRYPTO '93*, volume 773 of *LNCS*, pages 319–330. Springer, 1993. doi:10.1007/3-540-48329-2_27.

4 Bert den Boer. More efficient match-making and satisfiability: The five card trick. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT '89*, volume 434 of *LNCS*, pages 208–217. Springer, 1989. doi:10.1007/3-540-46885-4_23.

5 Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003. doi:10.1007/978-3-540-45146-4_27.

6 Rie Ishikawa, Eikoh Chida, and Takaaki Mizuki. Efficient card-based protocols for generating a hidden random permutation without fixed points. In Cristian S. Calude and Michael J. Dinneen, editors, *UCNC 2015*, volume 9252 of *LNCS*, pages 215–226. Springer, 2015. doi:10.1007/978-3-319-21819-9_16.

7 Julia Kastner, Alexander Koch, Stefan Walzer, Daiki Miyahara, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. The minimum number of cards in practical card-based protocols. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017*, volume 10626 of *LNCS*, pages 126–155. Springer, 2017. doi:10.1007/978-3-319-70700-6_5.

8 Alexander Koch. The landscape of optimal card-based protocols. *IACR Cryptology ePrint Archive*, 2018. Report 2018/951. URL: https://eprint.iacr.org/2018/951.

9 Alexander Koch. *Cryptographic Protocols from Physical Assumptions*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2019. doi:10.5445/IR/1000097756.

10 Alexander Koch and Stefan Walzer. Private function evaluation with cards. *IACR Cryptology ePrint Archive*, 2018. Report 2018/1113. URL: https://eprint.iacr.org/2018/1113.

11 Alexander Koch and Stefan Walzer. Foundations for Actively Secure Card-based Cryptography. In Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara, editors, *Fun with Algorithms, FUN 2021*, volume 157 of *LIPIcs*, pages 17:1–17:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FUN.2021.17.

12 Alexander Koch, Stefan Walzer, and Kevin Härtel. Card-based cryptographic protocols using a minimal number of cards. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015*, volume 9452 of *LNCS*, pages 783–807. Springer, 2015. doi:10.1007/978-3-662-48797-6_32.

13 Antonio Marcedone, Zikai Wen, and Elaine Shi. Secure dating with four or fewer cards. *IACR Cryptology ePrint Archive*, 2015. Report 2015/1031. URL: https://eprint.iacr.org/2015/1031.

14 Takaaki Mizuki. Card-based protocols for securely computing the conjunction of multiple variables. *Theoretical Computer Science*, 622:34–44, 2016. doi:10.1016/j.tcs.2016.01.039.

15 Takaaki Mizuki, Isaac Kobina Asiedu, and Hideaki Sone. Voting with a logarithmic number of cards. In Giancarlo Mauri, Alberto Dennunzio, Luca Manzoni, and Antonio E. Porreca, editors, *UCNC 2013*, volume 7956 of *LNCS*, pages 162–173. Springer, 2013. doi:10.1007/978-3-642-39074-6_16.

16 Takaaki Mizuki, Michihito Kumamoto, and Hideaki Sone. The five-card trick can be done with four cards. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 598–606. Springer, 2012. doi:10.1007/978-3-642-34961-4_36.

17 Takaaki Mizuki and Hiroki Shizuya. A formalization of card-based cryptographic protocols via abstract machine. *International Journal of Information Security*, 13(1):15–23, 2014. doi:10.1007/s10207-013-0219-4.

18 Takaaki Mizuki and Hiroki Shizuya. Practical card-based cryptography. In Alfredo Ferro, Fabrizio Luccio, and Peter Widmayer, editors, *FUN 2014*, volume 8496 of *LNCS*, pages 313–324. Springer, 2014. `doi:10.1007/978-3-319-07890-8_27`.

19 Takaaki Mizuki and Hideaki Sone. Six-card secure AND and four-card secure XOR. In Xiaotie Deng, John E. Hopcroft, and Jinyun Xue, editors, *FAW 2009*, volume 5598 of *LNCS*, pages 358–369. Springer, 2009. `doi:10.1007/978-3-642-02270-8_36`.

20 Tal Moran and Moni Naor. Polling with physical envelopes: A rigorous analysis of a human-centric protocol. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 88–108. Springer, 2006. `doi:10.1007/11761679_7`.

21 Tal Moran and Moni Naor. Basing cryptographic protocols on tamper-evident seals. *Theoretical Computer Science*, 411(10):1283–1310, 2010. `doi:10.1016/j.tcs.2009.10.023`.

22 Takeshi Nakai, Satoshi Shirouchi, Mitsugu Iwamoto, and Kazuo Ohta. Four cards are sufficient for a card-based three-input voting protocol utilizing private permutations. In Junji Shikata, editor, *ICITS 2017*, volume 10681 of *LNCS*, pages 153–165. Springer, 2017. `doi:10.1007/978-3-319-72089-0_9`.

23 Takeshi Nakai, Yuuki Tokushige, Yuto Misawa, Mitsugu Iwamoto, and Kazuo Ohta. Efficient card-based cryptographic protocols for millionaires' problem utilizing private permutations. In Sara Foresti and Giuseppe Persiano, editors, *CANS 2016*, volume 10052 of *LNCS*, pages 500–517, 2016. `doi:10.1007/978-3-319-48965-0_30`.

24 Valtteri Niemi and Ari Renvall. Secure multiparty computations without computers. *Theoretical Computer Science*, 191(1-2):173–183, 1998. `doi:10.1016/S0304-3975(97)00107-2`.

25 Takuya Nishida, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Card-based protocols for any boolean function. In Rahul Jain, Sanjay Jain, and Frank Stephan, editors, *TAMC 2015*, volume 9076 of *LNCS*, pages 110–121. Springer, 2015. `doi:10.1007/978-3-319-17142-5_11`.

26 Takuya Nishida, Takaaki Mizuki, and Hideaki Sone. Securely computing the three-input majority function with eight cards. In Adrian Horia Dediu, Carlos Martín-Vide, Bianca Truthe, and Miguel A. Vega-Rodríguez, editors, *TPNC 2013*, volume 8273 of *LNCS*, pages 193–204. Springer, 2013. `doi:10.1007/978-3-642-45008-2_16`.

27 Akihiro Nishimura, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. An implementation of non-uniform shuffle for secure multi-party computation. In *AsiaPKC 2016*, pages 49–55. ACM, 2016. `doi:10.1145/2898420.2898425`.

28 Akihiro Nishimura, Takuya Nishida, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Five-card secure computations using unequal division shuffle. In Adrian Horia Dediu, Luis Magdalena, and Carlos Martín-Vide, editors, *TPNC 2015*, volume 9477 of *LNCS*, pages 109–120. Springer, 2015. `doi:10.1007/978-3-319-26841-5_9`.

29 Akihiro Nishimura, Takuya Nishida, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. Card-based protocols using unequal division shuffles. *Soft Computing*, 22(2):361–371, 2018. `doi:10.1007/s00500-017-2858-2`.

30 Kazumasa Shinagawa, Takaaki Mizuki, Jacob C. N. Schuldt, Koji Nuida, Naoki Kanayama, Takashi Nishide, Goichiro Hanaoka, and Eiji Okamoto. Secure multi-party computation using polarizing cards. In Keisuke Tanaka and Yuji Suga, editors, *IWSEC 2015*, volume 9241 of *LNCS*, pages 281–297. Springer, 2015. `doi:10.1007/978-3-319-22425-1_17`.

31 Anton Stiglic. Computations with a deck of cards. *Theoretical Computer Science*, 259(1-2):671–678, 2001. `doi:10.1016/S0304-3975(00)00409-6`.

32 Itaru Ueda, Akihiro Nishimura, Yu-ichi Hayashi, Takaaki Mizuki, and Hideaki Sone. How to implement a random bisection cut. In Carlos Martín-Vide, Takaaki Mizuki, and Miguel A. Vega-Rodríguez, editors, *TPNC 2016*, pages 58–69. Springer, 2016. `doi:10.1007/978-3-319-49001-4_5`.

33 Tom Verhoeff. The zero-knowledge match maker. 2014. URL: `https://www.win.tue.nl/~wstomv/publications/liber-AMiCorum-arjeh-bijdrage-van-tom-verhoeff.pdf`.

34 Yohei Watanabe, Yoshihisa Kuroki, Shinnosuke Suzuki, Yuta Koga, Mitsugu Iwamoto, and Kazuo Ohta. Card-based majority voting protocols with three inputs using three cards. In

International Symposium on Information Theory and Its Applications, ISITA 2018, pages 218–222. IEEE, 2018. doi:10.23919/ISITA.2018.8664324.

A Analysis of the Protocol for Generalized Coupled Rotations

We proof that the protocol in Figure 6 implements generalized coupled rotations as stated by verifying the loop invariant from page 10 by induction.

$i = 0$. The protocol starts with all registers containing the value 0. In the first action, Alice picks $0 \leq k < t_0$ and performs $\pi = \varphi_0^k \circ \psi_{\mathsf{store}}^k$. Clearly, $\pi|_{\{1,\ldots,n\}} = \varphi_0^k$ and in $\pi(S)$ the store register contains k with both other registers containing 0. This establishes the invariant for $i = 0$, i.e. before the first execution of the loop.

$i \to i + 1$. Assume the loop invariant holds for i. In the beginning of the i+1st loop, the contents of store and temp are swapped, which leads to temp containing k, while the other registers contain 0. The permutation ψ_{copy} decrements the value of the temp register while simultaneously incrementing the value of store and main (each modulo t_0). Since the operation $(\mathsf{check}, x_0, \spadesuit)$ expects the value of temp to be 0, the only power of ψ_{copy} that will allow the check to pass is k. Assuming this happens, temp and main both contain k, while temp contains 0. Similar to before, $\widehat{\varphi}_i$ decrements main modulo t_i and since the operation $(\mathsf{check}, x_0, \spadesuit)$ expects main to contain 0, the only power of $\widehat{\varphi}_i$ that allows the check to succeed is k. Afterwards, the current iteration of the loop permuted the object cards by φ_i^k and left store containing k while the other registers contain 0. This establishes the loop invariant.

The three actions following the loop are essentially the ℓ-th iteration of the loop without the copying step so it is straightforward to verify that $\pi \in S_n$ is compatible with the protocol, iff $\pi|_{\{1,\ldots,n\}} = \varphi_\ell^k \circ \ldots \circ \varphi_0^k$ for some $0 \leq k < t_0$.

B Proof of the Main Theorem

We give the proof of our main theorem (page 16).

Proof of Main Theorem. As already mentioned, we obtain $\hat{\mathcal{P}}$ by replacing each shuffle action in $\mathcal{P}$ by two privatePerm actions. More precisely, let $(v_1, v_2, \ldots)$ be the sequence of those vertices in Q with shuffle actions. Then for each i set $A(v_i) = (\mathsf{shuffle}, \Pi_i, \mathcal{U}_i)$, where Π_i is some group and $\mathcal{U}_i$ the uniform distribution on Π_i. The tree $\hat{Q}$ is obtained from Q by replacing each v_i with two vertices $v_i^{(1)}$ and $v_i^{(2)}$ with $\hat{A}(v_i^{(p)}) = (\mathsf{privatePerm}, p, \Pi_i, \mathcal{U}_i)$, where $v_i^{(2)}$ is the child of $v_i^{(1)}$ and $p \in \{1, 2\}$. All other vertices remain unchanged.

Permutation schemes. To simplify the following argument we shall pick all relevant permutations a priori instead of producing them on-demand: A *permutation scheme* is a sequence $(\pi_1, \pi_2, \ldots)$ of permutations with $\pi_i \in \Pi_i$. We shall imagine that $\mathcal{P}$ is executed by first choosing a permutation scheme $\mathcal{T} = (\pi_1, \pi_2, \ldots)$ uniformly at random (each π_i uniformly at random from Π_i and independent of the rest)[12] and then executing the protocol as usual, except that we now use the chosen permutations, i.e. when reaching a shuffle action in vertex v_i we are determined to use π_i.

[12] Formally, $\Omega = \prod_{i \in \mathbb{N}} \Pi_i$ is a measurable space when augmented with the σ-algebra generated by $\bigcup_{i \in \mathbb{N}} \mathcal{F}_i$, with $\mathcal{F}_i := \{\Pi_1 \times \cdots \times \Pi_{i-1} \times \{\pi_i\} \times \Pi_{i+1} \times \cdots : \pi_i \in \Pi_i\}$.

Clearly, this does not affect the execution of $\mathcal{P}$ in the following sense. If I is a random input then the tuple $(I, O, W, \mathcal{T}^W)$ of input, output, execution path and permutation trace, resulting from this new way of executing $\mathcal{P}$ has the same distribution as the ordinary permutation trace of $\mathcal{P}$. By $\mathcal{T}^W$ we mean the projection of the scheme $\mathcal{T}$ to those components used in the execution, i.e. those on vertices occurring in W (in descending order).

In the same way we think of the execution of $\hat{\mathcal{P}}$, having players 1 and 2 pick permutation schemes $\mathcal{T}_1 = (\pi_1, \pi_2, \dots)$ and $\mathcal{T}_2 = (\tau_1, \tau_2, \dots)$ in advance and having player 1 use π_i if vertex $v_i^{(1)}$ is encountered and player 2 use τ_i if vertex $v_i^{(2)}$ is encountered. Then the tuple $(I, \hat{O}, \hat{W}, \mathcal{T}_1^{\hat{W}}, \mathcal{T}_2^{\hat{W}})$ of input, output, execution path and permutation traces obtained from this new way of executing $\hat{\mathcal{P}}$ has the same distribution as the ordinary execution trace of $\hat{\mathcal{P}}$. We use these modified execution traces in the following.

Computing the same function. In the following we make heavy use of the fact that $X \perp Y$ implies $f(X) \perp g(Y)$ whenever X and Y are (vectors of) random variables, f and g deterministic (measurable) functions and $\perp$ denotes independence of random variables. Here, O is determined by $(I, \mathcal{T})$, i.e. there is a deterministic measurable function f with $O = f(I, \mathcal{T})$. By construction, any permutation done by player 1 in $\hat{\mathcal{P}}$ is immediately followed by a corresponding permutation by player 2 and we see $\hat{O} = f(I, \mathcal{T}_2 \circ \mathcal{T}_1)$. Clearly, the folded permutation scheme $\mathcal{T}_2 \circ \mathcal{T}_1$ has the same distribution as $\mathcal{T}$, so $\hat{O}$ has the same distribution as O. Since we made no assumptions on I, $\mathcal{P}$ and $\hat{\mathcal{P}}$ compute the same function.

Passive Security. Note that W is determined by $(I, \mathcal{T})$, meaning there is a deterministic measurable function g with $W = g(I, \mathcal{T})$. If ext is the function acting on paths by replacing occurrences v_i with the two vertices $v_i^{(1)}$ and $v_i^{(2)}$ then we have by construction $\hat{W} = (\mathsf{ext} \circ g)(I, \mathcal{T}_2 \circ \mathcal{T}_1)$. We now see that $(I, O, W) = (I, f(I, \mathcal{T}), g(I, \mathcal{T}))$ has exactly the same distribution as

$$(I, \hat{O}, \mathsf{ext}^{-1}(\hat{W})) = (I, f(I, \mathcal{T}_2 \circ \mathcal{T}_1), g(I, \mathcal{T}_2 \circ \mathcal{T}_1)).$$

Therefore, the passive security of $\mathcal{P}$ reflected in $(I, O) \perp W$ translates into $(I, \hat{O}) \perp \mathsf{ext}^{-1}(\hat{W})$ which implies $(I, \hat{O}) \perp \hat{W}$. This is the crucial step in the following chain of reasoning:

$\mathcal{T}_p \perp \mathcal{T}_2 \circ \mathcal{T}_1$	Players choice masked by other players choice
$\Rightarrow \mathcal{T}_p \perp (I, \mathcal{T}_2 \circ \mathcal{T}_1)$	Schemes are chosen a priori, independent of I
$\Rightarrow \mathcal{T}_p \perp (I, \mathcal{T}_2 \circ \mathcal{T}_1, \hat{O}, \hat{W})$	Since $\hat{O} = f(I, \mathcal{T}_2 \circ \mathcal{T}_1)$, $\hat{W} = (\mathsf{ext} \circ g)(I, \mathcal{T}_2 \circ \mathcal{T}_1)$
$\Rightarrow \mathcal{T}_p \perp (I, \hat{O}, \hat{W})$	Projection
$\Rightarrow (\mathcal{T}_p, \hat{W}) \perp (I, \hat{O})$	Using $\hat{W} \perp (I, \hat{O})$
$\Rightarrow (\mathcal{T}_p^{\hat{W}}, \hat{W}) \perp (I, \hat{O})$	$\mathcal{T}_p^{\hat{W}}$ is a function of $\mathcal{T}_p$ and $\hat{W}$.

This also shows the corresponding independence for the ordinary execution trace, proving passive security of $\hat{\mathcal{P}}$.

Permutation Security. This follows immediately from passive security and Proposition 6.

Implementation. By Proposition 2, each group Π_i is implemented with active security and choice using the stated helping decks.

Active Security. The last two points constitute an actively secure implementation by Definition 5. ◀

C The Issue of Reusing Helping Cards

Assume we already implemented some permutation protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ for permutation sets Π_1 and Π_2 using some helping decks $\mathcal{H}_1$ and $\mathcal{H}_2$. Now we design another permutation protocol $\mathcal{P}_3$ implementing Π_3 and using its own deck of helping cards $\mathcal{H}_3$. Assume some $\mathsf{privatePerm}$ actions of $\mathcal{P}_3$ involve Π_1 and Π_2 and we intend use $\mathcal{P}_1$ and $\mathcal{P}_2$ as "subroutines". It is hence interesting to ask, what helping deck do we need for $\mathcal{P}_3$ in total.

Within $\mathcal{P}_3$ the deck $\mathcal{H}_3$ is *in use*, potentially encoding important information, so unless we make further assumptions, subroutines must treat those cards as *object cards*. If, however, the subroutines $\mathcal{P}_1$ and $\mathcal{P}_2$ are used sequentially, they may share resources. So all in all, we need $(\mathcal{H}_1 \cup \mathcal{H}_2) + \mathcal{H}_3$.

This assumes that the required helping cards from $\mathcal{H}_1$ can be re-used in $\mathcal{P}_2$ after they were used in $\mathcal{P}_1$. In particular, they need to be turned, which assumes that the arrangement of $\mathcal{H}_1$ after use does not contain sensitive information any more. This is reasonable: Not only do all of our own protocols end with the helping cards in canonical order, it would also be easy to destroy any information encoded in them by shuffling them after use, e.g. by using repeated uniform cuts.

D Implementing a Non-closed Shuffle Operation

Our focus on uniform closed shuffle operations has its reasons, but this should not distract from the fact that many other shuffle operations are both important and implementable. Let us take a special case of $(\mathsf{shuffle}, \{\mathsf{id}, (1\ 2\ 3\ 4\ 5)^3\}, \mathcal{U})$. It was for instance put to use in [2], albeit without further elaboration on its security or implementation.

Note that Π has an implementation with active security and choice (by virtue of being the subset of a cut), but performing $(\mathsf{privatePerm}, p, \Pi, \mathcal{U}(\cdot))$ for $p \in \{1, 2\}$ one after the other as we do for closed permutation sets could result in the permutation $(1\,2\,3\,4\,5)^6$. However, we can use a similar idea as we did when implementing cuts. We propose the procedure shown in Figure 12 which is a "protocol implementing a shuffle".

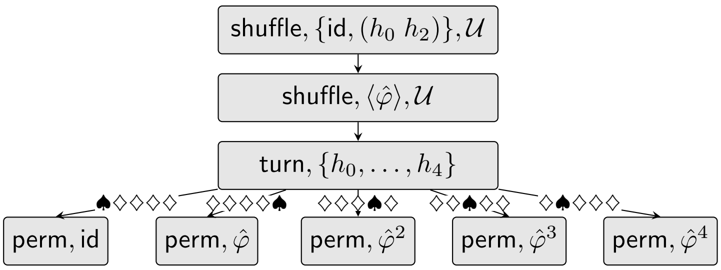


Figure 12 Protocol implementing $(\mathsf{shuffle}, \Pi = \{\mathsf{id}, (1\ 2\ 3\ 4\ 5)^3\}, \mathcal{U})$ with five helping cards (details explained in the text).

We assume that we initially have five object cards in positions 1 through 5 and a helping deck $\mathcal{H} = [\![♠, 4\cdot♢]\!]$ originally lying in positions h_0 through h_4 (say $h_i = i+6$). The ♠ starts at position h_0, but after the first shuffle operation will end up at a position h_s, where s can be 0 or 2 with equal probability. We now perform some power of $\hat{\varphi} = (1\ 2\ 3\ 4\ 5) \circ (h_0\ h_1\ h_2\ h_3\ h_4)$ which rotates both the helping sequence and the object cards by some uniformly random $0 \le k < 5$, leaving ♠ in position $h_{s+k \pmod 5}$.

The turn step reveals the helping cards and thereby $s + k \pmod 5$. Now $\hat{\varphi}^{-s-k \pmod 5}$ is performed, leaving the helping sequence in its original state and the object cards rotated by $k - s - k = -s$. Since $-s$ is with equal probability 0 or 3 $\pmod 5$ a uniformly random permutation from Π happened as desired. The only information that was revealed is $s + k$ which is independent of $-s$. Note that the two involved shuffle operations are uniform closed and may therefore be implemented as in Section 7. With this, we implement the non-closed shuffle with more basic shuffle operations.

We are confident that a clean formalization and generalization of this concept is possible and excited about future research that explores what other shuffle operations can be implemented in this sense.