Card-based Cryptography Meets Formal Verification*

Alexander Koch, Michael Schrempp, and Michael Kirsten

Karlsruhe Institute of Technology (KIT), Germany
alexander.koch@kit.edu, michi.schrempp@freenet.de, kirsten@kit.edu

Abstract. Card-based cryptography provides simple and practicable protocols for performing secure multi-party computation (MPC) with just a deck of cards. For the sake of simplicity, this is often done using cards with only two symbols, e.g., ♣ and ♡. Within this paper, we target the setting where all cards carry distinct symbols, catering for use-cases with commonly available standard decks and a weaker indistinguishability assumption. As of yet, the literature provides for only three protocols and no proofs for non-trivial lower bounds on the number of cards. As such complex proofs (handling very large combinatorial state spaces) tend to be involved and error-prone, we propose using formal verification for finding protocols and proving lower bounds. In this paper, we employ the technique of software bounded model checking (SBMC), which reduces the problem to a bounded state space, which is automatically searched exhaustively using a SAT solver as a backend.
Our contribution is twofold: (a) We identify two protocols for converting between different bit encodings with overlapping bases, and then show them to be card-minimal. This completes the picture of tight lower bounds on the number of cards with respect to runtime behavior and shuffle properties of conversion protocols. For computing AND, we show that there is no protocol with finite runtime using four cards with distinguishable symbols and fixed output encoding, and give a four-card protocol with an expected finite runtime using only random cuts. (b) We provide a general translation of proofs for lower bounds to a bounded model checking framework for automatically finding card- and length-minimal protocols and to give additional confidence in lower bounds. We apply this to validate our method and, as an example, confirm our new AND protocol to have a shortest run for protocols using this number of cards.

Keywords: secure multiparty computation · card-based cryptography · formal verification · bounded model checking · standard decks

1 Introduction

Card-based cryptographic protocols allow to perform secure multi-party computation (MPC), i.e., jointly computing a function while not revealing more

information about each individual input than absolutely necessary, with just a (regular) deck of playing cards, as long as they have indistinguishable backs. Let us start with an example. Assume that Alice and Bob meet in a bar and spend the evening together. After quite some chat, they would like to find out whether to have a second date. They are faced with the following problem: In case only one of them likes to meet again, this would cause an uncomfortable embarrassment, if he or she is the first to come out.[1] Fortunately, Alice is a notable cryptographer and likes card games, so she has with her a standard deck of cards. She remembers the protocol by Niemi and Renvall [NR99] for computing the AND function of two bits, here for outputting "yes", if both players share this mutual interest, and "no" otherwise. Doing so using an MPC protocol hides the input of the respective other player, unless it is obvious from their own input and output, hence hiding a "yes"-choice given of only one player, from the other.

In order to get a feeling for how such card-based protocols work, let us introduce the said protocol by Niemi and Renvall. It uses five cards with distinguishable symbols, which we denote – for simplicity[2] – as $\boxed{1}\,\boxed{2}\,\boxed{3}\,\boxed{4}$ and $\boxed{5}$. It is essential that the cards' backs are indistinguishable, such that when they are put face-down on the table, the only thing observable is $\boxed{}\,\boxed{}\,\boxed{}\,\boxed{}\,\boxed{}$. With these cards, the two players can encode a commitment to a bit (yes or no) by the order of two cards $\boxed{i}\,\boxed{j}$, $i, j \in \{1, \ldots, 5\}$ (with $i \neq j$) via the encoding

$$\boxed{i}\,\boxed{j} \mathrel{\hat{=}} \begin{cases} 0, & \text{if } i < j, \\ 1, & \text{if } i > j. \end{cases}$$

Alice inputs her bit by putting the cards $\boxed{1}\,\boxed{2}$ face-down and in the respective order on the table (she puts $\boxed{1}\,\boxed{2}$ for input 0, and $\boxed{2}\,\boxed{1}$ for input 1), while Bob does the same using his cards $\boxed{3}\,\boxed{4}$. We need an additional helper-card, here a $\boxed{5}$, which is put to the left of the players' cards.

The protocol starts by swapping Alice's second card with Bob's first card in the card sequence (pile) on the table. The resulting card configuration has an interesting property, namely that the order of the cards $\boxed{1}$ and $\boxed{4}$ in this sequence already encodes the output of the protocol, i.e., it reads $\boxed{4}\,\boxed{1}$ if the output is 1, and $\boxed{1}\,\boxed{4}$ otherwise. Hence, by securely removing the cards $\boxed{2}$ and $\boxed{3}$ (which is explained below), one directly obtains the output. We see this by inspecting all possible cases:

Bits	Input sequence	After swap	Removing $\boxed{2}+\boxed{3}$
$(0,0)$	$\boxed{5}\,\boxed{1}\,\boxed{2}\,\boxed{3}\,\boxed{4}$	$\boxed{5}\,\boxed{1}\,\boxed{3}\,\boxed{2}\,\boxed{4}$	$\boxed{5}\,\boxed{1}\,\boxed{\text{x}}\,\boxed{\text{x}}\,\boxed{4}$
$(0,1)$	$\boxed{5}\,\boxed{1}\,\boxed{2}\,\boxed{4}\,\boxed{3}$	$\boxed{5}\,\boxed{1}\,\boxed{4}\,\boxed{2}\,\boxed{3}$	$\boxed{5}\,\boxed{1}\,\boxed{4}\,\boxed{\text{x}}\,\boxed{\text{x}}$
$(1,0)$	$\boxed{5}\,\boxed{2}\,\boxed{1}\,\boxed{3}\,\boxed{4}$	$\boxed{5}\,\boxed{2}\,\boxed{3}\,\boxed{1}\,\boxed{4}$	$\boxed{5}\,\boxed{\text{x}}\,\boxed{\text{x}}\,\boxed{1}\,\boxed{4}$
$(1,1)$	$\boxed{5}\,\boxed{2}\,\boxed{1}\,\boxed{4}\,\boxed{3}$	$\boxed{5}\,\boxed{2}\,\boxed{4}\,\boxed{1}\,\boxed{3}$	$\boxed{5}\,\boxed{\text{x}}\,\boxed{4}\,\boxed{1}\,\boxed{\text{x}}$

[1] This is known as the "dating problem".

[2] Alice and Bob in the story might, e.g., use 7, 8, 9, 10 and a queen with any symbol.

We can remove the cards $\boxed{2}$ and $\boxed{3}$, while keeping the relative order of all cards in the sequence intact, by cutting the cards, i.e., rotating the sequence by a random offset which is unknown to the players. We can then securely turn the first card and remove it in case it is $\boxed{2}$ or $\boxed{3}$. Due to the cut, the turned card is random, and hence it does not reveal anything about the inputs. When both cards are removed, we reach a configuration where $\boxed{5}$ is the first card by the same procedure where the two remaining cards encode the AND result. Here, the $\boxed{5}$ played the crucial role of a separator that keeps the relative order of the remaining cards, starting from the separator, intact when doing a random cut. (A formal version of this protocol is given in Protocol 2 and Figure 7.)

In this paper, we are interested in whether we can do away with the helping card $\boxed{5}$, and whether there are simpler protocols. Moreover, in order to handle the increasing combinatorial state space (relative to protocols on decks of just ♣ and ♡), we introduce formal verification to the field of card-based cryptography.

1.1 Secure Multiparty Computation with Cards

In combining different protocols, one can do much more than just computing the AND function. For example, it is possible to compute arbitrary Boolean circuits by combining the well-known fact that any circuit can be expressed using only NOT and AND gates, with a method to duplicate the physically encoded bit in case of forking wires, which we make explicit by a COPY gate. In the encoding above, NOT simply inverts the order of the two cards, and a COPY-protocol is given, e.g., in [M16]. Using this setup, we can do general MPC for any function *without needing to trust a possibly corrupted computer*.

A particular advantage of protocols using physical assumptions is that they can provide a *bridge to reality*. Examples of this are given in [GBG14; FFN14], where the authors give a protocol for proving in zero-knowledge that a nuclear warhead (to be disarmed due to an international treaty) conforms to a prescribed template, without giving away anything about its internal design. In our setting of cryptography with cards, this bridge is used if the cryptographic protocol is embedded in a real card game, e.g., to prevent cheating[3]. Here, using computers is not only cumbersome, but there is no guarantee that the card sequence on my hand is the one I input into the software, hence no bridge to the physical world.

Another application of such protocols is to explain MPC in an interesting and motivating way to students in cryptography lectures. Card-based cryptography tries to find protocols for the above-mentioned AND and COPY functionalities which are card-minimal, simple and practicable. For simplicity, many protocols in card-based cryptography work with specially constructed decks, e.g., of only two symbols, ♣ and ♡. This is easy for explanation, and there are nice and easily describable protocols, such as the five-card trick by den Boer [dB89] and the six-card AND protocol by Mizuki and Sone [MS09].

[3] As an example, in a Duplicate Bridge tournament, one might prove that all sessions are handed the same cards, eliminating the need of a trusted dealer (no pun intended).

However, the setting where all cards are distinguishable, as described above, has several advantages. Firstly, we assume little about the indistinguishability of cards, which leads to stronger security guarantees. (This is more similar to the indistinguishable version of tamper-evident seals, such as scratch-off cards, by Moran and Naor [MN10].) We only need the backs (or envelopes wrapping the cards, if one wishes) to be indistinguishable. Secondly, these standard decks are more commonly available, in contrast to constructed decks. If one were to use standard decks for the protocols above, they would need multiple copies of the same card. Thirdly, considering this setting may lead to protocols using less cards than the optimal ones in the two-symbol deck setting. In fact, as our paper shows, one may use less cards than in the two-symbol deck setting. For example, our new four-card Las Vegas AND protocol presented in Section 5 uses only a very basic, practicable shuffling mechanism, namely random cuts, and uses one card less than the provably card-minimal Las Vegas AND protocol (restricted to certain types of practical shuffles) in the two-symbol deck setting. As of yet, there has only been little research in this direction, with [NR99; M16] being the only works that consider the setting where all cards have distinguishable symbols, called "standard deck" setting. Nothing is known about non-trivial lower bounds on the number of cards. This is likely due to the large state space, as there are many more distinguishable card re-orderings compared to the two-symbol case.

Within this paper, our interest is to find an automatic way of constructing compact card-based protocols which are secure and correct, based on only the two standard operations *turn* and *shuffle*, given the desired number of cards. We exploit the observation that, to the best of our knowledge, all findings in the literature employ only protocols of comparatively small length using only a small number of cards. Based on the hypothesis that we may always find some number n which is greater than or equal to any length-minimal card-protocol, we apply the automatic off-the-shelf formal program-verification technique *software bounded model checking (SBMC)* [BCC+99]. This technique allows, given such a bound n, to encode a program verification task into a decidable set of logical equations, which can then be solved by a SAT or an SMT solver. In this work, we propose an automatic method based on SBMC that, given the desired numbers of cards and protocol length, either constructs such a protocol if and only if one exists, or proves the underlying SAT formula to be unsatisfiable, i.e., shows that no such protocol exists. Based thereon, we propose that the cumbersome and error-prone task of finding such protocols or proving their non-existence by hand may be supported or complemented by such an automatic approach which is flexibly adaptable to a variety of card-based protocols and desired restrictions.

Prior to our work, it was not yet clear which role the input encoding plays when devising new protocols. This is the question on whether it can make a difference regarding the possibility of a protocol if we provide, e.g., $\boxed{1}\,\boxed{2}$ to Alice and $\boxed{3}\,\boxed{4}$ to Bob, or $\boxed{1}\,\boxed{3}$ to Alice and $\boxed{2}\,\boxed{4}$ to Bob. We provide an analysis of this question, showing that with certain restrictions, there is a relatively large freedom in choosing the input (and/or output) bases. This is a useful prerequisite in proving the impossibility of a protocol with a given number of cards.

1.2 Contribution

Our contribution consists in providing interesting new protocols and impossibility results, as well as a fully automatic method based on formal verification to support such findings. The specific advances therein are the following (cf. also Table 1 for a comparison to the literature):

(1) A four-card AND protocol in the standard deck setting, improving upon [NR99] by one card, and reaching the theoretical minimum on the number of cards. W.r.t. shuffling, this protocol only uses an expected number of 6 random cuts, compared to 7.5 random cuts in a (shortened) variant of [NR99]. Additionally, it has a natural interpretation and using only random cuts makes it particularly easy to implement in an actively secure way, cf. [KW17].

(2) We show that under certain conditions the cards for encoding input or output can be chosen freely. For one-bit output protocols and if five or more cards are available, we can freely choose both input and output bases by only extending the protocol by expected three shuffle and three turn steps. For this matter, we identify two protocols for converting a bit encoding if the new encoding shares one card with the old one.

(3) We show that there is no finite-runtime protocol for converting between bases with non-empty intersection using four cards. Moreover, there cannot be a finite-runtime AND protocol with four cards if we fix the basis in advance.

(4) We introduce formal verification to card-based cryptography by providing a technique which automatically finds new protocols using as few as possible operations and searches for lowest bounds on card-minimal protocols.

Table 1. Minimum number of cards required by AND and basis conversion protocols, subject to the running time and shuffle restrictions specified in the first two columns. Note that random cuts are a subclass of uniform closed shuffles.

Running Time	Shuffle Restr.	#Cards	Protocol	Lower Bound
AND Protocols:				
Las Vegas	random cuts	4	Theorem 3	– (trivial)
finite	–	$\geq 5^{a}, \leq 8$	[M16, Sect. 3.4]	Theorem 2
finite	uniform closed	$\geq 5^{a}, \leq 8$	[M16, Sect. 3.4]	Theorem 2
Disjoint Basis Convert Protocols:				
finite	uniform closed	4	[M16, Sect. 3.2]	– (trivial)
Overlapping Basis Convert Protocols:				
Las Vegas	random cuts	3	Theorem 4	– (trivial)
finite	–	5	Theorem 5	Theorem 1
finite	uniform closed	5	Theorem 5	Theorem 1

[a] Lower bound result only holds for fixed output basis, flexible case is still open.

1.3 Related Work

The feasibility of card-based cryptographic MPC is due to [dB89; CK93; NR98], with a formal model given by [MS14]. The only two papers looking at standard deck solutions are [NR99; M16]. Lower bounds on card-based cryptographic protocols are given by [KWH15; KKW+17; K18] for the two-symbol deck setting. The card-minimal protocol for this setting, using only practicable (i.e., uniform closed) shuffles, is given by [AHM+18] and uses five cards. The state trees used for protocols in this paper are devised by [KWH15; KKW+17].

To the best of our knowledge, this is the first work which applies formal methods to the field of card-based cryptography. However, a large range of research has been done using formal methods in the more general field of secure two-party and multiparty computations. This can be clustered into either analyzing security protocols given as high-level, abstract (and usually idealized) models, or program-based approaches targeting real(istic) protocol (software) implementations. Avalle, Pironti, and Sisto [APS14] further structure this into the two main approaches of automated model extraction and automated code generation. We refer the interested reader to overviews as given by Blanchet [B12] or Avalle, Pironti, and Sisto, and only go into a few selected works for which we identified closer links to our approach, e.g., using software bounded model checking (SBMC), SAT solvers on real(istic) protocol implementations, or relating in the analyzed security model. Standard cryptographic assumptions using lower-level computational models are – albeit more realistic – usually harder to formalize and automate. One notable line of research is CBMC-GC [FHK+14] which builds on top of the tool CBMC [CKL04]. It uses SBMC in a compiler framework translating secure computations of ANSI C programs into an optimized Boolean circuit which can subsequently be implemented securely utilizing the garbled circuit approach. Another similar setting to ours is analyzed in [RSH19], where also an "honest-but-curious" attacker model is assumed. Therein, a domain-specific language is built on top of the F* language, a full-featured, verification-oriented, effectful programming language [SHK+16]. Swamy et al. then implement MPC programs with enabled formal verification provided by the semantics of the language.

1.4 Outline

We give the computational model of card-based protocols, security definitions, etc. and the necessary preliminaries as well as a basic setup for software bounded model checking in Section 2. Section 3 discusses which freedom one has when choosing the specific cards for encoding inputs and outputs to card-based protocols and introduces a formal relabeling operation. We give lower bounds on the number of cards for AND and basis-conversion protocols in Section 4. A four-card Las Vegas AND protocol and two basis-conversion protocols are presented in Section 5 and Section 6, respectively. Section 7 gives results from applying our formal verification setup based on SBMC to our new AND protocol.

2 Preliminaries

In this section, we first formally introduce card-based protocols with their computational model (including some basic required notions), a convenient formal protocol representation, a suitable security notion, and the formal requirements for proving lower bounds. Secondly, we introduce our applied formal technique called software bounded model checking, on which, thirdly, we establish our general technique for automatically finding card- and length-minimal protocols.

2.1 Card-based Protocols

Formally, a *deck* $\mathcal{D}$ of cards is a multiset over a *(deck) alphabet* or symbol set Σ. We denote multisets by $[\![\cdot]\!]$, e.g., $[\![\heartsuit, \heartsuit, \clubsuit, \clubsuit]\!]$ is a deck over $\{\heartsuit, \clubsuit\}$. In this paper, we focus mainly on decks $\mathcal{D} = [\![1, \ldots, n]\!]$, $n \in \mathbb{N}$, where each symbol occurs exactly once. Following [M16], we call these decks *standard decks*, because decks of common card games are a good representation of such formal decks.

For encoding a bit, we additionally assume a linear order on the card symbols in Σ, which is the usual order on $\mathbb{N}$ for standard decks, and $\clubsuit < \heartsuit$ for simple two-element decks. Two face-down cards with distinct symbols $s_1, s_2 \in \Sigma$ then *encode a bit* via the following encoding rule introduced in [NR99]:

$$s_1\, s_2 \mathrel{\hat{=}} \begin{cases} 0, & \text{if } s_1 < s_2, \\ 1, & \text{if } s_1 > s_2. \end{cases}$$

Card-based protocols proceed by mainly two actions on the sequence or pile of cards: We can introduce uncertainty (about which card is which) by shuffling them in arbitrary or in certain controlled ways, e.g., by cutting the cards in quick succession, so that players do not know which card ended up where in the card sequence (or pile). Slightly more formal, a (uniform) shuffle is specified by a permutation set, from which one element is drawn uniformly at random and applied to the cards, without the players learning which one it was. Secondly, we may turn over cards and publicly learn their symbol, and act on the basis of this information. Moreover, we may deterministically permute the cards.

Permutations and Groups. Let S_n denote the *symmetric group* on $\{1, \ldots, n\}$. For elements $x_1, \ldots, x_k \in \{1, \ldots, n\}$ the *cycle* $(x_1\ x_2\ \ldots\ x_k)$ is the *cyclic* permutation π with $\pi(x_i) = x_{i+1}$ for $1 \leq i < k$, $\pi(x_k) = x_1$ and $\pi(x) = x$ for all x not occurring in the cycle. Every permutation can be written as a composition of pairwise disjoint cycles. For example, $(1\ 3\ 2)(4\ 5)$ maps $1 \mapsto 3, 3 \mapsto 2, 2 \mapsto 1, 4 \mapsto 5$, and $5 \mapsto 4$. The identity permutation is denoted as id.

Given permutations $\pi_1, \ldots, \pi_k \in S_n$, $\langle \pi_1, \ldots, \pi_k \rangle$ denotes the group generated by $\pi_1, \ldots, \pi_k$. A shuffle is a *random cut* if its permutation set is the group $\langle \pi \rangle = \{\pi^0, \ldots, \pi^{l-1}\}$ generated by a single element π *which is a cycle* $(x_1\ x_2\ \ldots\ x_l)$. A shuffle is called a *random bisection cut* if its permutation set is generated by a π which is the composition of pairwise disjoint cycles of length 2. Finally, an S_k*-shuffle* is a shuffle with permutation set S_k.

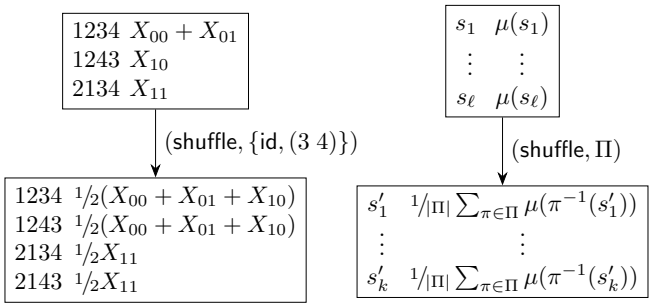


Fig. 1. A shuffle operation, given by example (left), and via the general rule (right).

Computational Model and Protocol Tree Representation. For our formal descriptions, we make heavy use of the KWH trees introduced in [KWH15] and shown to be equivalent to the computational model of [MS14; MS17] in [KKW+17]. We start by the start node

$$\begin{array}{|ll|}\hline 12\,34 & X_{00} \\ 12\,43 & X_{01} \\ 21\,34 & X_{10} \\ 21\,43 & X_{11} \\ \hline\end{array}$$

and add eventually needed further cards ($\boxed{5}$, $\boxed{6}$, ...) to the right of the players bits. The state (or KWH) tree is directed, with annotations at the outgoing edges of the state, specifying the action that is performed next. Let μ be the state with the outgoing annotation, then the actions are defined as:

1. $(\mathsf{shuffle}, \Pi)$ leads to a μ' as in Figure 1, where $\Pi \subseteq S_{|\mathcal{D}|}$ is a permutation set.
2. (turn, T) branches the tree into states μ_v for each observation v possible by revealing the cards at positions from the set $T \subseteq \{1, \ldots, |\mathcal{D}|\}$, as in Figure 2. μ_v contains the sequences from μ which are compatible with the observation v. For each sequence s compatible with v, we have $\mu_v(s) := \mu(s)/\Pr[v]$, where $\Pr[v] \in (0, 1]$ is the probability of observing v.
3. (perm, π) permutes the sequences of μ according to π.
4. $(\mathsf{result}, p_1, p_2)$ stops the computation and returns the cards at p_1, p_2 as output.

We start by a state that encodes the input sequences attached to their respective symbolic input probabilities, see [KKW+17] for a thorough explanation:

$$\begin{array}{|ll|}\hline 12\,34 & X_{00} \\ 12\,43 & X_{01} \\ 21\,34 & X_{10} \\ 21\,43 & X_{11} \\ \hline\end{array}$$

A protocol computes a Boolean function $f\colon \{0,1\}^2 \to \{0,1\}$ if the start state (tree root) encodes each $b \in \{0,1\}^2$ in the first four cards (the remaining cards being at fixed positions), and in the leaf nodes of the protocol's state tree, it holds

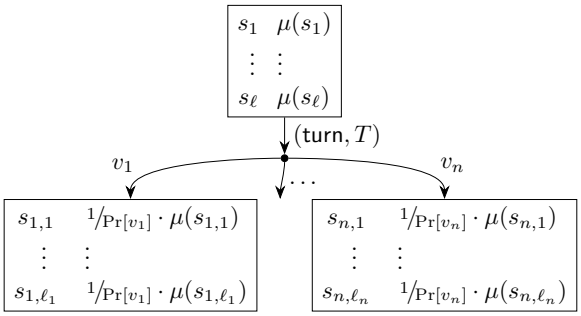


Fig. 2. A turn operation. Here, $v_1, \ldots, v_n$, are the possible observation by turning the cards at positions in T. For each $i \in \{1, \ldots, n\}$ the $s_{i,1}, \ldots, s_{i,\ell_i}$ are the sequences from $s_1, \ldots, s_\ell$ which are compatible with v_i. Note that in secure protocols, the probability of observing v_i, denoted as $\Pr[v_i]$, is constant.

for the positions given by the result operation that the cards at these positions encode a value $o \in \{0, 1\}$ if all X_i occurring in $\mu(s)$ for sequence s satisfy $f(i) = o$ *(Correctness)*. We say that a protocol has *finite runtime* if its tree is finite. It is a *Las Vegas* protocol, if it is not finite runtime, but the expected length of any path in its tree is finite. Note that while we consider looping protocols, we do not consider the case where a complete restart is necessary. For self-similar infinite trees, we simplify by drawing edges to earlier states.

Security of Card-based Protocols. We slightly adjust the security notion from the literature to standard decks. For more details, we refer to [K19]. Since different encodings for the same bit are possible, we want the encoding basis of the output bit to not give away anything about the inputs. We say that a protocol is *secure* if at any turn operation the probability for each observation v is a constant $\rho \in [0, 1]$ (using $\sum_{i \in \{0,1\}^2} X_i = 1$), *and additionally* if at any result operation the probability of each output basis is constant in the same sense.

As in [KKW+17], for our impossibility proofs and formalizations with bounded model checkers, it is useful to consider a weaker form of security, which is a necessary criterion for security as defined above: A protocol is *possibilistically output-secure*, if at any state of the protocol, every output can still be possible. This weakens the normal security guarantee, as the probability for a given input sequence could be higher in this state. One could even be able to exclude a specific input sequence, if the corresponding output can still be possible through another input sequence. Together with possibilistic input-security, this discussion leads to the following formal definition:

Definition 1 (cf. [KKW+17]). *A protocol $\mathcal{P} = (\mathcal{D}, U, Q, A)$ computing a function $f\colon \{0,1\}^2 \to \{0,1\}$ has* possibilistic input security (possibilistic output security) *if it is correct, i.e., output $O = f(I)$ almost surely and for uniformly*[4]

[4] Actually, the distribution does not matter, as long as $\Pr[I = i] > 0$ for all $i \in \{0,1\}^k$.

random input I and any visible sequence trace v with $\Pr[v] > 0$ *as well as any input* $i \in \{0,1\}^2$ *(any output* $o \in \{0,1\}$*) we have* $\Pr[v \,|\, I = i] > 0$ $(\Pr[v \,|\, f(I) = o])$.

Proving Lower Bounds. We call two states, μ and μ', similar, if μ is equal to μ' up to row or column permutation. This is an interesting equivalence relation for reducing the state space and we make use of it in our impossibility results.

As in [KKW+17, Definition 3], we define reduced states, where states are not annotated by their symbolic probabilities, but by the result that is specified by their inputs. This simplifies impossibility proofs by reducing information and the state space. Any such reduced tree captures only a weak form of security, possibilistic security, as discussed above where each output (reachable in principle) needs to be still possible. Showing that a protocol is impossible even in this weak setting implies its general impossibility.

To obtain a reduced state tree, we project all the symbolic probabilities of the sequences in a state tree to a *type* (representing the possible future output associated with the sequence in a correct protocol, see below), which can be any $o \in \{0,1\}$. For this, let $\mathcal{P}$ be a protocol computing a function $f\colon \{0,1\}^2 \to \{0,1\}$ and μ be a state in the state tree. For any sequence s with $\mu(s)$ being a polynomial with positive coefficients for the variables $X_{b_1}, \ldots, X_{b_i}$ $(i \geq 1)$, set $\hat{\mu}(s) \coloneqq o \in \{0,1\}$ if $o = f(b_1) = f(b_2) = \ldots = f(b_i)$ in the resulting *reduced state* $\hat{\mu}$. We call sequences in $\hat{\mu}$ according to their type *o-sequences*.

For proving impossibility results, we make use of the backwards calculus as given in [K18]. We highlight the main ideas here, but refer to it for reference. Denote by $\mathsf{shuf}^{-1}(\mathcal{G})$, for a set of states $\mathcal{G}$, the *set of states that are transformed into a state in* $\mathcal{G}$ *by a shuffle*. The trivial shuffle is allowed, i.e., $\mathcal{G} \subseteq \mathsf{shuf}^{-1}(\mathcal{G})$. Moreover, $\mathsf{turn}_{\mathrm{f}}^{-1}(\mathcal{G})$ is the set of states being in $\mathcal{G}$ or having a turnable position i such that all immediate successor states from a turn at i are in $\mathcal{G}$. Define by $\mathsf{cl}_{\mathrm{f}}(\mathcal{G})$ *the closure* of $\mathsf{turn}_{\mathrm{f}}^{-1}(\cdot)$ and $\mathsf{shuf}^{-1}(\cdot)$ operations on $\mathcal{G}$. Hence, it holds that if the start state is not in $\mathsf{cl}_{\mathrm{f}}(\mathcal{G})$, then no finite-runtime protocol can exist.

2.2 Automatic Formal Verification Using SBMC

In the following, we introduce an automatic technique from formal program verification, namely software bounded model checking (SBMC), to the field of card-based cryptography. We first describe the general technique of using SBMC to check for software properties, before we explain how we apply it to search for cryptographically secure card-based protocols. In a nutshell, we translate the task to a reachability problem in software programs (which will later-on be a program encoding operations on an abstract state tree as described above), which the SBMC tool encodes into an instance of the SAT problem.

We assume we are given an imperatively defined function f under the form of an imperative program (for example, written in the C language), that uses some parameter values taken among a set of possible start values I. An entry $i \in I$ is a list of values, one value for each such parameter: it gives a value to everything that a run of f depends on, such as its input variables, or anything that is considered

non-deterministic (i.e., of arbitrary, but fixed, value for any concrete evaluation of f) from the point of view of f. For this reason, those parameters are qualified as "non-deterministic", to distinguish them from normal parameters used in a programming language to pass information around. Moreover, some values can be "derived", thus, computed in f from the non-deterministic parameter values, or declared as constants in f, and both values of non-deterministic parameters or derived values can then be used as normal parameters in the program. We are also given a software property to be checked about f, in the form $C^{\text{ant}} \Rightarrow C^{\text{cons}}$, where *ant* and *cons* stand for antecedent and consequence respectively. Both C^{ant} and C^{cons} are sets of Boolean statements. A Boolean statement is a statement of f that evaluates to a Boolean value, for example, a simple statement checking that some computed intermediate value is positive. An entry i is said to satisfy a set of Boolean statements if and only if all Boolean statements in the set evaluate to true during the execution of f using the non-deterministic parameter values i, and is said to fail the set of Boolean statements otherwise. The property $C^{\text{ant}} \Rightarrow C^{\text{cons}}$ requires that for all possible entries $i \in I$, *if* i satisfies C^{ant}, *then* i satisfies C^{cons}. As an example, assume f computes, given i, two intermediate integer values v_1 and v_2, and then returns a third value v_3. The property to be checked could, e.g., be: *if* v_1 is negative, *then* v_2 is positive and v_3 is odd. A solver that is asked to check a software property $C^{\text{ant}} \Rightarrow C^{\text{cons}}$ thus exhaustively searches for an entry i that satisfies C^{ant} but fails C^{cons}. The property is valid if and only if there does not exist any such entry i, i.e., it is impossible to find.

SBMC is a fully-automatic static program analysis technique used to verify whether such a software property is valid, given a function and a property to be checked. It covers all possible inputs within a specified bound. It is static in the sense that programs are analyzed without executing them on concrete values or considering any side channels. Instead, programs are symbolically executed and exhaustively checked for errors up to a certain bound, restricting the number of loop iterations to limit runs through the program to a bounded length. This is done by unrolling the control flow graph of the program and translating it into a formula in a decidable logic that is satisfiable if and only if a program run exists which satisfies C^{ant} and fails C^{cons}. The variables in the formula are the non-deterministic parameters of f, and their possible values are taken from I.

This reduces the problem to a decidable satisfiability problem. Modern SAT-solving technology can then be used to verify whether such a program run exists, in which case an erroneous input has been found, and the run is presented to the user. If the solver cannot find such a program run, it may be either because the property is valid, or because it is invalid only for some run which exceeds the bound. In some cases, SBMC is also able to infer statically which bound is sufficient to bring a definitive conclusion.

2.3 Automatic Formal Verification for Card-based Protocols

Our approach employs a standardized program representation of the KWH trees introduced in [KWH15] (and described in the beginning of this section). This allows a general programmatic encoding of both shuffle and turn operations, as

well as of the fixed input state (indicated by the input card sequences from the table in the very beginning of this paper), the non-deterministic reachable states, and the logical function to be computed securely.

The input state is trivially derived from the specified numbers of cards as the size and order of the players' commitments is fixed and the (without loss of generality) consecutively ordered card sequence of (distinguishable) helper-cards is simply prepended to the input card sequence, annotated with their respective input probabilities. Any input state thus consists of exactly four distinguishable card sequences. Based on this input state, the program performs a loop, which successively performs turn or shuffle operations based on the input state and computes the resulting states from which it continues performing turn or shuffle operations. The loop ends when the specified bound (representing the length of the protocol to be found) is reached, checks whether the final state is indeed a valid computation of the secure function, and (if and only if the check is successful) the found protocol is then presented to the user.

However, this task involves multiple computational complexities, most notably both the number of (possibly) reachable states, and the choice of the next operation, i.e., either choosing the card(s) to be turned or which shuffle to perform. We partially overcome the first computational complexity by not considering Las Vegas protocols as this relieves us from checking every reachable sequence of states to be finite. In fact, we compute all reachable states after every protocol operation, but only check each of them to be valid, and then proceed our operations on only one of them, which is non-deterministically chosen among them. The second computational complexity consists in first non-deterministically choosing whether to shuffle or to turn, and then to perform the respective operation. The turn operation is less interesting as it is mostly the obvious implementation for updating the computed state and its probabilities using mostly standard imperative program operations, except that the turn observations are again non-deterministically chosen, hence making the SBMC tool consider any of them to be possible. The more interesting operation is the shuffle operation, as it must randomly draw a set of permutations on which the thereby reachable states are computed. We implement this by non-deterministically choosing a set of permutations from a precomputed set of all generally possible permutations. Both the amount[5] and the choices of the respective permutations are chosen non-deterministically. Moreover, we restrict our experiments to only closed shuffles and proceed by restricting the computed set of permutations to be either closed or of size one (i.e., a simple permutation).

Finally, after iterating the afore-mentioned loop for the specified bound number with the described operations and restricting that final state indeed computes the secure function, we specify the software property C^{cons} to be checked simply as the Boolean value `false`. This trivially unsatisfiable property implies that the verification task always fails once there exist input and non-deterministic parameters such that the respective program run reaches the statement in the

[5] In order to keep the execution times still manageable for our experiments, we bound this amount by the (arguably quite reasonable) number 8.

program which checks this property. The SBMC tool exhaustively searches for a run of the specified length through the program which leads from the starting state to a correct and secure state which satisfies the given security notion, i.e., reaches the above-metioned statement. Hence, if there exists any protocol of the specified length which computes the secure function and for which the specified operations and valid intermediate states (representing KWH-trees) exist, such a protocol is presented by our method. If no such protocol can be found, we know there is no card-based protocol of the specified length satisfying all our restrictions on permitted turn and shuffle operations, as well as intermediate and final states. This means there exists no model for the SAT formula which encodes the set of all permitted program runs given our specified requirements.

Hence, assuming our translation of KWH trees and respective protocol operations into a simple imperative program are correct, this method can then be used in an iterative manner to strengthen the bounds from the literature. Note that this is largely based on the so-called "small-scope hypothesis", i.e., a large number of bugs are already exposed for small program runs. We apply this hypothesis to the setting of card-based security protocols as all protocols in the literature only use a small number of turn and shuffle operations and the length of any found protocol is below ten operations.

This approach can be generalized to search for card-based protocols using a pre-defined number of actions and adhering to a given formal security notion. We have written a general program[6] to search for such situations parameterized in the desired restrictions on actions and security notions. Note that, in order to cope with the still considerable state space size, we use the refined security notion of output-possibilistic security.

3 On the Choice of Cards for Input and Output

We essentially show that the choice of input basis (or output basis, but not necessarily both) is irrelevant for the functioning of the protocol. In rare cases, one has to append two operations to existing protocols to make them fully basis flexible. In the Niemi–Renvall protocol shown above, the protocol description specifies Alice's cards to be of symbols $1, 2$, and Bob's to be of symbols $3, 4$ and the helping card to be a 5. To simplify later proofs and to demonstrate an interesting symmetry in card-based protocols, we show that this choice is irrelevant for the functioning of the protocol.

For this, we define a *relabeling* from deck alphabet Σ to a deck alphabet Σ', i.e., a bijective function $\lambda\colon \Sigma \to \Sigma'$.[7] A relabeling of a sequence $s = (s_1, \ldots, s_n)$ is a relabeling of each of its symbols, i.e., $\lambda(s) := (\lambda(s_1), \ldots, \lambda(s_n))$. A relabeling of a state is given by the relabeling of all its sequences, a relabeling of a protocol/state (sub)tree is the relabeling of all its states as described by Figures 3 and 5.

[6] The source code is available under https://github.com/mi-ki/cardCryptoVerification.

[7] In case of the decks being a subset of $\mathbb{N}$, we may use usual permutation notation. We require that if λ maps x to y, that the cardinalities of x and y are equal in the deck.

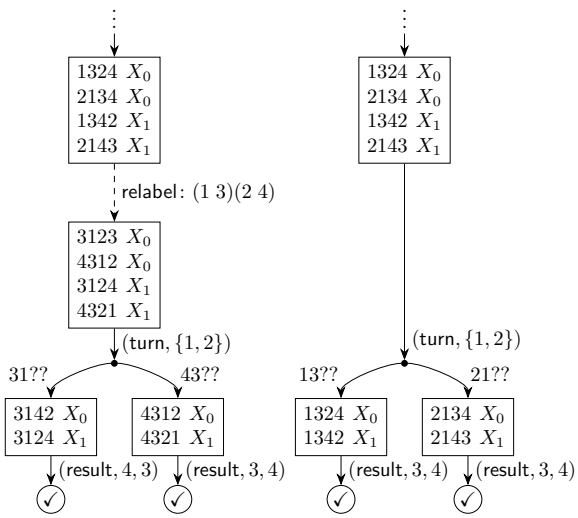


Fig. 3. Example of the relabel action, swapping the card symbols of 1 and 3, and of 2 and 4, respectively. This action is for abbreviated writing only, it does not actually relabel the physical cards, which seems impossible without learning their symbols. Hence, the tree on the left is virtually translated to the right. Note that the relabeling only affects the sequences, the observations at edges belonging to turn actions and may swap the order of the indices in result operations.

Lemma 1. *If $\mathcal{P}$ is a protocol with deterministic output basis, one can relabel the cards without affecting the functioning.*

Note that the deterministic output basis restriction is important, because if we have a randomized output encoding such as in Figure 4 on the left, a relabeling might affect the monotonicity of the encoding of only one of the possible output bases. In this case, we make use of the following lemma, as illustrated Figure 4.

Lemma 2. *Every protocol with one-bit output and a randomized output basis can be transformed into a protocol with deterministic output basis, by inserting a shuffle and a turn before any result operation with randomized output basis.*

4 Impossibility of Finite-Runtime Four-Card AND and Basis Conversion with Overlapping Bases

In this section we give our main impossibility results.

Theorem 1. *There is no four-card finite-runtime basis conversion protocol for overlapping bases with deck $\mathcal{D} = [\![1, 2, 3, 4]\!]$.*

Proof. We proceed by using the backwards calculus technique from [K18], as described in Section 2.1. That is, we show that if we start with the set of (highly-structured) final states $\mathcal{G}_0$ of basis conversion protocols and enlarge this set

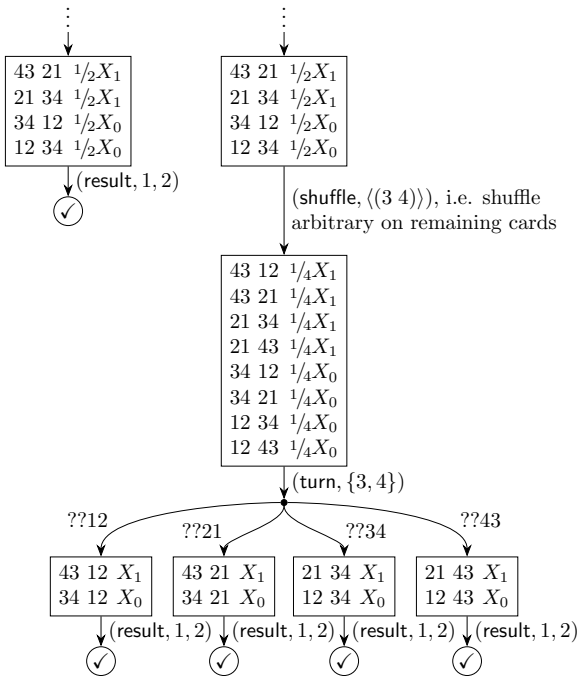


Fig. 4. Example of making the basis deterministic, cf. Lemma 2. On the left you see a tree part with one-bit output and randomized basis, i.e., the output basis may be $\{1,2\}$ or $\{3,4\}$, each with a probability of $1/2$. We can make it known to the players, i.e., deterministic, by splitting up the state via an S_k-shuffle (here: $k=2$) on the remaining cards (so that they no longer contain any information), turning these and then doing the result operation. By what is visible in the turn, one can derive the output basis.

iteratively by states which reach the given states by a shuffle or a turn, we obtain the closure $\mathsf{cl}_{\mathrm{f}}(\mathcal{G}_0)$. If we consider only reduced states, the set of possible states is finite, so applying $\mathsf{turn}_{\mathrm{f}}^{-1}(\cdot)$ and $\mathsf{shuf}^{-1}(\cdot)$ operations to the growing set of states, starting from $\mathcal{G}_0$, will become stationary. It remains to show that the start state is not contained in the closure. We assume w.l.o.g. the input basis $\{1,2\}$ with helping cards 3 and 4, and the output basis $\{o_1 < o_2\}$ such that $|\{1,2\} \cap \{o_1, o_2\}| = 1$. For simplicity, we want the output basis $\{1,3\}$ and argue later why this choice did not affect the proof statement. Hence, the final state is any choice of at least one 1-sequence and one 0-sequence of the state on the left:

13 24	0
13 42	0
31 24	1
31 42	1

12 34	0
21 34	1

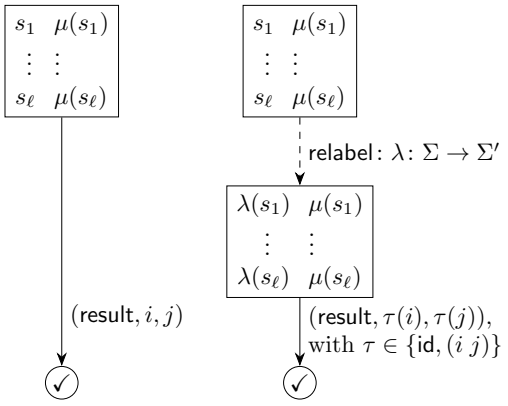


Fig. 5. The formal rule for relabeling leaf nodes of one-bit output protocols. Let $r_1 = s_k[i], r_2 = s_k[j] \in \mathcal{D}$ be the output symbols (before relabeling) of some arbitrary sequence s_k of μ. Then, $\tau = \mathsf{id}$, if $r_1 < r_2$ implies $\lambda(r_1) < \lambda(r_2)$ (λ is monotone on r_1, r_2) and $\tau = (i\ j)$ otherwise.

The state on the right is the start state of a basis-conversion protocol. Both states are considered up to similarity.

We have $\mathsf{shuf}^{-1}(\mathcal{G}_0) = \mathcal{G}_0$, i.e., shuffling steps do not help in the last step of a output-possibilistically secure protocol, because any subset of a final state which contains at least one 1-sequence and one 0-sequence (required as 1-/0-sequences cannot be generated out of thin air by a shuffle), is already final. Hence, we consider $\mathcal{G}_1 := \mathsf{turn}_\mathrm{f}^{-1}(\mathcal{G}_0)$, i.e., the states turnable at a position i, where all immediate child nodes when turning at i are in $\mathcal{G}_0$. W.l.o.g. we assume the turn to be at position 4. By [K18, Lemma 3], we use that $\mathcal{G}_1 = \mathsf{turn}_\mathrm{f}^{-1}(\mathcal{G}_0) = \mathcal{G}_0 \cup \mathsf{turn}_\mathrm{f}^{-1}(\mathsf{cc}(\mathcal{G}_0))$, where $\mathsf{cc}(\mathcal{G}_0)$ is the subset of $\mathcal{G}_0$ with states that have a constant column:

13 24	0
31 24	1

13 42	0
31 42	1

However, we aim to enlarge this set (which we can do since our claim is only made stronger by monotonicity of the backwards operations) by the states

24 13	0
42 13	1

24 31	0
42 31	1

,

because they would be reachable anyway via a disjoint basis conversion due to [M16, Sect. 3.2]. The states from $\mathcal{G}_1 \setminus \mathcal{G}_0$ look as follows:

$\ldots a$	0
$\ldots a$	1
$\ldots b$	0
$\ldots b$	1
$\ldots c$	0
$\ldots c$	1
$\ldots d$	0
$\ldots d$	1

where at least two of the blocks are present, and $a, b, c, d \in \mathcal{D}$ are pairwise distinct. Note that the start state cannot be of this form, as it contains only two sequences. To show that another backwards turn step does not enlarge the set by showing that $\mathsf{cc}(\mathcal{G}_1) = \mathsf{cc}(\mathcal{G}_0)$. For this, note that the states from $\mathsf{cc}(\mathcal{G}_0)$ have two constant columns, but with the specific pairing that if one is 1, the other is 3 and vice versa, or if one is 2, the other is 4 and vice versa. Hence, having another constant column in the state from $\mathcal{G}_1 \setminus \mathcal{G}_0$ above, say at position 3, would need the same symbol (given by the pairing) in the fourth column. Hence, it can only have two sequences, i.e., it is already in $\mathcal{G}_0$. This shows that $\mathsf{turn}_\mathrm{f}^{-1}(\mathcal{G}_1) = \mathcal{G}_1$.

Now, for the main step of the proof, set $\mathcal{G}_2 := \mathsf{shuf}^{-1}(\mathcal{G}_1)$ and $\mathcal{G}_3 := \mathsf{turn}_\mathrm{f}^{-1}(\mathcal{G}_2)$. Because the shuffling is unrestricted, applying another backwards shuffle to $\mathcal{G}_2$ cannot give a larger set, as we can always combine two shuffles into one. The remaining proof will show that $\mathcal{G}_3 = \mathcal{G}_2$ in which case no further enlargement is possible. Afterwards, showing that the start state is not in $\mathcal{G}_2$ finishes the proof.

As $\mathcal{G}_2$'s states are subsets of $\mathcal{G}_1$'s states, $\mathsf{cc}(\mathcal{G}_2)$'s general form is on the left:

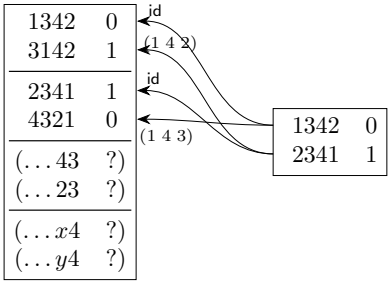
$\ldots da$	0
$\ldots da$	1
$\ldots db$	?
$\ldots dc$	?

$\ldots da$	0
$\ldots da$	1
$\ldots db$	?
$(\ldots ab$	?)
$\ldots dc$	?
$(\ldots ac$	?)
$(\ldots xd$	?)
$(\ldots yd$	?)

where ? can be either 0 or 1 and x, y are either both a, or one is b and the other c. To see this, observe that it is a subset of the state on the right where we leave out at least all sequences interfering with our wish of a constant column in this position (in parentheses on the right). Our aim is to show that these states are more specifically the states of $\mathsf{cc}(\mathcal{G}_0)$ again, i.e., it is impossible to reach any state of form in $\mathcal{G}_1$ via a shuffle from these states. Due to the complexity of the situation, we do a case distinction on the number of sequences of $\mu \in \mathsf{cc}(\mathcal{G}_2)$.

Let us consider only the first case, the other cases are analogously and are to be found in a full version of the paper. Let μ contain *two sequences*. If they

were both from the first block, the state would trivially be in $\mathsf{cc}(\mathcal{G}_0)$. This leaves us with two choices, either include a sequence ending with da or exclude it. For concreteness, we choose w.l.o.g. $a = 2$, $d = 4$, $b = 1$ and $c = 3$, and have this:



Reaching this state on the left by a shuffle should contain at least $\{\mathsf{id}, (1\ 4\ 3), (1\ 4\ 2)\}$. But if we apply $(1\ 4\ 2)$ to the first sequence gives a sequence 3241 which is not possible on the left side due to its trailing 1. The other cases are similar.

Theorem 2. *There is no four-card finite-runtime AND protocol with deck* $\mathcal{D} = [\![1, 2, 3, 4]\!]$ *with fixed-in-advance output basis.*

Proof. If the output basis is not given using only Alice's or only Bob's cards, this follows from Theorem 1, because if there would be such an AND protocol, by fixing the second bit to 1 one could easily generate a basis-convert protocol, which is impossible. In the remaining case, e.g., of the output basis being Alice's cards, say 1, 2, this would not be a basis-convert, as the bit remains unchanged. In this case, a close analysis of the proof of Theorem 1 above yields that the theorem also holds in this case. We omit the details, and refer to the full version.

5 Card-Minimal Protocols for AND

Theorem 3. *There is a four-card Las Vegas AND protocol with deck* $\mathcal{D} = [\![1, 2, 3, 4]\!]$ *using only random cuts.*

Proof. See Figure 6 and Protocol 1.

To get a better understanding of why the protocol works and how it is related to the protocol of [NR99], let us consider exemplarily the case that the first card to be revealed is a 1, the other cases are analogous. In this situation, let us look at the different cases, given in Table 2. Using the method as before, we can remove $\boxed{3}$ by performing a random cut while leaving the relative order intact ($\boxed{1}$ here is assigned the role of the $\boxed{5}$ in Niemi and Renvall's protocol) and waiting until it appears when turning. Later we can remove the $\boxed{1}$ from the remaining cards, to get the output encoded using the cards $\boxed{2}$ and $\boxed{4}$. A closer analysis of the situation after removing $\boxed{3}$ shows that one can take a shortcut when one is not bound to the output being cards $\boxed{2}$ $\boxed{4}$ (which is not our goal, because in the

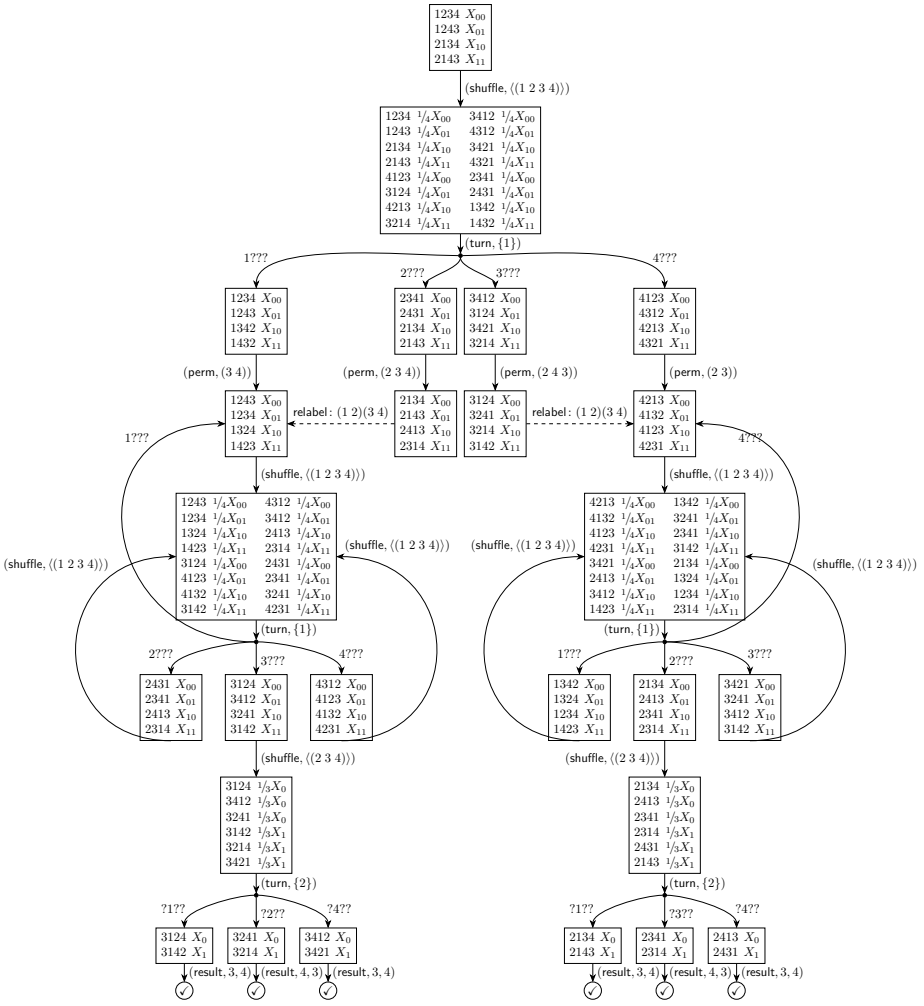


Fig. 6. Four-card Las Vegas AND protocol using random cuts, cf. Protocol 1. Here, $X_0 \coloneqq X_{00} + X_{01} + X_{10}$ and $X_1 \coloneqq X_{11}$. The relabel operations are not actual actions to be performed but help abbreviate the write-up of the protocol, see Section 3.

Table 2. The different states of Protocol 1 after $\boxed{1}$ was revealed in the first turn. The permutation to be applied in this case is (3 4). The situation is similar in all other cases.

Bits	Sequence	After permutation	Removing $\boxed{3}$
$(0,0)$	$\boxed{1}\,\boxed{2}\,\boxed{3}\,\boxed{4}$	$\boxed{1}\,\boxed{2}\,\boxed{4}\,\boxed{3}$	$\boxed{1}\,\boxed{2}\,\boxed{4}\,\boxed{\text{x}}$
$(0,1)$	$\boxed{1}\,\boxed{2}\,\boxed{4}\,\boxed{3}$	$\boxed{1}\,\boxed{2}\,\boxed{3}\,\boxed{4}$	$\boxed{1}\,\boxed{2}\,\boxed{\text{x}}\,\boxed{4}$
$(1,0)$	$\boxed{1}\,\boxed{3}\,\boxed{4}\,\boxed{2}$	$\boxed{1}\,\boxed{3}\,\boxed{2}\,\boxed{4}$	$\boxed{1}\,\boxed{\text{x}}\,\boxed{2}\,\boxed{4}$
$(1,1)$	$\boxed{1}\,\boxed{4}\,\boxed{3}\,\boxed{2}$	$\boxed{1}\,\boxed{4}\,\boxed{2}\,\boxed{3}$	$\boxed{1}\,\boxed{4}\,\boxed{2}\,\boxed{\text{x}}$

other cases besides the first turn being 1 it is different anyway, and one would have to add conversion protocols to ensure this). The situation is as follows: The remaining three cards are either a cyclic rotation (cut) of the sequence $\boxed{1}\,\boxed{2}\,\boxed{4}$, if the output is 0, or a cyclic rotation of the sequence $\boxed{1}\,\boxed{4}\,\boxed{2}$, otherwise. A cut cannot rotate a sequence of the former type to become the other, or vice versa. After the cut we can safely turn any card and, from the resulting symbol, deduce in which order the other cards must be output to encode the protocol result.

Protocol 1. Our four-card AND protocol. The first bit is in basis $\{1,2\}$, the second in $\{3,4\}$, and the output in $\{1,2,3,4\}\setminus\{v_2,v_3\}$, where v_2, v_3 are the last two revealed symbols. See Figure 6 for a KWH tree representation.

$(\mathsf{shuffle}, \langle(1\ 2\ 3\ 4)\rangle)$
$v_1 := (\mathsf{turn}, \{1\})$
if $v_1 = 1$ **then** $(\mathsf{perm}, (3\ 4))$
else if $v_1 = 2$ **then** $(\mathsf{perm}, (2\ 3\ 4))$
else if $v_1 = 3$ **then** $(\mathsf{perm}, (2\ 4\ 3))$
else if $v_1 = 4$ **then** $(\mathsf{perm}, (2\ 3))$

Let $\pi := (1\ 3)(2\ 4)$
repeat
  $(\mathsf{shuffle}, \langle(1\ 2\ 3\ 4)\rangle)$
  $v_2 := (\mathsf{turn}, \{1\})$
until $v_2 = \pi(v_1)$

$(\mathsf{shuffle}, \langle(2\ 3\ 4)\rangle)$
$v_3 := (\mathsf{turn}, \{2\})$
Let $\sigma := (1\ 4)(2\ 3)$
if $v_3 = \sigma(v_2)$ **then** $(\mathsf{result}, 4, 3)$
else $(\mathsf{result}, 3, 4)$

For an analysis of the number of shuffle steps in the protocol, observe that we have performed two shuffles until we reach the loop condition, which holds

with probability $1/4$. After the loop, we have one additional shuffle step. Hence, the expected number of shuffles is $3 + \sum_{n=1}^{\infty} \left(1 - \frac{1}{4}\right)^n = 6$.

Comparison to [NR99]. The previous protocol, using five cards, was described in the introduction. For a pseudo-code description, see Protocol 2.

Protocol 2. Five-card AND protocol by Niemi and Renvall [NR99]. The first bit is in basis $\{1, 2\}$, the second in basis $\{3, 4\}$. The output basis is $\{1, 4\}$. See also Figure 7 for a KWH tree representation.

```
(perm, (3 4))
repeat
  | (shuffle, ⟨(1 2 3 4 5)⟩)
  | v := (turn, {1})
until v = 2 or v = 3
repeat
  | (shuffle, ⟨(2 3 4 5)⟩)
  | v := (turn, {2})
until v = 2 or v = 3
repeat
  | (shuffle, ⟨(3 4 5)⟩)
  | v := (turn, {3})
until v = 5
(result, 4, 5)
```

As Niemi and Renvall state, their running time in the number of shuffle steps is calculated as follows: Their protocol starts with a shuffle and repeats this with probability $3/5$. The second loop contains a shuffle and has a repeating probability of $3/4$. The shuffle in the final loop is repeated with probability $2/3$. In total, the expected running time is $3 + \sum_{n=1}^{\infty} \left(\frac{3}{5}\right)^n + \sum_{n=1}^{\infty} \left(\frac{3}{4}\right)^n + \sum_{n=1}^{\infty} \left(\frac{2}{3}\right)^n = 3 + 1.5 + 3 + 2 = 9.5$. However, for a fair comparison to our protocol, we eliminate the last loop from their protocol, as its only function is to ensure that the output is in basis $\{1, 4\}$, which our protocol does not guarantee. In this case, the modified Niemi–Renvall protocol has an expected number of $3 + 1.5 + 3 = 7.5$ shuffle steps. Hence, our four-card AND protocol needs one card less and outperforms the Niemi–Renvall protocol by an expected number of 1.5 shuffle steps.

6 Card-Minimal Protocols for Basis Conversion with Overlapping Bases

In this section, we give two protocols for converting a basis encoding in the case where the old and the new encoding share a card. The first protocol has an expected (finite) running time of three shuffle and turn operations. While it has

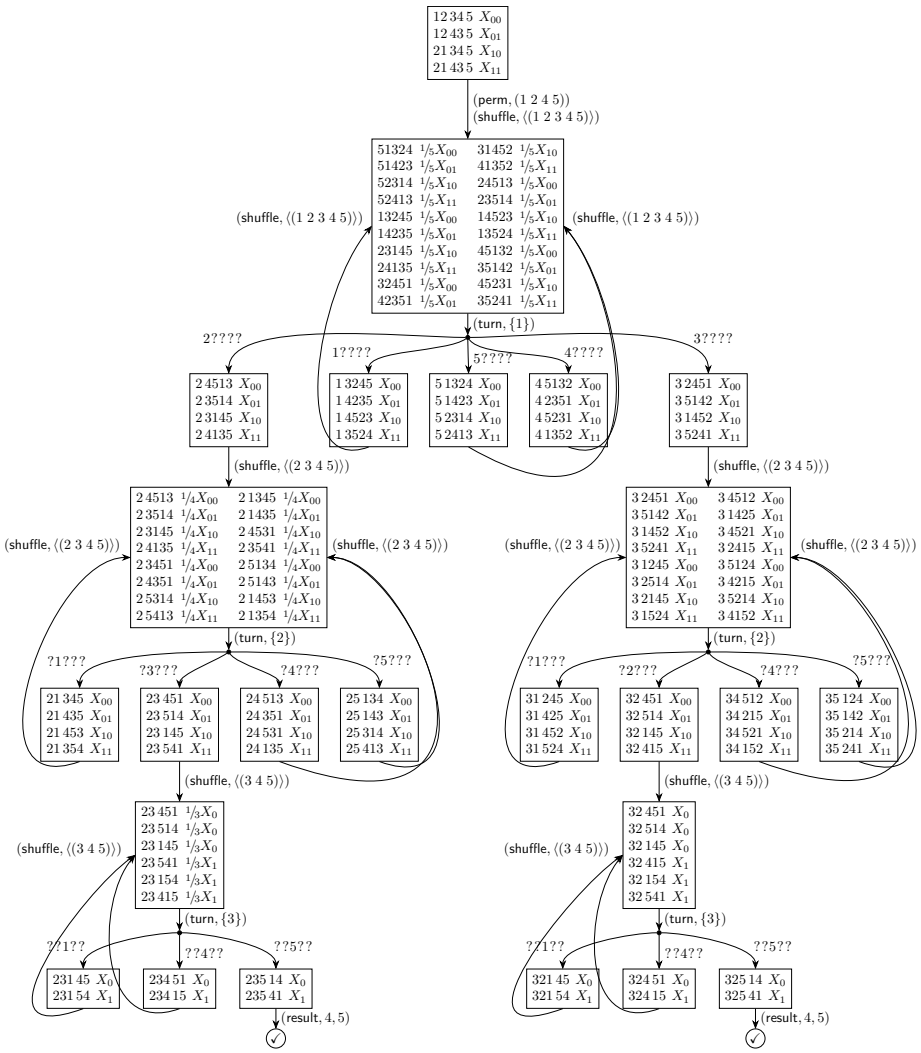


Fig. 7. KWH tree of the five-card AND protocol of [NR99] with $\mathcal{D} = [\![1, 2, 3, 4, 5]\!]$ using only random cuts, cf. Protocol 2. Note that $X_0 := X_{00} + X_{01} + X_{10}$ and $X_1 := X_{11}$. The output is in basis $\{1, 4\}$.

not been explicit in the literature, it is in a way implicit in the protocol by Niemi and Renvall [NR99], as the authors aimed to get a fixed-in-advance output basis.

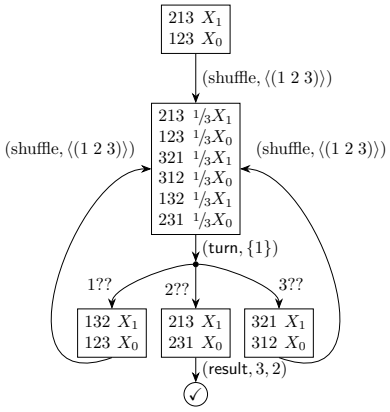


Fig. 8. Three-card Las Vegas basis convert for $\mathcal{D} = [\![1, 2, 3]\!]$ with uniform closed shuffles.

Theorem 4. *There is a three-card Las Vegas basis-conversion protocol for overlapping bases with deck $\mathcal{D} = [\![1, 2, 3]\!]$ and uniform closed shuffles.*

Proof. See Figure 8 and Protocol 3.

Protocol 3. Three-card Las Vegas basis conversion protocol as given in Figure 8 with $\mathcal{D} = [\![1, 2, 3]\!]$, input basis $\{1, 2\}$ and output basis $\{1, 3\}$

repeat
  (shuffle, ⟨(1 2 3)⟩)
  $v :=$ (turn, {1})
until $v = 2$
(result, 3, 2)

Theorem 5. *There is a five-card finite-runtime basis conversion protocol for overlapping bases with deck $\mathcal{D} = [\![1, 2, 3, 4, 5]\!]$. It only uses two random bisection cuts as shuffle operations.*

Proof. This is just applying the basis conversion of [M16] twice, cf. Protocol 4.

Protocol 4. Five-card finite-runtime conversion protocol with overlapping bases for $\mathcal{D} = [\![1, 2, 3, 4, 5]\!]$, input basis $\{1, 2\}$ and output basis $\{1, 3\}$

$(\mathsf{shuffle}, \langle(1\ 2)(4\ 5)\rangle)$
$v := (\mathsf{turn}, \{1\})$
if $v = 2$ **then** $(\mathsf{perm}, (1\ 2)(4\ 5))$

$(\mathsf{shuffle}, \langle(1\ 3)(4\ 5)\rangle)$
$v := (\mathsf{turn}, \{4\})$
if $v = 4$ **then** $(\mathsf{result}, 1, 3)$
else $(\mathsf{result}, 3, 1)$

```
1 struct sequence {
2   uint val[numberOfCards];
3   struct fractions probs;
4 };
```

Listing 1. C struct holding the state trees.

7 An Illustration of Our Verification Methodology

In the following, we exemplify our translation of card-based cryptographic protocols using standard decks to a specific the bounded model checker CBMC which takes programs in the C language, and compute a secure AND function. For our experiments, we used CBMC 5.11 [CKL04] with the built-in solver based on the SAT-solver MiniSat 2.2.0 [ES03]. All experiments are performed on an AMD Opteron(tm) 2431 CPU at 2.40 GHz with 6 cores and 32 GB of RAM.

We translate KWH trees in the C language using a simple encoding into a bounded C program with only static structures and no pointers, e.g., we employ C structs (see Listing 1) holding an array of card sequences for the sequence s, attached with their respective values for each probability (for the probabilistic security notion) or dependency (for output-possibilistic security) X_i occurring in $\mu(s)$, which is simply encoded by another C struct `fractions`. The sequences are constructed using non-deterministic values restricted by respective software conditions to enforce a lexicographic ordering. Moreover, we assign the starting values in $\mu(s)$ with fixed (i.e., deterministic) values based on the constructed sequences. Subsequently, an array of (consecutively) reachable states is constructed non-deterministically using simple implementations of the turn and the shuffle operation as explained in Section 2. We then repeatedly (after each turn/shuffle) check whether all possible resulting (non-deterministic) states correctly and securely compute the specified function, e.g., here a secure AND.

An example shuffle operation is shown in Listing 2 for the case of output-possibilistic security. Therein, the keyword `__CPROVER_assume` is used by the bounded model checker to restrict all program runs passing this statement to satisfy the specified (Boolean) condition. By assigning values using the spe-

```
1 uint permSetSize = nondet_uint();
2 __CPROVER_assume (0 < permSetSize);
3 __CPROVER_assume (permSetSize <= NUM_POSS_SEQ);
4 uint permutationSet[permSetSize][numberOfCards];
5 uint takenPermutations[NUM_POSS_SEQ] = { 0 };
6
7 for (uint i = 0; i < permSetSize; i++) {
8   uint permIndex = nondet_uint();
9   __CPROVER_assume (permIndex < NUM_POSS_SEQ);
10  __CPROVER_assume (!takenPermutations[permIndex]);
11
12  takenPermutations[permIndex] = 1;
13  for (uint j = 0; j < numberOfCards; j++) {
14    permutationSet[i][j] =
15      startState.seq[permIndex][j] - 1;
16  }
17 }
18 struct state result =
19   doShuffle(startState, permutationSet, permSetSize);
20 __CPROVER_assume (isBottomFree(result));
```

Listing 2. Simplified shuffle operation for CBMC.

cial function `nondet_uint()`, we assign a non-deterministic non-negative integer number, which is restricted to values greater than zero and at most of value `NUM_POSS_SEQ` (which is a variable computed by the pre-processor and is the maximum number of sequences possible with the given deck) in the following program statement. In the shown example, the non-determinism is used to construct a set of permitted permutation sets (to be used by the shuffle operation), which makes the SBMC tool inspect the following program code for all possible assignments of this value. If necessary, this may result in a fully exhaustive search, however, the prover is often able to restrict the domain based on further program statements and dependencies seen in the rest of the program. A similar trick is used when computing the concrete permutations using the non-deterministic value of `permIndex` in order to check all possible permutations which possibly move the values, but preserve all existing numbers in the sequence itself. This is done using the `int`-array `takenPermutations`, which is first initialized to zero and, when choosing a concrete permutation, assumed to be zero at position `permIndex`, however set to the number one right afterwards (such that it is not permitted to be chosen again). In the subsequent inner loop, the permutations are assigned choosing the according cards from the sequences in the start state using the non-deterministic value `permIndex`. Finally, the shuffle is applied, resulting in the state variable `result`, which is then checked using a further method `isBottomFree` to not contain any sequences with impermissible values for X_i, which would result in incorrect computations of the AND function.

We applied our approach to the computation of a secure AND protocol using four cards in order to, firstly, substantiate our proof that no protocol of a length below six can be found, and, secondly, automatically find a permitted protocol using six operations. Using our approach, we were able to show that no four-card protocol exists using five operations within 57 hours and constructed an output-possibilistic protocol using six operations within 31 hours. The sizes of the constructed formulas consisted of between 150 and 180 million SAT clauses.

8 Conclusion

In this paper, we proposed a new method to search card-based protocols for any secure computation, by giving a general formal translation applicable to be used by the formal technique of software bounded model checking (SBMC). This method allows us to find new protocols automatically, and prove lower bounds on required shuffle and turn operations for any protocol, and provide an example for the computation of a minimal AND protocol. We also found a new protocol that only uses the theoretical minimum of four distinguishable cards for an AND computation. Moreover, we supported this finding by our automatic method in showing the impossibility of any protocol using less shuffle and turn operations using only practicable shuffles (random cuts). The protocol is hence optimal w.r.t. the running time restriction "restart-free Las-Vegas". For the four-card standard deck setting, we showed that there is no finite runtime protocol, regardless of the shuffle operations used. This result completes the picture of tight lower bounds for the four-card setting. Finally, we showed tight lower bounds on basis conversions for single bits and proposed the missing protocols, and establish the theorem that using a minimum of five cards, both input- and output-bases can be chosen freely, which fosters our impossibility result for the four-card setting.

Open Problems. Let us point out some open problems in the card-based security area that could be approached based on the findings in this paper: (1) For finite-runtime protocols, there exist no proven tight lower bounds on the required number of cards (five to eight cards). We recommend more research applying computer-aided formal methods at this point, as the state space for five or more cards is very large. (2) Our verification approach is fast for finding protocols and/or lower bounds on the operations needed in a protocol for given shuffle-restrictions. However, this is based on the assumption that protocols exist already for a given predefined length to find or confirm impossibility results. Investigating computer-aided formal methods for universal impossibility results might be worthwhile. (3) The two most common settings in card-based cryptography are the standard deck setting with only distinguishable cards and the two-color decks using ♣ and ♡. However, it may be possible that by mixing these settings (e.g., only distinguishable cards with one pair of identical cards), we might find more efficient protocols (especially in the finite runtime setting). For such a mixed setting, [SM19] provide nice results to use in further research.

Appendix: Protocols from the Literature

This appendix contains the 8-card AND protocol of [M16] (Figure 9) and a second four-card protocol which uses a number of 4.5 shuffles in expectation, which are, however, non-closed and hence, more impractical to implement, cf. Figure 10.

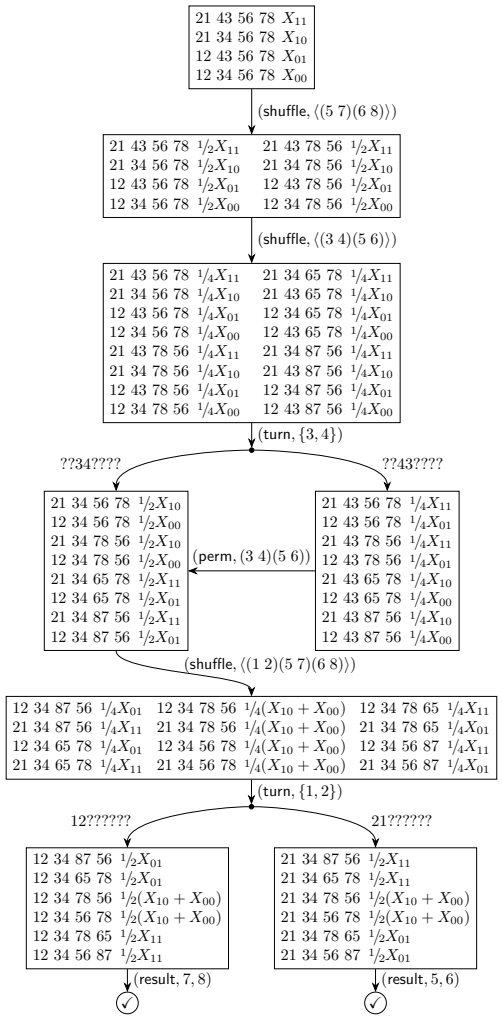


Fig. 9. The eight-card finite-runtime AND protocol of [M16], with $\mathcal{D} = [\![1, \ldots, 8]\!]$ and uniform-closed shuffles. Output is in basis $\{5, 6\}$ or $\{7, 8\}$, each with probability $1/2$.

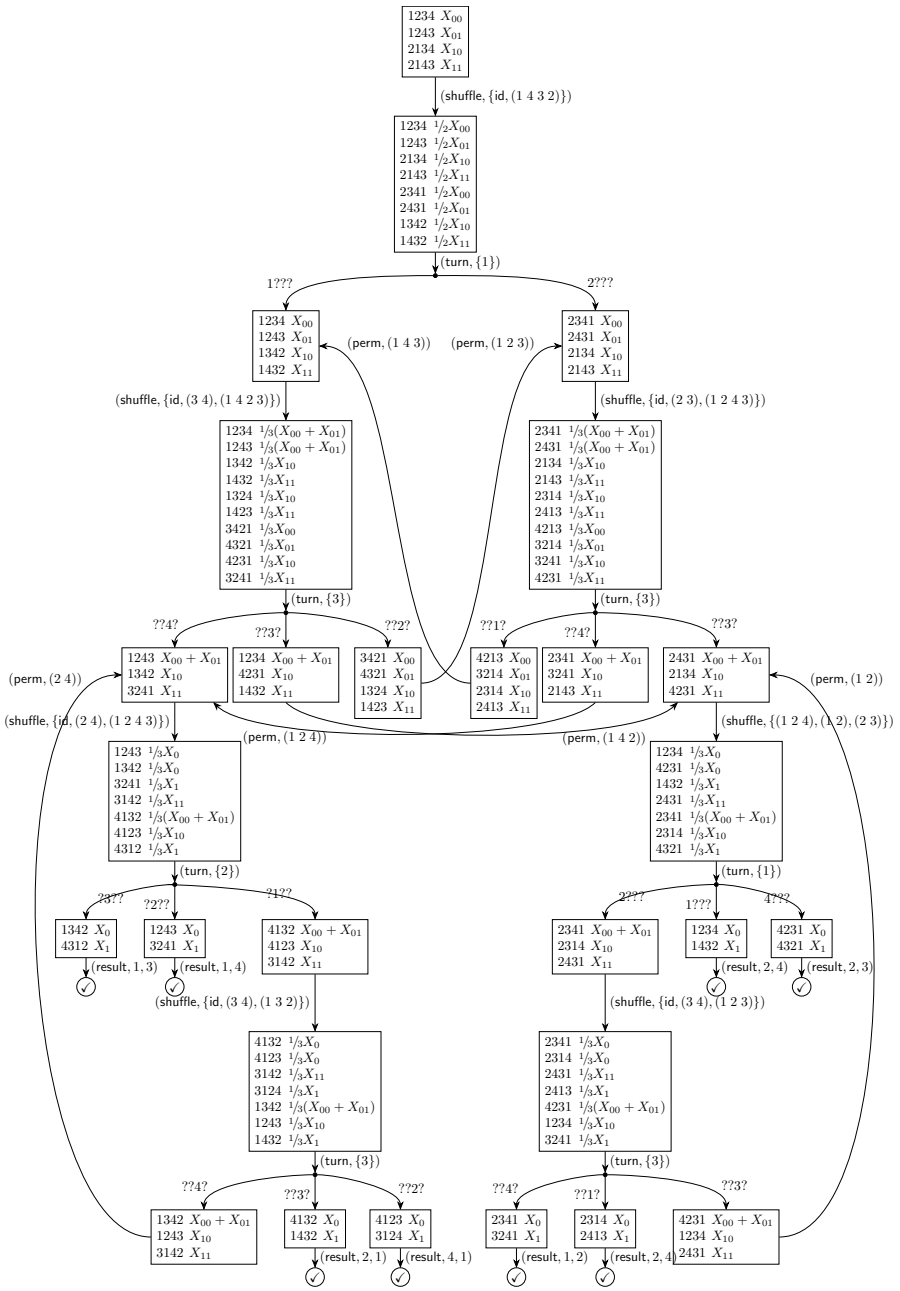


Fig. 10. A four-card Las Vegas AND protocol with deck $\mathcal{D} = [\![1, 2, 3, 4]\!]$ and uniform shuffles. Note that $X_0 \coloneqq X_{00} + X_{01} + X_{10}$ and $X_1 \coloneqq X_{11}$. The output is in one of the bases $\{1, 3\}, \{1, 4\}, \{2, 3\}, \{3, 4\}$, determined by the position of the final state in the tree, and can be converted as needed.

References

[AHM+18] Y. Abe, Y.-i. Hayashi, T. Mizuki, and H. Sone. "Five-Card AND Protocol in Committed Format Using Only Practical Shuffles". In: *APKC@AsiaCCS 2018*. Ed. by K. Emura et al. ACM, 2018, pp. 3–8. DOI: 10.1145/3197507.3197510.

[APS14] M. Avalle, A. Pironti, and R. Sisto. "Formal verification of security protocol implementations: a survey". In: *Formal Asp. Comput.* 26.1 (2014), pp. 99–123. DOI: 10.1007/s00165-012-0269-9.

[B12] B. Blanchet. "Security Protocol Verification: Symbolic and Computational Models". In: *POST 2012*. Ed. by P. Degano and J. D. Guttman. LNCS 7215. Springer, 2012, pp. 3–29. DOI: 10.1007/978-3-642-28641-4_2.

[BCC+99] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. "Symbolic Model Checking without BDDs". In: *TACAS 1999*. Ed. by R. Cleaveland. LNCS 1579. Springer, 1999. DOI: 10.1007/3-540-49059-0_14.

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

[CKL04] E. M. Clarke, D. Kroening, and F. Lerda. "A Tool for Checking ANSI-C Programs". In: *TACAS 2004*. Ed. by K. Jensen and A. Podelski. LNCS 2988. Springer, 2004, pp. 168–176. DOI: 10.1007/978-3-540-24730-2_15.

[dB89] B. den Boer. "More Efficient Match-Making and Satisfiability: The Five Card Trick". In: *EUROCRYPT '89*. Ed. by J. Quisquater and J. Vandewalle. LNCS 434. Springer, 1989, pp. 208–217. DOI: 10.1007/3-540-46885-4_23.

[ES03] N. Eén and N. Sörensson. "An Extensible SAT-solver". In: *SAT 2003*. Ed. by E. Giunchiglia and A. Tacchella. LNCS 2919. Springer, 2003, pp. 502–518. DOI: 10.1007/978-3-540-24605-3_37.

[FFN14] B. Fisch, D. Freund, and M. Naor. "Physical Zero-Knowledge Proofs of Physical Properties". In: *CRYPTO 2014*. Ed. by J. A. Garay and R. Gennaro. LNCS 8617. Springer, 2014, pp. 313–336. DOI: 10.1007/978-3-662-44381-1_18.

[FHK+14] M. Franz, A. Holzer, S. Katzenbeisser, C. Schallhart, and H. Veith. "CBMC-GC: An ANSI C Compiler for Secure Two-Party Computations". In: *CC 2014*. Ed. by A. Cohen. LNCS 8409. Springer, 2014, pp. 244–249. DOI: 10.1007/978-3-642-54807-9_15.

[GBG14] A. Glaser, B. Barak, and R. J. Goldston. "A zero-knowledge protocol for nuclear warhead verification". In: *Nature* 510 (2014), pp. 497–502. DOI: 10.1038/nature13457.

[K18] A. Koch. *The Landscape of Optimal Card-based Protocols*. 2018. Cryptology ePrint Archive, Report 2018/951.

[K19] A. Koch. "Cryptographic Protocols from Physical Assumptions". PhD thesis. Karlsruhe Institute of Technology (KIT), 2019. DOI: 10.5445/IR/1000097756.

[KKW+17] J. Kastner, A. Koch, S. Walzer, D. Miyahara, Y.-i. Hayashi, T. Mizuki, and H. Sone. "The Minimum Number of Cards in Practical Card-based Protocols". In: *ASIACRYPT 2017*. Ed. by T. Takagi and T. Peyrin. LNCS 10626. Springer, 2017, pp. 126–155. DOI: 10.1007/978-3-319-70700-6_5.

[KW17] A. Koch and S. Walzer. *Foundations for Actively Secure Card-based Cryptography*. 2017. Cryptology ePrint Archive, Report 2017/423.

[KWH15] A. Koch, S. Walzer, and K. Härtel. "Card-based Cryptographic Protocols Using a Minimal Number of Cards". In: *ASIACRYPT 2015*. Ed. by T. Iwata and J. H. Cheon. LNCS 9452. Springer, 2015, pp. 783–807. DOI: 10.1007/978-3-662-48797-6_32.

[M16] T. Mizuki. "Efficient and Secure Multiparty Computations Using a Standard Deck of Playing Cards". In: *CANS 2016*. Ed. by S. Foresti and G. Persiano. LNCS 10052. Springer, 2016, pp. 484–499. DOI: 10.1007/978-3-319-48965-0_29.

[MN10] T. Moran and M. Naor. "Basing cryptographic protocols on tamper-evident seals". In: *Theor. Comput. Sci.* 411.10 (2010), pp. 1283–1310. DOI: 10.1016/j.tcs.2009.10.023.

[MS09] T. Mizuki and H. Sone. "Six-Card Secure AND and Four-Card Secure XOR". In: *FAW 2009*. Ed. by X. Deng et al. LNCS 5598. Springer, 2009, pp. 358–369. DOI: 10.1007/978-3-642-02270-8_36.

[MS14] T. Mizuki and H. Shizuya. "A formalization of card-based cryptographic protocols via abstract machine". In: *Int. J. Inf. Sec.* 13.1 (2014), pp. 15–23. DOI: 10.1007/s10207-013-0219-4.

[MS17] T. Mizuki and H. Shizuya. "Computational Model of Card-Based Cryptographic Protocols and Its Applications". In: *IEICE Transactions* 100-A.1 (2017), pp. 3–11. DOI: 10.1587/transfun.E100.A.3.

[NR98] V. Niemi and A. Renvall. "Secure Multiparty Computations Without Computers". In: *Theor. Comput. Sci.* 191.1-2 (1998), pp. 173–183. DOI: 10.1016/S0304-3975(97)00107-2.

[NR99] V. Niemi and A. Renvall. "Solitaire Zero-knowledge". In: *Fundam. Inform.* 38.1-2 (1999), pp. 181–188. DOI: 10.3233/FI-1999-381214.

[RSH19] A. Rastogi, N. Swamy, and M. Hicks. "Wys*: A DSL for Verified Secure Multi-party Computations". In: *POST 2019*. Ed. by F. Nielson and D. Sands. LNCS 11426. Springer, 2019, pp. 99–122. DOI: 10.1007/978-3-030-17138-4_5.

[SHK+16] N. Swamy, C. Hritcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P. Strub, M. Kohlweiss, J. K. Zinzindohoue, and S. Z. Béguelin. "Dependent types and multi-monadic effects in F". In: *POPL 2016*. Ed. by R. Bodık and R. Majumdar. ACM, 2016, pp. 256–270. DOI: 10.1145/2837614.2837655.

[SM19] K. Shinagawa and T. Mizuki. "Secure Computation of Any Boolean Function Based on Any Deck of Cards". In: *FAW 2019*. Ed. by Y. Chen et al. LNCS 11458. Springer, 2019, pp. 63–75. DOI: 10.1007/978-3-030-18126-0_6.