Karlsruher Institut für Technologie

Cryptographic Protocols from Physical Assumptions

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Alexander Koch

Tag der mündlichen Prüfung:	5. Juli 2019
Erster Referent:	Prof. Dr. Jörn Müller-Quade
Zweiter Referent:	Prof. Dr. Marc Fischlin

Cryptographic Protocols from Physical Assumptions

Inaugural dissertation for the fulfillment of the requirements for the academic degree of Doctor of Natural Sciences (Dr. rer. nat.) accepted by the KIT Department of Informatics of the Karlsruhe Institute of Technology (KIT) submitted by Alexander Koch

Day of examination: July 5, 2019
First Referee: Prof. Dr. rer. nat. Jörn Müller-Quade, Karlsruhe Institute of Technology
Second Referee: Prof. Dr. phil. nat. Marc Fischlin, Technical University Darmstadt

Published by KIT, Karlsruhe, Germany.
DOI: 10.5445/IR/1000097756.

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Prof. Dr. Jörn Müller-Quade, for his continuous support, his motivation and his enthusiastic way of teaching and doing cryptography. I also greatly enjoyed his guidance and the flexibility and freedom to choose my research interests. I am also very thankful to Prof. Dr. Marc Fischlin for offering to serve as a second reviewer of the thesis.

I am deeply grateful for Prof. Dr. Ryo Nishimaki for his excellent supervision, and Prof. Dr. Masayuki Abe for providing me the terrific opportunity to do a research internship at NTT Secure Platform Laboratories in Summer 2017. I particularly enjoyed the productive, collegial and international atmosphere. Hence, a big thanks to all colleagues and co-interns. I would like to express a particular thanks to Saho Uchida from NTT for her kind support and for allowing me to practice my Japanese in lively conversations.

A very special thanks goes to Stefan Walzer for countless invaluable discussions and advice, our longstanding fruitful collaboration, and his enthusiasm for our joint projects. I would also like to thank my other coauthors, for the very productive joint work and the intensive hours we spent together in front of the whiteboard. Let me express my sincere gratitude to all my former and present colleagues and visiting researchers which made spending time at the institute a pleasure. Also, thanks to many colleagues and coauthors offering to proofread parts of the thesis.

Last but not least, I cannot put into words the exceptional and constant support by my family, friends and Thomas. Without them, I would not be where I am now.

Abstract

Modern cryptography does not only enable to protect your personal data on the Internet, or to authenticate for certain services, but also evaluate a function on private inputs of multiple parties, without anyone being able to learn something about these inputs[1]. Cryptographic protocols of this type are called *secure multiparty computation* and are suitable for a broad spectrum of applications, such as for voting or auctions, where the vote or bid should remain private.

To prove security of such protocols, one needs assumptions that are often complexity-theoretic in nature – for example that it is difficult to factorize sufficiently large numbers. Security assumptions that are based on *physical principles* exhibit quite some advantages when compared to complexity-theoretic assumptions: the protocols are often conceptually simpler, the security is independent of the computational power of an attacker, and the functioning and security is often more transparent to humans[2]. Examples of such assumptions are physically isolated or incorruptible hardware components (cf. [BKM+18]), write-only devices for logging, or scratch-off cards as common in letters for personal PINs. Also, the non-cloneability of quantum states that follows from the principles of quantum theory, is a physical security assumption that is, e.g., used to realize non-cloneable "quantum money".

This dissertation covers, besides protocols that use the security of certain simple hardware components as a trust anchor, particularly cryptographic protocols for secure multiparty computation that is executed with the help of a deck of *physical playing cards*. The security assumption is that the cards have indistinguishable backs and that certain shuffling actions can be performed securely. One application of these protocols is a didactic method to illustrate cryptography and to allow for secure multiparty computation that can be performed completely without any computer/hardware involved.

In this area of cryptography, researchers aim to construct protocols using a minimal number of cards – and to prove them optimal in this sense. Depending on the requirements posed to running time (finite vs. finite only in expectation) and to the practicability of the used shuffles, one can derive different lower bounds for the necessary number of cards. This thesis derives a lower bound for all combinations of these requirements in the case of AND[3] and constructs or identifies protocols in the literature that use this minimal number of cards. In total, AND is possible and optimal with

[1] With the exception of knowledge that can be deduced from the output and own inputs efficiently.

[2] For example, the German Constitutional Court demanded: "When electronic voting machines are deployed, it must be possible for the citizen to check the essential steps in the election act and in the ascertainment of the results reliably and without special expert knowledge." (BVerfG, Judgment of the Second Senate of 03 March 2009).

[3] A logical AND of two bits encoded in cards; together with negation and duplication of bits, this is sufficient for computing arbitrary circuits.

four (in the case of expectedly finite running time [KWH15; K18]), five (in the case of requiring practicable shuffling *or* a finite running time [KWH15; K18]) or six cards (for finite running time *and simultaneously* using only practicable shuffling [KKW+17]).

For the necessary structural insights we developed "state diagrams", a graph-based representation of all possible protocol runs, which allow for direct reading of correctness and security from the diagram [KWH15; KKW+17]. This method has since found broad usage in the area-relevant literature. (With this method, proofs of lower bounds with respect to the number of cards become proofs that certain protocol states are not reachable in the associated combinatorial graph structure.) Using this, we were able to formalize the respective notions of card-based cryptography as a C program and prove the run-minimality (given certain restrictions) of a card-minimal AND protocol using a Software Bounded Model Checking approach [KSK19].

Moreover, we give conceptionally simple protocols in the case of secure multiparty computation that additionally protects the *function to be computed* [KW18] for each of the following computational models: (universal) circuits, branching programs, Turing machines and RAM machines. Furthermore, we give an analysis of how card-based protocols can be executed such that the only interaction is in the other parties watching for the correct execution of the protocol. This allows for (weakly interactive in the aforementioned sense) program obfuscation, where one party can execute a program encoded in cards on his own inputs, without learning anything about its internal workings, except what can be deduced from input- and output-behavior. This is in general impossible without such physical assumptions. Additionally, we formalize a security notion against active attackers and specify a method that – using very weak security assumptions – compiles a passively secure protocol fulfilling certain conditions into an actively secure version [KW17].

A second physical security assumption analyzed in this thesis is to assume primitive, incorruptible hardware, such as a TAN generator. This allows for example a secure authentication of a human user via a corrupted/untrusted terminal, without requiring that the user conducts cryptographic computation by herself (such as multiplying large primes). A construction is given in the case of money withdrawal at a corrupted ATM with the help of a very simple trusted second device and with very weak security assumptions to the available communication channels [AGH+19b]. The given protocol remains secure when being run concurrently with other protocols (exhibits so-called Universal Composability security), was designed in a modular way, and uses a plausible security assumption. Hence, its functioning is transparent to humans.

Overall, by giving several card-based protocols, systematized proof methods for showing lower bounds and respective proofs, as well as our results for the secure utilization of a non-trusted terminal, together with a categorization of these into a systematic presentation of the different physical assumptions used in cryptography, this thesis presents a significant contribution to cryptography based on physical assumptions.

Zusammenfassung

Moderne Kryptographie erlaubt nicht nur, personenbezogene Daten im Internet zu schützen oder sich für bestimmte Dienste zu authentifizieren, sondern ermöglicht auch das Auswerten einer Funktion auf geheimen Eingaben mehrerer Parteien, ohne dass dabei etwas über diese Eingaben gelernt werden kann[4]. Kryptographische Protokolle dieser Art werden *sichere Mehrparteienberechnung* genannt und eignen sich für ein breites Anwendungsspektrum, wie z.B. geheime Abstimmungen und Auktionen.

Um die Sicherheit solcher Protokolle zu beweisen, werden Annahmen benötigt, die oft komplexitätstheoretischer Natur sind, beispielsweise, dass es schwierig ist, hinreichend große Zahlen zu faktorisieren. Sicherheitsannahmen, die auf *physikalischen Prinzipien* basieren, bieten im Gegensatz zu komplexitätstheoretischen Annahmen jedoch einige Vorteile: die Protokolle sind meist konzeptionell einfacher, die Sicherheit ist unabhängig von den Berechnungskapazitäten des Angreifers, und die Funktionsweise und Sicherheit ist oft für den Menschen leichter nachvollziehbar[5]. Beispiele für solche Annahmen sind physikalisch getrennte oder unkorrumpierbare Hardware-Komponenten (vgl. [BKM+18]), Write-Only-Geräte für Logging, oder frei zu rubbelnde Felder, wie man sie von PIN-Briefen kennt. Auch die aus der Quantentheorie folgende Nicht-Duplizierbarkeit von Quantenzuständen ist eine physikalische Sicherheitsannahme, die z.B. verwendet wird, um nicht-klonbares „Quantengeld“ zu realisieren.

In der vorliegenden Dissertation geht es neben Protokollen, die die Sicherheit und Isolation bestimmter einfacher Hardware-Komponenten als Vertrauensanker verwenden, im Besonderen um kryptographischen Protokolle für die sichere Mehrparteienberechnung, die mit Hilfe *physikalischer Spielkarten* durchgeführt werden. Die Sicherheitsannahme besteht darin, dass die Karten ununterscheidbare Rückseiten haben und, dass bestimmte Mischoperationen sicher durchgeführt werden können. Eine Anwendung dieser Protokolle liegt also in der Veranschaulichung von Kryptographie und in der Ermöglichung sicherer Mehrparteienberechnungen, die gänzlich ohne Computer ausgeführt werden können.

Ein Ziel in diesem Bereich der Kryptographie ist es, Protokolle anzugeben, die möglichst wenige Karten benötigen – und sie als optimal in diesem Sinne zu beweisen. Abhängig von Anforderungen an das Laufzeitverhalten (endliche vs. lediglich im Erwartungswert endliche Laufzeit) und an die Praktikabilität der eingesetzten Mischoperationen, ergeben sich unterschiedliche untere Schranken für die mindestens benötigte

[4] Mit der Ausnahme von Informationen, die aus der Ausgabe und eigenen Eingaben effizient abgeleitet werden können

[5] Zum Beispiel forderte das Bundesverfassungsgericht: „Beim Einsatz elektronischer Wahlgeräte müssen die wesentlichen Schritte der Wahlhandlung und der Ergebnisermittlung vom Bürger zuverlässig und ohne besondere Sachkenntnis überprüft werden können.“ (BVerfG, Urteil des Zweiten Senats vom 03. März 2009)

Kartenanzahl. Im Rahmen der Arbeit wird für jede Kombination dieser Anforderungen ein UND-Protokoll[6] konstruiert oder in der Literatur identifiziert, das mit der minimalen Anzahl an Karten auskommt, und dies auch als Karten-minimal bewiesen. Insgesamt ist UND mit vier (für erwartet endliche Laufzeit [KWH15; K18]), fünf (für praktikable Mischoperationen *oder* endliche Laufzeit [KWH15; K18]) oder sechs Karten (für endliche Laufzeit *und gleichzeitig* praktikable Mischoperationen [KKW+17]) möglich und optimal.

Für die notwendigen Struktureinsichten wurden so-genannte „Zustandsdiagramme" mit zugehörigen Kalkülregeln entwickelt, die eine graphenbasierte Darstellung aller möglichen Protokolldurchläufe darstellen und an denen Korrektheit und Sicherheit der Protokolle direkt ablesbar sind [KWH15; KKW+17]. Dieser Kalkül hat seitdem eine breite Verwendung in der bereichsrelevanten Literatur gefunden. (Beweise für untere Schranken bzgl. der Kartenanzahl werden durch den Kalkül zu Beweisen, die zeigen, dass bestimmte Protokollzustände in einer bestimmten kombinatorischen Graphenstruktur nicht erreichbar sind.) Mit Hilfe des Kalküls wurden Begriffe der Spielkartenkryptographie als C-Programm formalisiert und (unter bestimmten Einschränkungen) mit einem „ Software Bounded Model Checking"-Ansatz die Längenminimalität eines kartenminimalen UND-Protokolls bewiesen [KSK19].

Darüber hinaus werden konzeptionell einfache Protokolle für den Fall einer sicheren Mehrparteienberechnung angegeben, bei der sogar zusätzlich die *zu berechnende Funktion geheim* bleiben soll [KW18], und zwar für jedes der folgenden Berechnungsmodelle: (universelle) Schaltkreise, binäre Entscheidungsdiagramme, Turingmaschinen und RAM-Maschinen. Es wird zudem untersucht, wie Karten-basierte Protokolle so ausgeführt werden können, dass die einzige Interaktion darin besteht, dass andere Parteien die korrekte Ausführung überwachen. Dies ermöglicht eine (schwach interaktive) Programm-Obfuszierung, bei der eine Partei ein durch Karten codiertes Programm auf eigenen Eingaben ausführen kann, ohne etwas über dessen interne Funktionsweise zu lernen, das über das Ein-/Ausgabeverhalten hinaus geht. Dies ist ohne derartige physikalische Annahmen i.A. nicht möglich. Zusätzlich wird eine Sicherheit gegen Angreifer, die auch vom Protokoll abweichen dürfen, formalisiert und es wird eine Methode angegeben um unter möglichst schwachen Sicherheitsannahmen ein passiv sicheres Protokoll mechanisch in ein aktiv sicheres zu transformieren [KW17].

Eine weitere, in der Dissertation untersuchte physikalische Sicherheitsannahme, ist die Annahme primitiver, unkorrumpierbarer Hardware-Bausteine, wie z.B. einen TAN-Generator. Dies ermöglicht z.B. eine sichere Authentifikation des menschlichen Nutzers über ein korrumpiertes Terminal, ohne dass der Nutzer selbst kryptographische Berechnungen durchführen muss (z.B. große Primzahlen zu multiplizieren). Dies wird am Beispiel des Geldabhebens an einem *korrumpierten Geldautomaten* mit Hilfe eines als sicher angenommenen zweiten Geräts [AGH+19b] und mit möglichst schwachen Anforderungen an die vorhandenen Kommunikationskanäle gelöst. Da das angegebene Protokoll auch sicher ist, wenn es beliebig mit anderen gleichzeitig laufenden Protokollen ausgeführt wird (also sogenannte Universelle Komponierbarkeit aufweist), es modular

[6] Ein logisches UND zweier in Karten codierter Bits; dieses ist zusammen mit der Negation und dem Kopieren von Bits hinreichend für die Realisierung allgemeiner Schaltkreise.

entworfen wurde, und die Sicherheitsannahme glaubwürdig ist, ist die Funktionsweise für den Menschen transparent und nachvollziehbar.

Insgesamt bildet die Arbeit durch die verschiedenen Karten-basierten Protokolle, Kalküle und systematisierten Beweise für untere Schranken bzgl. der Kartenanzahl, sowie durch Ergebnisse zur sicheren Verwendung eines nicht-vertrauenswürdigen Terminals, und einer Einordnung dieser in eine systematische Darstellung der verschiedenen, in der Kryptographie verwendeten physikalischen Annahmen, einen wesentlichen Beitrag zur physikalisch-basierten Kryptographie.

Contents

Part I.
The Landscape of Security from Physical Assumptions

1. Introduction

While the physical world surrounds us in our day-to-day life since the beginning, the digital world (the "Neuland") has (by comparison) only very recently started to permeate our world. In the (admittedly rather digital) field of cryptography, it has seemingly been the other way round.

Yes, cryptographers have established a long-standing tradition of telling interesting stories, and to use real-world analogies to explain their craft. Usually the narrative includes at least two protagonists, Alice and Bob[1], who have to protect not only from honest-but-curious Eve, but also from malicious Malory, and may involve locked boxes which are passed between the parties to illustrate the concept of a key exchange protocol. *But*, a formal, rigorous and fruitful treatment of physical methods and assumptions to develop cryptographic protocols is quite recent and still flourishing.

This is especially true as using physical objects allow us to circumvent many impossibility results. For example, tamper-proof hardware tokens due to [K07] (to be discussed in the next chapter) suffice to construct protocols for general "secure computation" with very strong (composable) security guarantees without the need of a trusted authority to set up, e.g., a public-key infrastructure – something that was thought impossible before, due to seminal results by Canetti and Fischlin [CF01] and Canetti, Kushilevitz, and Lindell [CKL03]. Here, a *secure computation* involves multiple parties, which would like to jointly evaluate a function (such as: "who of them is richest"), without giving away anything about the individual parties' input values that is not obvious from the output. Throughout this thesis, we will see many instances of such computations. An additional example of avoiding impossibilities when trying to authenticate as a human via an untrusted platform, is given in Part III on "human-friendly electronic payment".

Moreover, protocols employing physical assumptions may offer qualitatively stronger security guarantees, or other security aspects, such as fairness (informally: if a protocol yields an output, all parties learn it), using different trust models, deniability and non-coercion. A particular advantage that only-digital protocols cannot offer, is to provide a *bridge to reality*. Examples of this are given in [GBG14; FFN14], where the authors provide a protocol for proving (in zero-knowledge) that a nuclear warhead that is to be disarmed due to an international treaty conforms to a certain prescribed template, without giving away anything about its internal design.

Also, due to our familiarity with the physical world, many protocols that make use of day-to-day objects, such as envelopes or ballots used in cryptographic voting schemes, are often much easier to understand, or are just more transparent than computer hardware executing some program. This might be crucial for a protocol to be even

[1] Appearing at the beginning of time, at least counting from [RSA78] – not to make use of tricks in https://xkcd.com/1323/.

considered for real-world use. As cited in the abstract, the German Constitutional Court demanded that "[w]hen electronic voting machines are deployed, it must be possible for the citizen to check the essential steps in the election act and in the ascertainment of the results reliably and without special expert knowledge." (BVerfG, Judgment of the Second Senate of 03 March 2009). Finally, physical tools such as the wooden boxes mentioned in the beginning, can be used fruitfully for didactics. Besides giving the opportunity to do secure computation without a computer, this is one of the main aims of card-based cryptography, which will be covered in detail in Part II.

Outline of the Thesis

Let us give a short overview on how this thesis is organized. The main focus of Part I is to give an overview of how physical security assumptions have been used in the area of cryptography and provable security. We will categorize such physical cryptography broadly into three domains (by its main object), namely to obtain security guarantees with i) idealized *hardware* covered in Chapter 2, ii) idealized, easily-manipulateable physical *objects* in Chapter 3 and iii) idealized physical *processes* or *properties*. To keep the focus of the thesis, we omit the third category from our discussion, as it is not connected to research results contained in this thesis. Following this exposition, we summarize the main contribution of the thesis, describe how these results fit into the developed landscape and give an overview of the author's publications in Chapter 4.

Part II contains results on secure computation with a deck of cards. It starts with an introduction to card-based cryptography (Chapter 5), followed by Chapter 6 on the basic computational model. We describe our state diagram formalism for secure protocols in Chapter 7 and give several new protocols in Chapter 8. One of the main results of this thesis is to deduce how many cards are necessary for AND and COPY protocols, also with respect to certain restrictions to running time and practicability of the involved shuffles. This is given in Chapter 9, together with an extensive part on methodology of such proofs. Additionally, we apply methods of formal verification (bounded model checking) to prove the minimality of a protocol's shortest run in Chapter 10. Protocols which additionally protect the function to be evaluated, and hence allow for program obfuscation with cards, are given in Chapter 11. Finally, Chapter 12 contains a method to achieve actively secure card-based protocols and discusses the insecurity against malicious attacks for several protocols from the literature.

In Part III, we make use of partial hardware trust assumptions and use the physical security of certain channels to implement secure protocols for electronic payment and money withdrawal at a possibly malicious ATM, involving a human user. The problem is introduced in Chapter 13. Chapter 14 contains our model of secure payment in the UC framework, which is a separate contribution. In Chapter 15 we derive requirements for secure payment, and use these findings to show that many real-world protocols for money withdrawal do not satisfy this criterion, in Chapter 16. Finally, a secure protocol for money withdrawal is proposed in Chapter 17. Part IV concludes, points to some resonance in the literature that the present work has evoked and identifies future research directions in the field of cryptography from physical security assumptions.

2. Security from Idealized Hardware

Many researchers have suggested using hardware as a facilitator for cryptographic protocols or as a trust anchor, based on a wide range of security properties or functionality features that such hardware might exhibit. Let us go through the most important and impactful hardware assumptions used for cryptographic protocols. Throughout this chapter, we mostly focus on results that attain security in the strong Universal Composability (UC) framework. For an introduction to UC-security, we refer to Section 13.3.

2.1. Physically Uncloneable Functions

Introduced as "Physical One-Way Functions" by Pappu et al. [PRTG02], so-called Physically Uncloneable Functions (PUFs) are simple, stateless hardware modules which serve as a (noisy) evaluation of a function with high min-entropy output. The used manufacturing process unavoidably and purposefully includes imprecision and slight variations in the chip, leading to a response behavior that is hard to clone. Although the concrete security requirements depend on the cryptographic protocol or application it is to be used in, many definitions include one-wayness and unforgeability of the output. Armknecht et al. [AMSY16] give a good survey on these notions and integrates them into one consistent and unified framework. For a survey on technical implementations of PUFs, see [KKR+12].

When it comes to the achievable level of security when using PUFs as hardware given to the players, Brzuska et al. [BFSK11] were the first to give a construction of secure multiparty computation in the UC framework that is even unconditionally secure. However, as noted by Ostrovsky et al. [OSVW13], their construction assumes that the PUFs in use are fully trusted. An adversary that is able to create a hardware chip which looks and behaves like a PUF can easily break the security of their scheme. These malicious PUFs come in two flavors: they can be either stateful[1] or stateless, with the latter being plausibly much more easy to craft in a way that it is as simple (and indistinguishable from the outside) as the PUF that is to be imitated. In the setting of stateful malicious PUFs, Ostrovsky et al. [OSVW13] present a protocol for secure computation that achieves computational UC-security. *Unconditional* UC-security in the (stateful) malicious PUF-setting has only be attained for commitment protocols, cf. [DS13]. Indeed, Dachman-Soled et al. [DFK+14] show that unconditional general computation and oblivious transfer is impossible, even in the stand-alone setting. On

[1] While these PUFs may, e.g., log all input-output values, it is important to note that PUFs neither can communicate back to the adversary, nor does he get access to the PUF after the protocol.

the positive side, they construct an unconditional and efficient UC-secure protocol for general computation, if one restricts the adversary to only issue stateless PUFs.

2.2. Signature Cards

Hofheinz, Müller-Quade, and Unruh [HMU07] proposed using trusted signature cards as an alternative setup assumption for UC-secure computation. These are modeled as an ideal hardware functionality which, upon receiving a message as an input by its holder, outputs a signature to it. Using these, one can also obtain UC-security in the long-term setting, as shown in [MU07]. Long-term security guarantees that the cryptographic protocol remains secure even if the adversary gets unlimited computational power after the protocol has ended, which is a very interesting security feature in the light of possible future technological advances that might threaten currently-used cryptographic schemes. One example of such an imminent threat is the possible advent of scalable quantum computers.

As further research has shown, one can also handle *untrusted* signature cards, e.g., by restricting to signature schemes which have a unique (non-randomized) signature for each message, and by taking special care for the case when the card aborts dependent on the message to be signed, thereby leaking to the outside that a message from a certain set was to be signed. See [MMN18] for reference, which additionally achieves reusability of the signature card in multiple protocols without impacting security, as in the global UC framework introduced in [CDPW07]. ([HMU07] also offer reusability of signature cards, but their signature cards are not modeled in the global UC framework.)

2.3. Tamper-Proof Hardware Tokens

Besides hardware modules which compute a random function (such as PUFs) or implement a signing functionality, one can also assume hardware, such as smart or SIM cards or USB authentication tokens, which can execute arbitrary code. These have been proposed by Katz [K07] as a setup assumption for UC-secure computation. In contrast to previous setup assumptions, such as a common reference string or a public-key infrastructure, this does not need a trusted central party responsible for establishing the setup assumption. Here, the security relies on two assumptions: i) the code of the token and any internal secrets are completely opaque to the holder, i.e., the token is *tamper-proof* and cannot be brought to reveal its secrets by any engineering measure, and ii) the token cannot communicate with the outside (except possibly through regular protocol messages), in particular, it cannot send any security-relevant information back to its original creator.

Tamper-proof hardware mainly comes in two flavors, dependent on how lightweight the used hardware is supposed to be: i) *stateful* tokens which can reliably store data and keep a non-trivial internal state, ii) *stateless* or (stateful but) *resettable* tokens (such as a smart-card reliant on an external power source) which should still work if an adversary repeatedly cuts off power to the token and thereby resetting its state.

Differently from the discussion on PUFs, malicious tokens are always assumed to be able to keep a state, aiding the purposes of the attack.

Obviously, assuming stateful tokens is a much stronger assumption, leading to strong feasibility results. For example, Goyal et al. [GIS+10] show that even non-interactive and unconditionally UC-secure two-party computation is possible with simple stateful tamper-proof hardware tokens against malicious adversaries. Here, non-interactive two-party computation starts with a single charge of tokens being sent from sender to receiver, followed by a computation phase without any communication. In terms of efficiency, [DKM11; DKM12] were able to reduce the number of necessary stateful tokens to the provable minimum of one and two tokens for interactive and non-interactive two-party computation, respectively, while retaining unconditional UC security in both cases. (Note that using a single token is especially beneficial, as one does not need to take into account the threat of multiple malicious tokens covertly communicating at the receiver's place.)

In the case of stateless or resettable tokens, one is restricted to realizing functionalities that are compatible with being reset at any time in the protocol, called "resettable functionality" in the following. First of all note that (as pointed out in [GIS+10]) stateless tokens by themselves cannot achieve unconditional security, as an unbounded adversary can completely learn the behavior of the token (unless one restricts the number of resets, as in [DS13; DKMN15b]). In this setting, due to [DMMN13; DKMN15a], one can achieve arbitrary resettable two-party functionalities with UC-security using only a single token and the existence of one-way functions. More recently, [HPV17] constructed constant-round adaptively secure protocols which allow all parties to be corrupted. As a variant of the tamper-proof hardware model, Fisch, Freund, and Naor [FFN15] suggested to use disposable circuits which can be completely destroyed after the computation, to realize unconditional UC-security computation (with input-dependent abort). This suggestion is especially useful in the context of physical computations [FFN14], such as determining or proving a match of certain genes, where one needs an "information barrier" after the protocol. A good survey on the use of tamper-proof hardware tokens can be found in [N15].

2.4. Trusted Hardware with Constrained Functionality

The above assumptions on secure hardware have first been introduced as a trusted functionality, evoking a search for generalizations to the respective untrusted hardware assumptions. This is understandable, as, e.g., a trusted PUF is a strong assumption. However, if the functionality to be implemented is very simple and can be formally verified as a fixed-function logic circuit, and it might also be plausible to build the hardware yourself or to obtain them in usual electronic stores – making targeted attacks much more difficult – then, such trust assumptions become more plausible. Moreover, often these modules do not carry any secrets themselves, making the tamper-proofness assumptions as above less important. Hence, it is worthwhile to consider such *trusted* hardware modules, in particular if they lead to strong security results. As an example,

[AMR15] uses a very simple secure equality check hardware module, to ensure the correct, UC-secure functioning of a parallel firewall setup, protecting against a malicious firewall. We will use the assumption of a trusted TAN generator or optical code reader in Part III of this thesis.

An example of achieving qualitatively stronger security by using trusted hardware such as data-diodes, air-gap switches and "output interface modules" is the *Fortified UC* framework [BKM+18], of which I am a co-author[2]. A more detailed description is given in Section 4.3.

Secure Oblivious Bingo Voting [ALMR16; ABL+17], where the voting machine does not even learn the vote cast by the user, has been realized with a special trusted physical module. Also in the context of voting, Moran, Naor, and Segev [MNS07] consider write-once memory as a trusted hardware assumptions to securely store votes even in adversarial environments. These or even simpler write-only devices, such as printer can also be used to achieve secure logging. Next, we discuss a special case of trusted hardware that deserves separate mentioning, due to its refined modeling and more complex functionality.

2.5. Secure Processors

Recently, processors that allow for *attested execution* in a sandboxed environment, a so-called *enclave*, such as Intel's SGX technology, have been formally modeled by Pass, Shi, and Tramèr [PST17] in the global UC framework. An attested execution of a program P on inputs i outputs not only the result, but also a signature on P and the output, certifying that the program has been correctly executed on this processor, resulting in the respective output. As the execution happens in an enclave, it is protected against tampering and other forms of modification and/or leakage, dependent on how weak the security assumptions are to be modeled. For example, [PST17; TZL+17] describe a variant where all internal secrets of the computation are allowed to leak (but not the signing key for the attestation), so-called transparent enclaves. Using these, they were able to construct UC-secure commitments and zero-knowledge proofs, which remain secure in the global UC framework, hence allowing for reusability of the processors after the protocol, cf. the discussion above in the case of reusable signature cards. Other interesting applications of secure processors are, e.g., given in [FVBG17] – implementing functional encryption using Intel SGX enclaves – and [NFR+17], which implements obfuscation for RAM programs in a very strong (virtual-black-box) sense. General program obfuscation of this strength has been shown to be impossible in software. One contribution of this thesis is to give an obfuscation of the same strength in the card-based cryptography setting, cf. Chapter 11.

[2] It was not included in this dissertation to keep the focus on security mechanisms which involve human aspects, and there is already an excellent and extensive description in the dissertation of the first author, cf. [B19].

3. Security from Physical Objects

In contrast to secure hardware, physical objects do not carry any internal logic or programming but are specifically-crafted or day-to-day things that can be used for cryptographic protocols. These are sometimes used as inspiration or analogy, such as when using boxes and locks to explain key exchange protocols, but can also be thought to really achieve a cryptographic computation, e.g., without a computer, and might thereby offer more tangible and transparent security than the digital counterparts. The most prominent example is in cryptographic voting schemes, where physical ballots or envelopes are formally modeled with a concrete security goal in mind.

Apart from these, one can also use these objects more broadly for recreational cryptography, or didactics, e.g., when illustrating secure multiparty computation or zero-knowledge proofs to university or high-school students. Finally, some of these are suitable for the theoretical interest of studying unconventional computational models.

3.1. Physical Envelopes and Ballots

Exploiting physical properties of voting machines or ballots is common in the cryptographic voting community to achieve seemingly contradictory properties such as receipt-freeness (informally, one cannot show others a receipt from which they can derive information about how one has voted, an important property for non-coercibility), and (public) verifiability of the fact that one's vote has been counted. For example, PunchScan [PH10] employs ballots which consist of two sheets of paper, fixed together but separable. The sheet on top has holes through which a code for the available options is visible. When voting, an ink punching device that is large enough to mark both sheets of paper when applied to a holes, is used. Crucial for the security of the scheme is that the two sheets of the ballot, taken together, do uniquely determine which hole is to be punched for the respective voting choice, but when separated, each single sheet does not give away anything about the vote. Hence, after voting, only one part of the ballot can be taken home by the voter to be used for verification that the vote has really been counted, while the other is used for the tally. The protocol is also analyzed more formally in [MN10b, Sect. 2.4].

Another voting scheme that uses physical properties is Scantegrity [CCC+09; CCC+10], and the schemes of Moran and Naor [MN06b; MN06a]. One ingredient common to many voting schemes are *tamper-evident seals*, such as envelopes, locked but breakable boxes or scratch-off cards, cf. [MN10a]. Besides voting, they allow to execute many cryptographic primitives, such as oblivious transfer and bit commitment (albeit not obfuscation). They distinguish four types, dependent on whether the seals are all indistinguishable, and on whether honest players have the ability to open a locked seal

(such as a closed envelope) and achieve distinct feasibility results for the different seal types.

In [IML05], the authors do not only introduce interesting ideas, and problems of game theory to the field of cryptography, including the notion of a "rational attacker" which tries to maximize its utility, but also employ a ballot-box and envelopes to implement unconditionally UC-secure multiparty computation in this rational attacker model (without an honest majority). In [ILM08; ILMas], they extend their ballot-box method to form the notion of "Verifiably Secure Devices" or "Transparent Computation", which essentially describes a transparent procedure implemented by a human or device employing such a ballot box functionality, to compute the desired functionality.[1]

3.2. Cryptography with Playing Cards

Secure multiparty computation can be done with a *deck of physical cards*, as first shown in [dB90; CK94; NR98]. In this area of *card-based cryptography*, one designs tangible protocols using a deck of cards with information-theoretic privacy features. We will cover these extensively in Part II and hence will focus on other uses of playing cards to attain interesting security properties.

Famously, Schneier [S99] invented a symmetric cipher, called Solitaire, that is executed with cards. While biased and hence not really secure, it is advertised with non-digital features such as plausible deniability (everyone may carry a deck of cards) and fast "secure erasure" (shuffling destroys the key) in case of a physical search. Toponce [T18] provides an overview over several alternative card ciphers that have been proposed and which exploit the fact that generating randomness is simple when shuffling cards, to be easy to execute.

In a different vein, cards have evoked the interest of cryptographers and researchers in combinatorics, via the following "Russian cards problem" introduced by Fischer and Wright [FW96]. Here, a prescribed number of cards is dealt to Alice, Bob and also Eve, and Alice and Bob want to establish a common secret about which Eve is left clueless, by publicly announcing some information about their cards. This is, e.g., further analyzed in [MSN99; AAA+05].

3.3. Cryptography with Other Objects

Fagin, Naor, and Winkler [FNW96] give a nice introduction to cryptography for the task of securely comparing private values, utilizing different physical objects, which besides cards and envelopes also includes a discussion on using cups and Airline reservation hotlines.

[1] Using these, they present a protocol to compute a certain game-theoretic equilibrium, which is a certain beneficial subclass of Nash equilibria that are efficiently reachable given a correlated starting strategy for the players, usually to be set up by a trusted party (which the authors avoid). The UC-security follows from the fulfillment of information-theoretic conditions due to [DM00].

Balogh et al. [BCIK03] establish secure multiparty computation for arbitrary Boolean functions using a (rather large) PEZ dispenser ideal functionality with ideal PEZ candies of two colors, where candies of the same color are indistinguishable. Here, each player may privately dispense a number of candies, dependent on the input and only this player learns the order and color of its candies. After the protocol, the color of the candy that is would be dispensed if one pressed once more, encodes the output bit. While certainly curious and evoking amusements, their research was motivated by the question of how far can one can go with a *deterministic* ideal functionality, to which one cannot trivially offload all the computation: "The main conclusion of our work is a surprising affirmative answer to an instance of this question: even a severely handicapped, physically realizable, and inherently 'leaky' trusted party (as the [PEZ dispenser]) allows nontrivial deterministic secure computation."

Naor and Shamir [NS95] invented "visual secret sharing", allowing to create physical transparent slides, each with random (but correlated) dot patterns, such that when both are placed on top of each other, a black-and-white image appears. (There have been many extensions, also to allow colors and detectability if someone uses a maliciously created transparency. Moreover, it has been used in an early version of the PunchScan scheme described above, for the two sheets of the ballot.)

Researchers have been creative in employing other objects for secure multiparty computation. For example, [MKS07b] describes how to compute any function with up to four variables using a 15 Puzzle where one can turn over the tiles to conceal their symbol, similarly to turnable cards in card-based cryptography. Moreover, Mizuki, Kugimoto, and Sone [MKS07a] found a way to use a dial lock to compute a specific class of functions securely. Similarly to card-based protocols, [SMS+15b] proposes simple protocols using polarized plates.

Physical computation is also described in [CV12], as "Physical GMW protocol", to achieve security in the framework of Universal Composability with Local Adversaries (LUC). However, they make very strong assumptions on available "machines" (the can not, e.g., use the reorderability that is possible with cards).

4. Contributions and Publication Overview

In this chapter, I would like to put our results into the context of the depicted landscape of security from physical assumptions, thereby briefly summarizing the contributions with reference to publications, on which this thesis is built.

4.1. Contribution to Security from Physical Objects

All of Part II can be seen as a direct contribution to the field of security from idealized physical objects, as it explores models, methods and protocols for secure card-based computations. Let us give a very brief summary, with a more detailed contribution to be found at the respective referenced chapters, when the used notions have been introduced in a bit more detail. In all these works, I am a lead/main author.

Parts of Chapters 6 to 9 are based on the following paper:

[KWH15] A. Koch, S. Walzer, and K. Härtel. "Card-Based Cryptographic Protocols Using a Minimal Number of Cards". In: *Advances in Cryptology – ASIACRYPT 2015*. Proceedings, Part I: 21st International Conference on the Theory and Application of Cryptology and Information Security (Auckland, New Zealand, Nov. 29–Dec. 3, 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. © International Association for Cryptologic Research (IACR) 2015.

Here, we introduce state diagrams, construct four- and five-card AND protocols and a protocol for an arbitrary k-ary Boolean function, and give the first lower-bound result on the number of cards for AND, namely that AND protocols with a finite running time require five cards. Moreover, we discuss restrictions to the underlying computational model.

Similarly, essential parts of Chapters 6 to 9 are also based on the following paper:

[KKW+17] J. Kastner, A. Koch, S. Walzer, D. Miyahara, Y. Hayashi, T. Mizuki, and H. Sone. "The Minimum Number of Cards in Practical Card-Based Protocols". In: *Advances in Cryptology – ASIACRYPT 2017*. Proceedings, Part III: 23rd International Conference on the Theory and Applications of Cryptology and Information Security (Hong Kong, China, Dec. 3–7, 2017). Ed. by T. Takagi and T. Peyrin. LNCS 10626. Springer, 2017, pp. 126–155. DOI: 10.1007/978-3-319-70700-6_5. © IACR 2017.

This paper basically extends all lower bounds results and corresponding methods to AND protocols with restricted (practicable) shuffles and to COPY protocols, which are also an essential building block for computing arbitrary Boolean circuits. It is a merge of two papers, where Sections 9.4 and 9.5 stems from the last four authors.

Chapter 12 is based on the following publication:

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

Here, we extensively cover the case of active security in card-based cryptography, which has not been systematically studied before. Most importantly, we give a generic compiler which transforms any passively secure card-based protocol into an actively secure version, given certain requirements to the used shuffling actions.

A larger part of Chapter 9, but also some part of Chapter 8 is taken from:

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

This paper completes the picture of tight lower bounds for AND and COPY, by extending and systematizing the methodology of lower bound proofs for card-based protocols, showing lower bounds for these functionalities and giving two additional AND protocols.

Chapter 11 and Section 8.6 of Chapter 8 are from:

[KW18] A. Koch and S. Walzer. *Private Function Evaluation with Cards*. 2018. Cryptology ePrint Archive, Report 2018/1113. In submission.

Here, we consider the case of secure multiparty computation where the function to be computed is a secret itself, and establish secure card-based protocols for universal circuits, branching programs, Turing machines and RAM machines. For this, we develop the abstraction of a "sorting" protocol, which allows to sort cards with respect to the order of another card sequence, obliviously. Many protocols from the literature can be framed as such a sorting protocol.

Focussing on the case of a deck where all cards are distinct, the following paper is the basis for parts of Chapter 8, but also contains Chapter 10:

[KSK19] A. Koch, M. Schrempp, and M. Kirsten. "Card-based Cryptography Meets Formal Verification". In: *Advances in Cryptology – ASIACRYPT 2019*. Proceedings: 25th Annual International Conference on the Theory and Application of Cryptology and Information Security (Kobe, Japan, Dec. 8–12, 2019). Ed. by S. Galbraith and S. Moriai. LNCS. Springer, 2019. © IACR 2019. In press.

Here, we construct a new four-card AND protocol for this deck type (improving on a five-card protocol) and introduce formal verification to the area of card-based cryptography. Using a developed bounded model checking technique, we prove that the new protocol has an execution path of minimal length. Finally, we consider protocols for card encoding conversions and give partial results on lower bounds for AND in this setting.

4.2. Contribution to Security from Idealized Hardware

Part III of this thesis is directly based on the following publication, to which I contributed significantly:

[AGH+19b] D. Achenbach, R. Gröll, T. Hackenjos, A. Koch, B. Löwe, J. Mechler, J. Müller-Quade, and J. Rill. "Your Money or Your Life—Modeling and Analyzing the Security of Electronic Payment in the UC Framework". In: *Financial Cryptography and Data Security. 23rd International Conference*. Revised Selected Papers: FC 2019 (Frigate Bay, Saint Kitts and Nevis, Feb. 18–22, 2019). Ed. by I. Goldberg and T. Moore. LNCS. Springer, 2019. Cryptology ePrint Archive, Report 2019/924. © International Financial Cryptography Association 2019. In press.

Here, we model electronic payment and money withdrawal in the UC framework, where a human initiator would like, e.g., to authenticate an amount to be withdrawn at an ATM which might be malicious. This contributes to the field of security from idealized hardware, as we explore the setting of, e.g., a trusted TAN generator for authentication. Additionally, more broadly connected to physical security, we analyze and employ different types of communication channels that are available due to the physical situation that the human interacts in (e.g., his ability to cover the PIN pad and thereby establishing a confidential channel to the ATM). Using these constraints and resources, we construct a UC-secure payment protocol. For other, more general contributions of the paper, including further modeling aspects and necessary criteria for security in our setting, see Section 13.1.

4.3. Other Works

This section contains a summary of my other works. Only the first has a connection to the topic of cryptography from physical assumptions.

Fortified Universal Composability

The Fortified Universal Composability framework [BKM+18] models different types of (physical) channels between the parties and/or between several simple, unhackable hardware modules, in particular channels with an *air-gap switch* (disconnect-able by the party in control of the switch) or with a *data diode* (then constituting a unidirectional channel). Using the isolation assumptions that come with these channel types, one can define what it means to be connected (by a path through the network) to "the outside" at a given point in time. The attacker model is then extended from i) static (physical) corruptions that are performed before parties are invoked[1], to ii) *remote hacks*, i.e., attacks that are possible during the protocol run, but only when connected

[1] Imagine an attacker, getting physical access to your device beforehand and soldering integrated circuits (ICs) to your mainboard, thereby permanently corrupting them.

to the outside. Additionally, one assumes an unhackable encryption module as well as an "output interface module", where the latter performs the task of demasking the output dependent on a successful MAC tag verification. Note that all modules do not need to carry pre-loaded secrets themselves and in total they implement very simple functions. (Moreover, there are variants which use two more such modules, if one aims for reactive functionalities or for security in the case that all parties are corrupted.)

We give a construction of protocols for secure multiparty computation with a security notion that is qualitatively stronger than commonly aimed-for *adaptive security* in that the inputs and outputs of all parties are completely protected (w.r.t. confidentiality and integrity) against remote hacks, unless they happen before the party received its input, or the attacker gains control over all parties. In contrast, ordinary adaptive security does not protect the inputs and outputs, i.e., they can be learned and modified by the adversary.

The new security definitions are given in an extended version of the Universal Composability (UC) framework [C01], and also feature the modularity and composable security of UC. However, note that the trusted hardware and channels assumed here are not already sufficient to circumvent the impossibility of commitments or oblivious transfer [CF01; CKL03]. (Hence, these assumptions are weaker than, e.g., tamper-proof hardware tokens, as discussed above.)

Fault-Tolerant Aggregate Signatures

In [HKK+16], we develop a notion of fault-tolerance for aggregate signatures schemes and give a generic construction that uses the combinatorial structure of cover-free families to attain the specified fault-tolerance guarantees. Aggregate signatures are signatures that simultaneously certify multiple claims (i.e., statements that a certain message has been signed by a specific user's signing key), and can be generated from single or aggregate signatures using an pre-defined aggregation method, cf. [BGLS03]. In usual (non-fault-tolerant) schemes, a single faulty signature in the aggregate renders the whole aggregate invalid (by the verification algorithm's output), without providing information on which claim to be verified is the culprit. Our construction assures that the verification algorithm outputs all signatures that are valid, provided that the number of faults does not exceed an a priori bound. (The length of an aggregate signature is logarithmic in the bound of tolerable faulty signatures.)

Practical and Robust Secure Logging

As a follow-up work to fault-tolerant aggregate signatures as described before, [HKK+17] develops a secure logging scheme that incorporates a similar fault-tolerance mechanism, adapted to signature schemes with sequential aggregation (cf. [LMRS04]) which are additionally forward secure, i.e., where the signing key is updated repeatedly with the guarantee that if an attacker breaks into the system and learns the current signing key, this does not help him to forge signatures for earlier signing keys. In the constructed logging scheme, signatures for the individual log entries are created as "part" of an

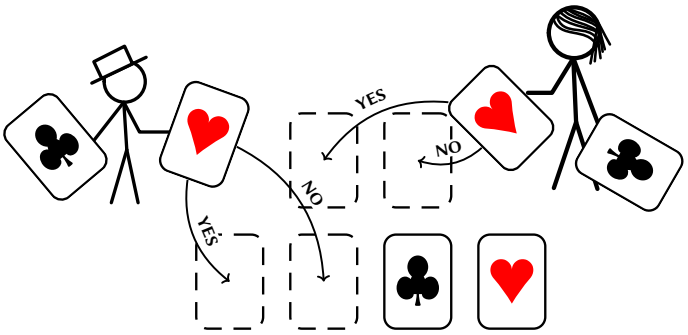
aggregate signature by the sequential aggregation process (also featuring forward security). In combination with a forward-secure signature on the length of the log file, this achieves truncation security, i.e., an attacker cannot revert the log file to a state from before the current signing key was active. In the logging scenario, fault-tolerance is particularly interesting: without this feature, a single bit-flip in one of the log entries leaves the administrator clueless on where the error/modification has occurred. Besides giving a formal model of secure logging and a matching space-efficient construction, the paper includes an implementation together with a performance evaluation.

Part II.
Secure Computation with Cards

5. Introducing Cryptography with Cards

Card-based cryptography is best illustrated by example. Let us begin by giving a concise and graphical description[1] of the six-card AND protocol of Mizuki and Sone [MS09]. This protocol enables two players, Alice and Bob, to securely compute the AND of their private bits. For instance, they may wish to determine whether they both like to watch a particular movie, without giving away their (possibly embarrassing) preference if there is no match. Using card-based cryptography, this is possible without computers – making the security tangible and eliminating the danger of malware.[2]

For this, we use a deck of six cards with indistinguishable backs and either ♡ or ♣ on the front. Each player is handed one card of each symbol and is asked to enter his or her bit by arranging the cards in one of two ways.

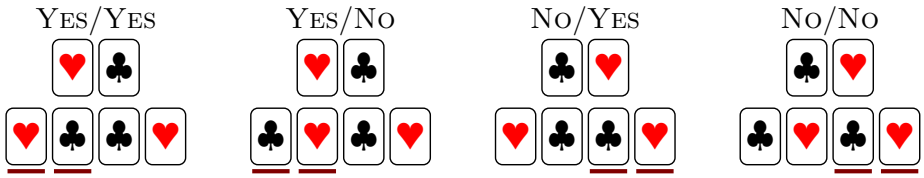


Bob (on the left) inputs "yes", by placing his ♡-card in the first position, and "no" by placing it in the second position; he places his ♣-card in the unused position. Alice encodes her input bit in a similar manner in the first row. We employ two additional cards, encoding "no" (♣♡) in the lower right part of the arrangement. Of course, as the players want their input bit to be secret, their cards are put face-down on the table, hence, making it impossible for the other party to observe the bit at this point. (The extra cards encoding "no" can be put publicly on the table and are turned face-down at the start of the protocol.)

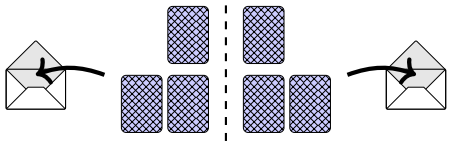
This puts the protocol in one of the following (hidden) configurations:

[1] The introductory exposition is mainly taken from [KKW+17], followed by a merge and new content of introductory sections of the other card-based cryptography papers forming the basis of this thesis.

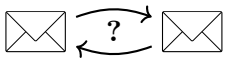
[2] Imagine a setting where Alice asks Bob to enter his bit into an app on a smartphone, which might well raise concerns, even if Bob has the app's source code.



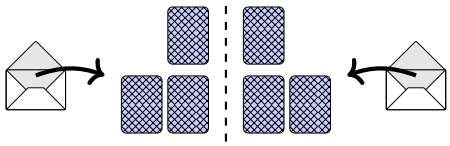
Observe that the *correct result* in the above encoding, (♣♡ = "no", and ♡♣ = "yes"), is *on the side of the heart* in the upper row. This stays invariant, if we split the cards in the middle of the arrangement and exchange both sides. This property is crucial for the protocol, as in the following we want to randomly exchange the two halves of the arrangement to obscure the input order of Alice's cards (they will be inverted with probability one half). For this, it is suggested to split the cards as discussed and put them into two indistinguishable envelopes:



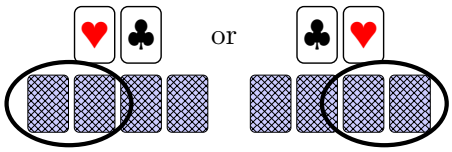
Next, we *shuffle* the envelopes, such that they changed places with probability one half and no player was able to keep track.



We extract the cards again and put them back into the geometric arrangement as before. (This assumes that they have been carefully put into the envelope so that it is still clear which card to place where.)



Due to the shuffling, the upper two cards do not give away Alice's bit any longer, so they can be safely turned over. The invariant ensures that the result is still on the side with the heart:

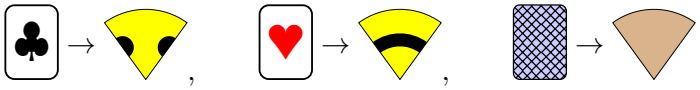


In total, we have performed an AND protocol in *committed format*, as the output are two face-down cards encoding the result. From observing the protocol (as an outsider) we did not learn anything about the order of these cards in the process. We can now decide, whether to open the resulting commitment, or whether to use it as input to some follow-up protocol.

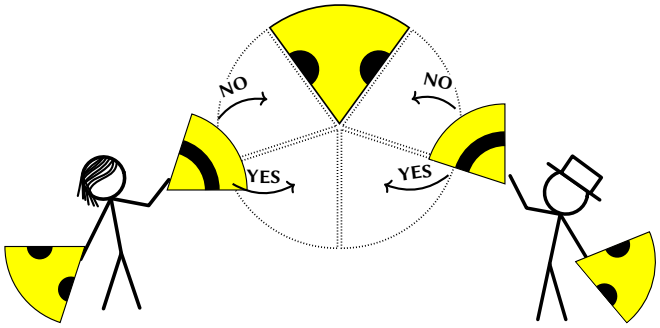
For an alternative way to interpret the steps of this protocol, note that $a \wedge b$ is equivalent to "if a then b else 0". If we would not care about privacy, we could directly turn over Alice's cards and if they show $\heartsuit\clubsuit = 1$, then the result is encoded by Bob's cards (at positions $3, 4$), and otherwise by the two helping cards encoding a 0 (at positions $5, 6$). But privacy requires us to mask Alice's bit first. For this, observe that $a \wedge b$ is *also* equivalent to "if $\neg a$ then 0 else b". Hence, we can invert Alice's bit if and only if we also exchange the two branches of the if-statement. In the example above, the shuffling (exchanging both halves of the configuration with probability $1/2$) does exactly this, namely swapping Alice's cards exactly when Bob's cards are exchanged with the two helping cards encoding 0. This effectively randomizes the order of Alice's cards, which can therefore be turned over without revealing her secret bit. The result is then under the cards as specified above. Because the helping cards and Bob's cards are indistinguishable, one cannot tell which of the two is at the specified output positions. (There is even a third interpretation, similar to the first, given in Section 8.6 on p. 104.)

The Five-Card Trick

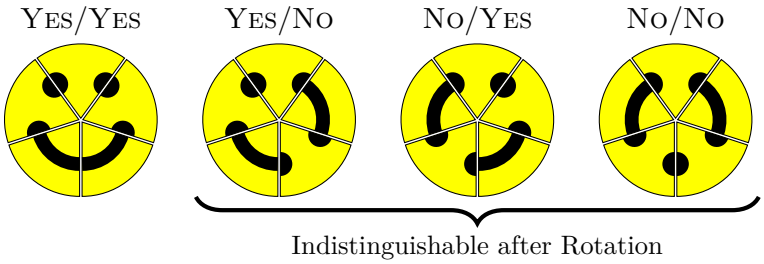
Let us describe one more important card protocol for AND as part of this introduction, namely the "Five-Card Trick" by den Boer [dB90]. This is likely the most well-known card-based protocol, and can be seen as the start of card-based cryptography. Moreover, as it differs from the previous protocol in several aspects, it will allow us to set up all relevant notions in the discussion below. For the presentation, we borrow the following very beautiful translation due to Verhoeff [V14]:



Using this, Alice and Bob can input their bits using two cards as before, with an additional ♣ card in between. The protocol uses a random cut as a shuffling mechanism, i.e., the players take turn in cutting the pile of cards in a fast way, so that nobody can keep track of which card ends up on top of the pile. In total this constitutes a cyclic rotation of the card sequence with unknown offset. Hence, an alternative implementation of the shuffle is to put the cards (with the perfectly suitable shape) face-down on a turntable:



It is easy to see that exactly if both players input "yes", the ♥/ cards (tiles) end up adjacent on the turntable. Hence, we have the following configurations:



The protocol as specified by den Boer ends with turning the face-down cards (or tiles) face-up and when the revealed ♡s are adjacent, this will be interpreted as a "yes", the correct value of the AND computation. In Verhoeff's setting this is encoded via a suggestive emoji (at least for the dating problem). Security of the inputs is immediate, apart from what is obvious from the output, as the three configurations that encode a "no" output are indistinguishable after rotation by a random and oblivious amount.

Note two differences to the previous protocol: i) the first protocol used a shuffle which swaps two halves of a sequence of cards, where the latter employs a random cyclic rotation (a random cut), and ii) we do not get the output in the same format as the inputs (two distinct face-down cards), and cannot directly reuse them in further computations. Hence, it is *not* in committed format. While most of the thesis focuses on committed format[3] protocols, it also includes a framework that let us analyze both protocols using the same, generalized security notion, cf. Chapter 6.

One distinctive feature is that these protocols do not need a computer, which makes their security tangible with no need to rely on opaque hardware or malware-prone operating systems. Hence, they are crucial in any situation where using computers is not an option. This includes scenarios where i) state-level adversaries have control over users' computers[4], ii) there is a plausible fear of Trojans, or iii) we like to prevent cheating in (card) games where computers impede the fun of the game. Moreover, the utility of these protocols is evident from their use in classrooms and lectures to illustrate secure multiparty computation to non-experts to the field of cryptography, or in an introductory course.

As Mizuki, Kumamoto, and Sone [MKS12] were able to reduce the number of cards in the five-card trick to the best possible of 4, which is already necessary to encode the inputs, protocols *not* in committed format are already well-understood in terms of the minimum number of cards. However, given the important disadvantage that they unavoidably reveal the final result during the computation, the quest for card-minimal committed format protocols has emerged as a central research task in card-based cryptography.

[3]First constructed by [NR98; S01; CK94].

[4]The protection we have in mind is against a broad targeting, not a powerful adversary targeting you. It might be comparatively easy to have widely distributed Trojans or a ban on strong cryptography – for attacking card-based MPC in a pervasive surveillance setting, you would need high-resolution high-speed CCTV cameras everywhere, which will be much more costly to implement.

Besides committed-format AND and negation (inverting the cards), there is one more ingredient necessary for computing arbitrary Boolean circuits: commitment copy. A circuit may contain forking wires which enter two or more gates. To see this, consider the three-input majority function as an example:

$$\mathsf{maj}(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_1).$$

Here, before computing $x_1 \wedge x_2$ using an AND protocol, we need to duplicate the input commitment of x_1 and x_2, as they are used in the other clauses as well. (Trusting a user to just input the same bit again is not an option as he or she might deviate and cause wrong outputs, but also because the inputs might not be known to any user if they are the output of some previously run protocol.)

Prior to this work, the protocols using the least number of cards for computing AND in committed format were

- the six-card protocol of Mizuki and Sone [MS09] as shown in the start of the introduction. This has a finite, deterministic running time.
- the five-card Las Vegas protocol of [CHL13], as described in Figure 7.7 on page 72. This protocol may end in a configuration which needs restarting with probability $1/2$ and utilizes a complex shuffle operation. (In the terminology of Section 6.7: it is non-closed.)

(Moreover, concurrent to this work, [AHMS18] published a five-card AND protocol, and also the four- and five-cards AND protocols by [RI19] constitute independent work. These will be discussed later.) This leads to the natural question on the minimality of cards needed for a secure committed format AND, which has been posed in several places in the literature, see, e.g., [MS09; MS14a; MKS12]. Moreover in [CHL13], the authors ask whether there is a "deterministic" five-card variant of their protocol. In this work, we answer these questions and many others, comprehensively.

Of course, the rigorous treatment of questions of this type requires a formal model of card-based computations, which was provided in [MS14a]. There, the authors defined the possible operations that a card-based protocol can make. To allow for strong impossibility results, the authors give a rather wide palette of possible operations that can be applied to the cards, e.g., shuffling with an arbitrary probability distribution on the set of permutations. Our work shows that this yields also rather strong (but unpractical) feasibility results by utilizing "non-closed" shuffles, as defined in Section 6.7.

On the Practicality of Card-based Protocols

Using the very general computational model of Mizuki and Shizuya [MS14a], we present a four-card AND protocol in committed format in Section 8.1, albeit with a running time that is finite only in expectation (a Las Vegas protocol). While this is an interesting achievement, a closer look reveals that there is much more to be hoped for. The most pressing point here is not the protocols' increased complexity, but a certain type of

card shuffling which is much more difficult to perform than in previous protocols. The authors identified two properties whose violation seems to be a cause of the difficulty in their implementation, namely:

Closedness. The set of possible permutations in a card shuffle is invariant under repetition, i.e., a subgroup of the respective permutation group.

Uniform Probability. Every possible permutation in a shuffle action has the same probability.

The most extreme example for the power of non-closed shuffles seems to be the $2k$-card Las Vegas protocol which we will propose in Section 8.3 computing an arbitrary k-ary Boolean function in three steps: we shuffle, turn and either output a result or restart. Here, all the work in computing the function is done by the complex and (in general) non-closed shuffle – suggesting that *non-closed shuffles in general* are too broad a shuffle class to consider. Besides these shuffle restrictions, there is one additional central parameter for *practical* card-based protocols, namely running time behavior. We consider the following practical:

Finite Running Time. This guarantees an a priori bound on the running time and allows to precisely predict how long the protocol will run. We regard this as the most practical.

Restart-Free Las Vegas (LV). While the running time of the protocol is finite only in expectation, it is usually just a small constant. When executing these protocols we may run in cycles but exit these cycles at least with some constant probability in every run. More importantly we do not end in a failure state where we have to restart the whole protocol and query players for their inputs again.

Protocols which do not fall under this class are called *restarting LV* protocols. Note that running a COPY protocol on the input bits before the protocol itself requires already at least five cards for the copy process of a single input bit, as shown in Section 9.4. In total this strategy likely needs more cards than just going for restart-free protocols instead.

One aim of Part II is to derive tight lower bounds on the number of cards for protocols in general, but also for those restricted to using practical – namely (uniform) closed – shuffles and/or a practical running time, namely finite running time or restart-free LV, for the two central ingredients of Boolean circuits: AND protocols and COPY protocols. For n-COPY protocols this can become particularly difficult as there are $(2n+1)!$ permutations on $2n+1$ cards, hence you really need to understand the underlying structure and cannot explore by just trying out some combinations and generalize from there. By giving a systematic approach to impossibility proofs, this work gets around these problems. For example, we show that no closed shuffle protocol can go with less than $2n+2$ cards, even if we allow Las Vegas behavior with restarts and non-uniform shuffles. This completely tightens all bounds w.r.t. running time (finite-runtime, Las Vegas with/without restarts) and shuffle parameters.

We summarize our results in Table 17.1 on p. 226, which also includes a survey on current bounds relative to certain restrictions on the operations.

Note that all protocols are described in the *honest-but-curious* setting (although some analysis of malicious behavior has been done in [MS14b]), i.e., the players execute the protocol according to its description, but gather any information they can possibly obtain. We will cover the case of active security in Chapter 12. The crucial component here is the security of the shuffle action, i.e., the key operations that introduce randomness in a controlled manner. All early protocols (such as the five-card trick) 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 [NR98, Sect. 3] and den Boer [dB90] 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 of the offsets. Other shuffle operations were justified, including "dihedral group" shuffles [NR98] and [S01, Sect. 7], random bisection cuts [MS09; UNH+16] and unequal division shuffles [CHL13; NNH+15; NHMS16].

Besides this thesis, [MS09; MKS12; NNH+15; AHMS18] 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[5] and non-committed format. Moreover, [NHMS15; M16a] and Section 8.3 are concerned with card-minimal protocols for general circuits. Except [FAN+17], which extends the work of [KWH15] to two-bit output functionalities, no other lower bounds on the number of cards appear in the literature, besides the results included in this thesis.

Standard Decks

For simplicity, many protocols in card-based cryptography work with specially created decks, i.e., of only two symbols, ♣ and ♡. This is easy for explanation, and there are nice and easy-to-describe protocols, such as the five-card trick of den Boer [dB90] and the six-card AND protocol of Mizuki and Sone [MS09].

However, the setting where all cards have distinguishable symbols has several advantages. First of all, we assume less about the indistinguishability of cards, which leads to stronger security guarantees. (This is also more similar to the (indistinguishable) model of tamper-evident seals, such as scratch-off cards, by Moran and Naor [MN10a].) We only need that the backs (or envelopes wrapping the cards, if one wishes) are indistinguishable. Secondly, these standard decks are more commonly available, in contrast to specially created decks. If one where to use standard decks for the protocols above, one would need multiple copies of them. 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 work shows, one may needs one card less than with two-symbol decks.

[5] 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 usually leak the output result in the process and are hence unsuitable for larger circuits.

For example, our new four-card Las Vegas AND protocol in Section 8.4 uses only a very basic, practicable shuffling mechanism (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 relatively little research in this direction, with [NR99; M16b] 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 vastly enlarged state space in this domain, as there are many more distinguishable re-orderings of the cards than when only given cards of two symbols.

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 for the possibility of a protocol, whether to give 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 give an analysis of this question, showing that with certain restrictions, there is 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.

6. Models of Card-based Cryptography

Card-based cryptography started out almost thirty years ago with elegant and easy-to-understand descriptions of how to operate on a deck of physical playing cards to securely compute the AND of two bits, as in the five-card protocol by den Boer [dB90]. Since then, around this idea of using face-down cards to represent idealized, perfect commitments to their symbols, a small scientific subfield of cryptography developed, to explore the space of possibilities that actions such as shuffling cards in certain controlled ways allow – as we have to concede that the physical world allows us to do a lot of things.

As any field matures, it needs formal and rigorous definitions of what its central objects are and, e.g., what we mean by properties such as security. This modeling, while important, is often non-trivial and allows for many choices to be made and to be argued for. In the following, starting from the first proposal of Mizuki and Shizuya [MS14a], we give a thorough exposition of how card decks and reorderings of card sequences are modeled (Sections 6.1 and 6.2), what protocols are formally (Section 6.3), how to encode bits and what it means for a protocol to compute a Boolean function (Sections 6.4 and 6.5). Moreover, we give two comprehensive definitions of security (Section 6.6), discuss certain natural subclasses of general card shuffling and identify additional variants and modifications to the defined model (Sections 6.7 and 6.8), conclude and discuss these and other choices for modeling card-based protocols in the literature (Section 6.9).

This chapter is an attempt to unify the different notions and concepts used in the literature into one framework, to rigorously define security in the strongest attainable sense, and to propose some restrictions to the computational model that still allow to do everything that is cryptographically meaningful.

6.1. Decks and Card Sequences

Let Σ and B be disjoint sets. A *card* c of *deck alphabet* Σ and *back alphabet* B is a pair denoted as $c = {}^{a}\!/\!_{b}$ where either $a \in \Sigma$ and $b \in B$ or the other way round. It is *face-up*, if $a \in \Sigma$ and $b \in B$, and *face-down* otherwise. We call c's entry from Σ the *symbol of* c, and its entry from B the *back of* c. The card then represents a face-down or face-up physical card displaying a to the public, while hiding b. We assume that the special symbol '?' is contained in any back alphabet and that, unless explicitly specified, cards have back alphabet $B = \{?\}$, which we then omit from the specification. If no ambiguities arise, we identify face-down cards with their symbol and just speak about a "card a" with $a \in \Sigma$ to denote a card ${}^{?}\!/\!_{a}$.

To model decks which may contain multiple identical cards, we use *multisets*, which – in contrast to sets – can contain elements more than once. To disambiguate, we enclose multisets in brackets as $[\cdot]$. A *deck* $\mathcal{D}$ *over deck alphabet* Σ is then a finite multiset of cards of deck alphabet Σ, which we assume w.l.o.g. to be face-down. For example, $[{}^{?}\!/\!_{\heartsuit}, {}^{?}\!/\!_{\heartsuit}, {}^{?}\!/\!_{\clubsuit}, {}^{?}\!/\!_{\clubsuit}]$, which we will write more concisely as $[\heartsuit, \heartsuit, \clubsuit, \clubsuit]$ or $[2 \cdot \heartsuit, 2 \cdot \clubsuit]$, is a deck over deck alphabet $\{\heartsuit, \clubsuit\}$, consisting of two cards with symbol $\heartsuit$ and two cards with symbol $\clubsuit$. The most commonly used decks in the literature are such *two-color decks*, which use a deck alphabet with two elements, and *standard decks*, in which each card occurs only once, such as $[1, \ldots, 52]$.[1] For notation, we denote unions of multisets by $\cup$, and disjoint unions by $+$, e.g., $[\diamondsuit, \spadesuit, \spadesuit] \cup [\diamondsuit, \heartsuit, \spadesuit, \clubsuit] = [\diamondsuit, \heartsuit, \spadesuit, \spadesuit, \clubsuit]$ whereas $[\diamondsuit, \spadesuit, \spadesuit] + [\diamondsuit, \heartsuit, \spadesuit, \clubsuit] = [\diamondsuit, \diamondsuit, \heartsuit, \spadesuit, \spadesuit, \spadesuit, \clubsuit]$.

For a card $c = {}^{a}\!/\!_{b}$, $\mathsf{vis}(c) = a$ denotes the "numerator" of c, i.e., the visible part of the card, and $\mathsf{symb}(c)$ is the symbol of c. In card-based protocols, the cards are lying on the table in a *sequence*. Hence, a full-deck card sequence is obtained by permuting $\mathcal{D}$ and choosing face-up or face-down for each card. We extend $\mathsf{vis}(\cdot)$ and $\mathsf{symb}(\cdot)$ from cards to card sequences in the canonical way. For a sequence Γ, we call $\mathsf{vis}(\Gamma)$ the *visible sequence* of Γ. As an example, let $\Gamma = ({}^{?}\!/\!_{\clubsuit}, {}^{\clubsuit}\!/\!_{?}, {}^{\heartsuit}\!/\!_{?}, {}^{?}\!/\!_{\heartsuit}, {}^{?}\!/\!_{\clubsuit})$ be a sequence of deck $\mathcal{D} = [\clubsuit, \clubsuit, \clubsuit, \heartsuit, \heartsuit]$, then its visible sequence is $(?, \clubsuit, \heartsuit, ?, ?)$. The set of all sequences and the set of all visible sequences on deck $\mathcal{D}$ of length $n \leq |\mathcal{D}|$ is denoted as $\mathsf{Seq}_n^{\mathcal{D}}$ and $\mathsf{Vis}_n^{\mathcal{D}}$, respectively. If $n = |\mathcal{D}|$, we just write $\mathsf{Seq}^{\mathcal{D}}$ and $\mathsf{Vis}^{\mathcal{D}}$ and omit the subscript. For notation, the ith entry of a sequence or tuple x is denoted by $x[i]$, and we denote the *concatenation* of two sequences x, y of length i and j by $x \,\|\, y \coloneqq (x[1], \ldots, x[i], y[1], \ldots, y[j])$.

6.2. Permutations, Groups and Group Actions

Central to card-based cryptography is reordering card sequences. Formally, a *permutation* of a set X is a bijective map $\pi \colon X \to X$. For $n \in \mathbb{N}$, the set S_n of all permutations of $\{1, \ldots, n\}$ is called *symmetric group*. It has a group structure with the identity map id as neutral element and composition $(\circ)$ as group operation. Note that $(\pi_2 \circ \pi_1)(x) = \pi_2(\pi_1(x))$ meaning π_1 is applied before π_2.

We will frequently make use of cycle notation for permutations. For distinct elements $x_1, \ldots, x_k \in X$ the *cycle* $(x_1\ x_2\ \ldots\ x_k)$ denotes 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 $x \in X$ not occurring in the cycle. The domain of π is not apparent from the cycle alone but can be any superset of the elements in the cycle. For multiple cycles 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

[1] While two-color decks appear more often in the literature than standard decks, we follow Mizuki [M16b] in naming them in this way, because decks of common card games (with all cards being distinguishable from each other) are a good representation of these formal decks. These decks were first used by Niemi and Renvall [NR99] for card-based protocols.

can be written in such a *cycle decomposition*. The *cycle structure* of a permutation is the multiset of cycle lengths occurring in a permutation's cycle decomposition.

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 $\tau \in S_n$. Interestingly, conjugates have the same cycle structure as the original permutation. 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.

Let G be a finite group. 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 $\mathsf{ord}(g) = |\langle g \rangle| = \min\{k \geq 1 \mid g^k = \mathsf{id}\}$. A group $G = \langle g \rangle = \{g^0, \ldots, g^{\mathsf{ord}(g)-1}\}$ generated by a single element g is called *cyclic*. In the following, we identify a group with the set of its elements and vice versa.

Let us describe more formally how cards or other objects are reordered by permutations. 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 $\pi(\Gamma) \coloneqq (\Gamma[\pi^{-1}(1)], \Gamma[\pi^{-1}(2)], \ldots, \Gamma[\pi^{-1}(n)])$. Intuitively, the object in position i is transported to position $\pi(i)$. A particular way to say this and to make the well-behavedness of this operation evident, is to define it as a group action.

Definition 6.1 (Group action, e.g., [DM96, Sect. 1.3])**.** Let X be a nonempty set, G a group, and $\varphi\colon G \times X \to X$ a function. For $g \in G, x \in X$, we write $g(x) := \varphi(g, x)$. Then, G *acts* on X via φ, or G is a *group action* on X (leaving φ implicit), if

- $\mathsf{id}(x) = x$ for all $x \in X$, where id denotes the neutral element in G,
- $(g \circ h)(x) = g(h(x))$ for all $x \in X$ and all $g, h \in G$.

It is easy to see that every permutation group acts on reorderable objects such as card sequences in this way, by applying the permutation as defined above. Given these prerequisites we are able to define more formally what a card-based protocol is.

6.3. Formal Card-based Protocols

We introduce a modified version of the model for card-based computations as introduced by Mizuki and Shizuya [MS14a; MS17]. We will later discuss variants to the model in Sections 6.8 and 6.9. We will refer to them as *formal protocols* or *Mizuki–Shizuya protocols* to distinguish them from two-player protocols we introduce in Chapter 12. This is because their notion "abstracts away" the players conducting the protocol (a fact that we will discuss in Chapter 12).

We define a *(Mizuki–Shizuya) protocol* as a 5-tuple $(\mathcal{D}, U, H, Q, A)$, where $\mathcal{D}$ is a deck, U is a set of same-length *input sequences* of face-down cards from $\mathcal{D}$ all sharing the same visible sequence, H is a (possibly empty) sequence of additional cards from $\mathcal{D}$,

which we call *helping cards*,[2] and Q is a set of states with two distinguished states q_0 and q_{fin}, being the *initial* and the *final state*. Finally, A is a *(partial) action function*

$$A\colon (Q \setminus \{q_{\text{fin}}\}) \times \mathsf{Vis}_\ell^{\mathcal{D}} \to Q \times \mathsf{Action}_\ell,$$

which specifies the state to be entered and the action to be done next, dependent on the current state $q \in Q \setminus \{q_{\text{fin}}\}$ and the current visible sequence $v \in \mathsf{Vis}_\ell^{\mathcal{D}}$. Here, ℓ is the total length of the card sequences to be acted upon, namely the unique length of the input sequences plus the length of the helping card sequence[3]. Moreover, Action_ℓ is the set of the following formal actions to be performed on a sequence $\Gamma = (c_1, \ldots, c_\ell) \in \mathsf{Seq}_\ell^{\mathcal{D}}$:

- (perm, π), for a permutation $\pi \in S_\ell$, permutes Γ according to π. Formally, this yields the sequence $\pi(\Gamma) = (\Gamma[\pi^{-1}(1)], \ldots, \Gamma[\pi^{-1}(\ell)])$.
- (turn, T), for a set $T \subseteq \{1, \ldots, \ell\}$, flips the cards at positions specified by the *turn set* T. Formally, for a card $c = {}^{a}\!/\!_{b}$ we define $\mathsf{swap}(c) \coloneqq {}^{b}\!/\!_{a}$ and transform Γ into $\mathsf{turn}_T(\Gamma)$, where $\mathsf{turn}_T(\Gamma)[i] \coloneqq \mathsf{swap}(\Gamma[i])$ if $i \in T$, and $\mathsf{turn}_T(\Gamma)[i] \coloneqq \Gamma[i]$, otherwise.
- $(\mathsf{shuffle}, \Pi, \mathcal{F})$, for a permutation set $\Pi \subseteq S_\ell$ and a probability distribution $\mathcal{F}$ on Π, draws a random permutation $\pi \in \Pi$ according to $\mathcal{F}$ and obliviously applies it to Γ.[4] Note that security will crucially rely on the assumption that nothing about π is learned by any observers at this step, except what can be learned from $\mathcal{F}$ and the visible sequences $\mathsf{vis}(\Gamma)$ and $\mathsf{vis}(\pi(\Gamma))$. If $\mathcal{F}$ is the uniform distribution on Π, we may omit it and write $(\mathsf{shuffle}, \Pi)$.
- $(\mathsf{rflip}, \Phi, \mathcal{G})$, for a probability distribution $\mathcal{G}$ on $2^{\{1,\ldots,\ell\}}$ with support $\Phi \subseteq 2^{\{1,\ldots,\ell\}}$, draws a set T from Φ according to $\mathcal{G}$, and turns the cards at the positions in T. The sequence afterwards is $\mathsf{turn}_T(\Gamma)$. This is meant as a probabilistic version of the turn action. (We will discuss in Section 6.8 a more general variant of this action.)
- $(\mathsf{result}, p_1, \ldots, p_r)$, for a list of distinct positions $p_1, \ldots, p_r \in \{1, \ldots, \ell\}$, halts the protocol and specifies that $O = (\Gamma[p_1], \ldots, \Gamma[p_r])$ is the *output* of the protocol. When this command occurs, we require the cards at the respective positions to be face-down, and the first component of A's output (the next state to be entered) to be the final state q_{fin}. Moreover, we require that q_{fin} is only entered in company with an action of this type.

[2] In introducing the helping card as a separate entry in the tuple, we deviate from the literature, to make it more clear which cards encode inputs.

[3] We require that the deck of a protocol does not contain any unnecessary cards, i.e., that $\mathcal{D}$ is the union of all input sequences (interpreted as multisets), on which we apply the disjoint union of the helping card sequence (interpreted as multiset). In the most common case that all input sequences use the same cards, we have $\ell = |\mathcal{D}|$.

[4] Note that we usually require that the support of $\mathcal{F}$ is Π, i.e., that no elements of Π are assigned a zero probability by $\mathcal{F}$. Relaxing this allows us to encode in the protocol description that some permutations (those with a probability of zero) may be done without impeding security and correctness, which honest players however will not perform. We will come back to this in Chapter 12.

- (restart), transforms the current sequence into the original start sequence. When this command occurs, we require the first component of A's output to be the initial state q_0. (This allows protocols to start over when they hit an "unlucky" case. In real implementations this might require the players to reenter their inputs, unless they have been securely copied beforehand.)

An *execution* or *run* of a protocol $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ proceeds as follows: it starts in state q_0 and with a sequence $\Gamma_0 = \Gamma_{\text{in}} \,\|\, H$, where $\Gamma_{\text{in}} \in U$ is the part of the sequence encoding the input and H is the sequence of helping cards. The execution then proceeds in steps, where in each step of the protocol, the output of the action function (given the current state and the current visible sequence as input) determines the next state to be entered and the next action to be performed. The action is then applied as described above. The protocol terminates if it reaches state q_{fin}.

A (finite) *sequence trace* of $\mathcal{P}$ is a tuple $(\Gamma_0, \Gamma_1, \ldots, \Gamma_t)$ of sequences, such that $t \in \mathbb{N}$, $\Gamma_0 = \Gamma_{\text{in}} \,\|\, H$ as above, and Γ_{i+1} arises from Γ_i by the action specified via the action function in a protocol execution, for $i = 0, \ldots, t-1$.[5] We call $(\mathsf{vis}(\Gamma_0), \mathsf{vis}(\Gamma_1), \ldots, \mathsf{vis}(\Gamma_t))$ a *visible sequence trace* of $\mathcal{P}$. Additionally, the *permutation trace* $\mathcal{T}$ of $\mathcal{P}$ contains all randomly chosen permutations in the shuffle steps of the protocol execution, in their order of appearance. The execution yields an *execution trace* $(I, O, \mathcal{T}, V)$, containing input, output, permutation trace and the visible sequence trace V. The output of non-terminating protocols is $O = \perp$.

Running Time of a Protocol. A protocol is called *finite-runtime*[6] if there is a fixed, a priori bound on the number of steps, and, in contrast, *Las Vegas*, if it terminates almost surely (i.e., with probability 1) and in a number of steps that is *only expectedly* finite. Moreover, we call a Las Vegas protocol *restart-free* if it does not use any restart actions. These notions are important, as the running time of a protocol is central for its practicability.

On the Verifiability of Actions. Implicit in the action function domain $(Q \setminus \{q_{\text{fin}}\}) \times \mathsf{Vis}_\ell^{\mathcal{D}}$ is that all actions depend only on public information, not on players' private inputs or on what they did before, which would not be directly verifiable. While usually assuming an honest-but-curious setting for card-based cryptography, we think this design choice is important, as otherwise one would overstretch this assumption by allowing a player to easily deviate from the protocol without any chance of detection. The more refined active security of the respective actions will later be discussed in Chapter 12.

6.4. Encoding Bits with Cards

In the introduction, we described how two cards with distinct symbols ♡ and ♣, resp. encode a bit via the following rule: if they are in order ♣♡, they represent a 0, and if

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

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

they are in order ♡♣, they represent a 1. Formally, we introduce a possibly randomized, injective *encoding function* $[\![\cdot]\!]$ that maps bits to (same-length) card sequences. For example, we can define a function $[\![\cdot]\!]_{2c}$ (for 2-*c*olor decks) that maps 0 to ($^?/_{\clubsuit}$, $^?/_{\heartsuit}$) and 1 to ($^?/_{\heartsuit}$, $^?/_{\clubsuit}$). We denote the corresponding *decoding* map by $\mathsf{val}_{[\![\cdot]\!]}$ or val_x, if x is a subscript given to $[\![\cdot]\!]_x$. We will also call this function a *valuation*, as it gives an interpretation to a given card sequence. In this sense, val_{2c} maps the card sequence (c_1, c_2) to 0 if $\mathsf{symb}(c_1, c_2) = (\clubsuit, \heartsuit)$, and to 1 if $\mathsf{symb}(c_1, c_2) = (\heartsuit, \clubsuit)$. If we want to make the randomness r (which we can also see as a choice) explicit in a randomized encoding, we write $[\![\cdot; r]\!]$. Then, $(c_1, \ldots, c_n)$ *encodes a bit* $b \in \{0, 1\}$ *w.r.t. encoding* $[\![\cdot]\!]$ *and randomness* r, if $(c_1, \ldots, c_n) = [\![b; r]\!]$.

Note that while above's encoding is fine for two-color decks, we would like it to be general enough to work with arbitrary decks. For this, we assume a linear order $<$ on the deck alphabet Σ and also write $c_1 < c_2$ if $\mathsf{symb}(c_1) < \mathsf{symb}(c_2)$ for cards c_1, c_2 of deck alphabet Σ. Let c_1, c_2 be two such cards with $c_1 < c_2$. Then, as in [NR99], we define an encoding $[\![\cdot]\!]_>$ via

$$[\![b; \{c_1, c_2\}]\!]_> \coloneqq \begin{cases} (c_1, c_2), & \text{if } b = 0, \\ (c_2, c_1), & \text{if } b = 1. \end{cases}$$

where $r = \{c_1, c_2\}$ represents[7] the choice or randomness that one may have with this encoding, here the used cards. We call these cards the *basis* of the encoding. In the case that the deck does not allow any choice (as in the two-color deck case), we call the encoding *deterministic* and do not specify the basis explicitly.

The respective valuation function is then

$$\mathsf{val}_>(c_1, c_2) \coloneqq \begin{cases} 1, & \text{if } c_1 > c_2, \\ 0, & \text{otherwise.} \end{cases}$$

In the case of c_1 and c_2 having the same symbol $s \in \Sigma$, this is undefined and may be set to an error symbol $\perp_s$. Note that if one sets $\clubsuit < \heartsuit$, we recover the behavior of val_{2c} above. Unless explicitly stated, $[\![\cdot]\!]_>$ is the *standard encoding* throughout the thesis and we will just write that (c_1, c_2) *encodes* $b \in \{0, 1\}$, or that (c_1, c_2) *is a commitment to* b.

We extend any encoding $[\![\cdot]\!]$ to bit strings $b \in \{0, 1\}^k$ (and randomness vectors $\vec{r}$ of length k) via $[\![b; \vec{r}]\!] \coloneqq [\![b[1]; \vec{r}[1]]\!] \,\|\, \cdots \,\|\, [\![b[k]; \vec{r}[k]]\!]$ and any corresponding valuation function val to bit sequences via

$$\mathsf{val}(c_1, \ldots, c_{nk}) \coloneqq (\mathsf{val}(c_1, \ldots, c_n), \ldots, \mathsf{val}(c_{n(k-1)+1}, \ldots, c_{nk})) \in \{0, 1\}^k,$$

where n is the output length of $[\![\cdot]\!]$. Then we say that a card sequence Γ of length nk *encodes a bit string* $b \in \{0, 1\}^k$ (w.r.t. $[\![\cdot]\!]$), if $\mathsf{val}(\Gamma) = b$. Similarly, a card sequence Γ of length $2k$ is a *commitment to* $b \in \{0, 1\}^k$, if $\mathsf{val}_>(\Gamma) = b$.

[7] Note that, we avoid specifying what the randomness is formally, as it might have a natural interpretation, e.g., as two-element card sets, or as rotations represented by $\mathbb{Z}/5\mathbb{Z}$, as in the case of the output of den Boers five-card trick [dB90], in which case we prefer to use this natural representation.

6.5. Protocols Computing Boolean Functions

As described above, a protocol $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ takes as input a card sequence from U, and outputs a card sequence as specified by its result action. We would like to define what it means for a protocol to compute a Boolean function, using the encoding rules as discussed in the previous section.

We say that a protocol $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ *computes a function* $f\colon \{0,1\}^k \to \{0,1\}^m$ w.r.t. input encoding $[\![\cdot]\!]_\mathrm{i}$ and output encoding $[\![\cdot]\!]_\mathrm{o}$, if the following holds:

- $U = [\![\{0,1\}^k]\!]_\mathrm{i}$, i.e., U is the image of the domain of f under input encoding.
- Correctness: A protocol run starting with a sequence encoding $b \in \{0,1\}^k$ w.r.t. $[\![\cdot]\!]_\mathrm{i}$ terminates almost surely with an output encoding $f(b)$ w.r.t. $[\![\cdot]\!]_\mathrm{o}$.

We say that a protocol is *in committed format* if it computes a function w.r.t. the standard input and output encoding $[\![\cdot]\!]_>$. This is the usual and natural commitment format in the literature, which can be again used as inputs for other protocols due to this standardization. We will discuss this choice of encoding below.

Note that there is an important distinction to make, and which is unlike most protocols in the wider cryptographic literature: Card-based protocols can be seen as protocols, which start with commitments to the inputs, and end with a commitment to the output. We also call this *delegated computation* or *commitment-in/commitment-out* paradigm, as it allows to compute on the values without any player being aware of inputs or learning the output, and gives strong composition guarantees. (Additionally, we make sure that opening the output commitment does not give away anything about b that is not already obvious from $f(b)$. We will do this more formally in the next section, and also in Chapter 12 on active security, where we include a discussion on protocols requiring input awareness.)

While the thesis focuses almost entirely on committed-format protocols, we wanted our definitions to encompass also two important so-called "non-committed format" AND protocols from the literature, namely [dB90; MKS12], which deviate from committed format protocols in using a non-standard output encoding. We will present them in Protocol 6.2 and Figure 7.12. Previously, these protocols have been interpreted as protocols necessarily revealing their output to learn their result, but we feel that by introducing non-standard encodings (and possibly finding conversion protocols to standard format), we can analyze all these protocols in a common framework.

We call protocols computing functions $f_\wedge\colon \{0,1\}^2 \to \{0,1\}$ with $(a,b) \mapsto a \wedge b$, and $f_\mathrm{copy}\colon \{0,1\} \to \{0,1\}^n$ with $a \mapsto a^n = (a, \ldots, a)$, $n \geq 2$, *AND* and *n-COPY* (or just *COPY*) *protocols*, respectively. In the following Example 6.1, we describe the committed format six-card AND protocol of Mizuki and Sone [MS09] (cf. the introduction for an informal description), using the Mizuki–Shizuya formalism above. This is meant as an illustration of the definitions, and not as a convenient way to write down card-based protocols. Hence, in Protocol 6.1 we also give a pseudocode version for comparison, and we will switch to pseudocode in all that follows.

Example 6.1. The six-card AND protocol of Mizuki and Sone [MS09], with deck $\mathcal{D} = [3 \cdot \heartsuit, 3 \cdot \clubsuit]$, input set $U = \{[\![b]\!]_> : b \in \{0,1\}^2\}$ and helping card sequence $H = (?/\clubsuit, ?/\heartsuit)$, is defined as $\mathcal{P}^{\mathrm{AND}} = (\mathcal{D}, U, H, \{q_0, q_1, q_2, q_{\mathrm{fin}}\}, A)$, with A as follows:

1. $A(q_0, (?,?,?,?,?,?)) = (q_1, (\mathsf{shuffle}, \langle (1\ 4)(2\ 5)(3\ 6)\rangle))$
2. $A(q_1, (?,?,?,?,?,?)) = (q_2, (\mathsf{turn}, \{1,2\}))$, i.e., turn the first two cards.
3. $A(q_2, (\heartsuit, \clubsuit, ?,?,?,?)) = (q_{\mathrm{fin}}, (\mathsf{result}, 3, 4))$, and
   $A(q_2, (\clubsuit, \heartsuit, ?,?,?,?)) = (q_{\mathrm{fin}}, (\mathsf{result}, 5, 6))$.

$(\mathsf{shuffle}, \langle (1\ 4)(2\ 5)(3\ 6)\rangle)$
$(\mathsf{turn}, \{1,2\})$
Let v be the current visible sequence
if $v = (\heartsuit, \clubsuit, ?,?,?,?)$ **then**
  $(\mathsf{result}, 3, 4)$
else if $v = (\clubsuit, \heartsuit, ?,?,?,?)$ **then**
  $(\mathsf{result}, 5, 6)$

Protocol 6.1. Pseudocode version of Example 6.1, a protocol to compute AND in committed format using deck $\mathcal{D} = [3 \cdot \heartsuit, 3 \cdot \clubsuit]$.

The five-card AND protocol of den Boer [dB90] is given in Protocol 6.2 and computes AND with respect to the following non-standard output encoding $[\![\cdot]\!]_{\heartsuit\heartsuit}$. For this, let $b \in \{0,1\}$ and $r \in \mathbb{Z}/5\mathbb{Z}$:

$$[\![b; r]\!]_{\heartsuit\heartsuit} := \begin{cases} (1\ 2\ 3\ 4\ 5)^r((?/\heartsuit, ?/\heartsuit, ?/\clubsuit, ?/\clubsuit, ?/\clubsuit)), & \text{if } b = 1, \\ (1\ 2\ 3\ 4\ 5)^r((?/\heartsuit, ?/\clubsuit, ?/\heartsuit, ?/\clubsuit, ?/\clubsuit)), & \text{otherwise.} \end{cases} \tag{6.1}$$

In other words, we choose a given sequence (candidates as above, the hearts are adjacent, iff $b = 1$) and then rotate by a random offset $r \in \mathbb{Z}/5\mathbb{Z}$. The respective valuation $\mathsf{val}_{\heartsuit\heartsuit}(c_1, \ldots, c_5)$ then decides whether the sequence $(c_1, \ldots, c_5)$ contains two (cyclically) adjacent hearts, and decodes to $b = 1$ in this case, or to 0 otherwise. We will later see that the protocol of Abe et al. [AHMS18] can be seen as continuation of this protocol to convert its output to the standard encoding. (This will be made precise in Section 7.5.)

Properties of Encodings and Rerandomization

One may ask what makes for the best encoding. In this section we would like to introduce important properties and argue for why the standard encoding is a worthy choice. For the discussion, let $[\![\cdot]\!]$ be an encoding function and let us consider the following list of desiderata:

1. There are (easy-to-use) protocols for AND, NOT and COPY which use $[\![\cdot]\!]$ for encoding inputs and outputs. (This is because AND, NOT and 2-COPY together is a complete set to allow computing any Boolean function.)

$(\mathsf{perm}, (3\ 5))$ $(\mathsf{shuffle}, \langle(1\ 2\ 3\ 4\ 5)\rangle)$ $(\mathsf{result}, 1, 2, 3, 4, 5)$

Protocol 6.2. The five-card trick of den Boer [dB90]: an AND protocol with deck $\mathcal{D} = [2 \cdot \heartsuit, 3 \cdot \clubsuit]$, input set $U = \{[\![b]\!]_> : b \in \{0,1\}^2\}$, and helping card sequence $H = (?\!/\!\clubsuit)$. The protocol uses a non-standard, randomized output encoding $[\![\cdot]\!]_{\heartsuit\heartsuit}$, given in (6.1).

2. $[\![\cdot]\!]$ should use the same set of cards for all values to be encoded. (In card-based protocols involving players, we have to hand them all cards necessary for encoding arbitrary inputs, anyway. Moreover, it is easier to guarantee security, if this property holds, for the same reasons as in the discussion on the previous item below.)

3. $[\![\cdot]\!]$ should use as few cards as possible.

4. $[\![\cdot]\!]$ should be either *deterministic* or *rerandomizable*. For the latter, we want the existence of a protocol that computes the identity function with the encoding used for inputs and outputs, but the output uses fresh randomness for the encoding. (Ideally, this protocol should not leak the randomness used in the input encoding, we will discuss this fact in the next section.)

While item 1 looks like a relatively pragmatic reason, it is crucial if inputs/outputs are to be used in general circuits. For example, the *single-card encoding* $[\![\cdot]\!]_\heartsuit$, which maps $1 \mapsto ?\!/\!\heartsuit$ and $0 \mapsto ?\!/\!\clubsuit$, violates this property, as Mizuki and Shizuya [MS14a] showed the impossibility of perfectly secure single-card COPY protocols. We believe their arguments can be extended to the impossibility of NOT and AND single-card encoding inputs and outputs. Moreover, there are no protocols in the literature which use the five-card encoding $[\![\cdot]\!]_{\heartsuit\heartsuit}$ for inputs.

Item 2 is clearly violated by the single-card encoding $[\![\cdot]\!]_\heartsuit$, but holds true for $[\![\cdot]\!]_{\heartsuit\heartsuit}$ and $[\![\cdot]\!]_>$ (the latter using a fixed choice of basis). Item 3 is a less strict requirement, but given that the single-card encoding does not fulfill the previous properties, we need at least two cards, which should be distinct for all values due to item 2. This identifies the standard encoding as optimal, and justifies its common use in card-based cryptography.

Concerning item 4, this holds for all three encodings discussed so far. While the single card encoding is deterministic, the five-card encoding is rerandomizable by applying a uniform random cut $(\mathsf{shuffle}, \langle(1\ 2\ 3\ 4\ 5)\rangle)$ to the cards. For the standard encoding, it depends on the randomness domain. If there is no choice, e.g., we restrict it to exactly the basis $\{?\!/\!\clubsuit, ?\!/\!\heartsuit\}$, then the encoding is deterministic. If we want to make the basis choices explicit in the encoding, we may write it as a superscript as in the following example: $[\![b; r]\!]_>^{\{\{1,2\},\{3,4\}\}}$ is an encoding of $b \in \{0,1\}$ in basis $r \in \{\{?\!/\!_1, ?\!/\!_2\}, \{?\!/\!_3, ?\!/\!_4\}\}$. Note that if there is more than one choice, we can rerandomize by randomly selecting a pair of cards from a sequence containing all permissible base choices and then applying

a base conversion protocol (as in [M16b]). We will discuss these protocols later in Chapter 8, but to prepare a discussion in the next section, let us point out here, that such a base conversion protocol leaks the previously used basis.

6.6. Security of Card-based Protocols

In [MS14a], the authors essentially define security as the stochastic independence of input sequence and what is visible during a protocol execution. More formally, they require that the random variable $\Gamma_0 = I$ with values in U specifying the input sequence, and V, which is the random variable of the visible sequence of an execution starting with input Γ_0, are stochastically independent. This definition is then applied to show the impossibility of single-card COPY protocols featuring this security notion.

While having relatively weak but meaningful notions strengthens impossibility results like this and the security notion is perfectly fine for these deterministic COPY protocols, we think that this definition falls short to really capture the security that is to be wished and that is offered by common card-based protocols. For example, it does not protect the output of a protocol, which is important, e.g., when computing randomized functions to be used in a larger circuit, where we do not want any intermediary results leaked. This can be accommodated by having the pair (I, O) being stochastically independent from V, if O is the random variable of the output sequence of a protocol run starting with input $\Gamma_0 = I$ and visible sequence trace $v = V$. (This implies that if the function to be computed is randomized, the used randomness should not be contained in V.)

However, this still fails to give a rigorous explanation on why non-committed format protocols, such as the five-card trick above, are secure, where there is some choice in the output encoding, which should not leak anything upon opening the output cards or using them in subsequent protocols. Hence, this section aims to generalize the above security notion to the case of randomized encodings. For the special case of deterministic encodings, both notions to be discussed next are equivalent to (I, O) being stochastically independent from V. For a concise notation, we denote stochastic independence of random variables X and Y by $X \perp Y$. Let us start with the formal definition:

Definition 6.2 (Security)**.** Let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a protocol computing a function f w.r.t. input encoding $[\![\cdot]\!]_{\mathrm{i}}$ and output encoding $[\![\cdot]\!]_{\mathrm{o}}$. Let I be a random variable with values in U, V be a random variable denoting the visible sequence trace of a run on input $\Gamma_0 = I$, and O the random variable of its output. Moreover, let R_{i} and R_{o} be the random variables of the randomness used in encoding I and O, respectively. Then $\mathcal{P}$ is *secure* if

$$R_{\mathrm{i}} \perp \mathsf{val}_{\mathrm{i}}(I) \text{ implies that } (\mathsf{val}_{\mathrm{i}}(I), \mathsf{val}_{\mathrm{o}}(O)) \perp (V, R_{\mathrm{o}}, R_{\mathrm{i}}).$$

In other words, given that R_{i} does not give anything away about $\mathsf{val}_{\mathrm{i}}(I)$, then, from learning the public visible sequence trace V and the randomness $R = (R_{\mathrm{i}}, R_{\mathrm{o}})$ used in the encodings, one does not learn anything about inputs and output values encoded in I and O, respectively, as well as about any (non-trivial) relations between them.

Additionally, if we are interested in protecting not only the bits but also the full card sequences used for in- and output, we say that $\mathcal{P}$ is *full-sequence secure* if it is secure and V is independent of $(R_\mathrm{i}, R_\mathrm{o})$, i.e., by looking at the public information, one does not learn anything about the randomness used in the input and output encodings.

In terms of probabilities, security means that if $\Pr[R_\mathrm{i} = r_\mathrm{i} \,|\, \mathsf{val}_\mathrm{i}(I) = i] = \Pr[R_\mathrm{i} = r_\mathrm{i}]$, then

$$\Pr[\mathsf{val}_\mathrm{i}(I) = i, \mathsf{val}_\mathrm{o}(O) = o \,|\, V = v, R_\mathrm{i} = r_\mathrm{i}, R_\mathrm{o} = r_\mathrm{o}] = \Pr[\mathsf{val}_\mathrm{i}(I) = i, \mathsf{val}_\mathrm{o}(O) = o],$$

where $r_\mathrm{i}, r_\mathrm{o}$ is the randomness used in I and O, respectively.

We can guarantee that $R_\mathrm{i} \perp \mathsf{val}_\mathrm{i}(I)$ in the very beginning of a computation by either only allowing a deterministic encoding, or by prepending a re-randomization protocol. In the case where there is a correlation between R_i and $\mathsf{val}_\mathrm{i}(I)$, e.g., because the randomness has been chosen adversarially, we want the re-randomization protocol to be full-sequence secure in order to hide R_i, which would leak information about $\mathsf{val}_\mathrm{i}(I)$[8]. Then, security of the protocol guarantees that $R_\mathrm{o} \perp \mathsf{val}_\mathrm{o}(O)$, which will be the input to following protocols, then already satisfying the prerequisite in the security definition.

Security of Common AND Protocols. Let us discuss why both the committed-format six-card AND protocol from [MS09] and the five-card trick of [dB90] are secure. To see security of the first protocol, note that its input and output is deterministic as it uses the standard encoding and is in the two-color deck case. Hence, R_i and R_o do not have any entropy and it suffices to show that $(\mathsf{val}_\mathrm{i}(I), \mathsf{val}_\mathrm{o}(O))$ is stochastically independent of V. Up until the turn, V contains only trivial visible sequences, so the only non-trivial observation to be made is the two symbols of the turned cards. As discussed in the introduction, the shuffle will ensure that the turned cards encode $[\![\mathsf{val}_\mathrm{i}(I)[1] \oplus m]\!]_>$, i.e., the first bit masked by a uniformly random $m \in \{0, 1\}$. Hence, $\mathsf{val}_\mathrm{i}(I)[1] \oplus m$, and consequently V, is independent of $\mathsf{val}_\mathrm{i}(I)$, and as AND is a deterministic function, also of $(\mathsf{val}_\mathrm{i}(I), \mathsf{val}_\mathrm{o}(O))$.

Let us turn to the security of den Boers protocol. While the input encoding is deterministic, R_o has values in $\mathbb{Z}/5\mathbb{Z}$ specifying the rotation of the candidate card sequence, as defined in (6.1). Because the protocol does not turn any cards, V is trivial and it remains to show that $(\mathsf{val}_\mathrm{i}(I), \mathsf{val}_\mathrm{o}(O))$ is stochastically independent of R_o. But this is certainly true, as R_o contains fresh randomness generated by the uniform cut, which by definition does not depend on any content of the cards shuffled.

Discussion. While the security definition may look a little complex at first sight, in Chapter 7 we will define an enriched way to speak about protocols, compressing all possible runs into a simple diagram, from which one can systematically check the security property by checking a local criterion for all turn and result actions. This will allow for

[8] As discussed above, this is not the case in the re-randomization protocol for the standard encoding $[\![\cdot]\!]_>$, cf. the previous section. However, we can just require all immediate inputs to have a deterministic encoding, which is the case in all protocols in the literature we know of.

even simpler, more systematic security proofs. Moreover, in Chapter 12 we will extend the discussion to the setting where adversarial players may try to influence the shuffling process or know how they took part in performing a shuffle operation.

In the above security definition, we have just one input string, where each bit might depend on the others. When the input is provided by players, e.g., each player has one bit of input, each of them has partial knowledge on the input sequence. Ideally, the security definition should also imply that, given this partial knowledge, one should not be able to learn any additional information from $V, R_\mathrm{o}, R_\mathrm{i}$ about the other players input values or the output value, relative to their prior knowledge. Note that our definition given above implies that, even given this partial knowledge on $\mathsf{val}_\mathrm{i}(I)$ and R_i, then $\mathsf{val}_\mathrm{i}(I)$ and $(V, R_\mathrm{o}, R_\mathrm{i})$ are still independent conditioned on the partial knowledge (given that $\mathsf{val}_\mathrm{i}(I)$ and R_i are independent).

While in this section, we take extra care to define the strongest security notion that we could conceive, but which is still attainable, we want to stress that our impossibility results in Chapter 9 are based on *much weaker* notions of security, namely that at the end of the protocol, all outputs in the image of the function to be computed shall still be possible (have probability greater that zero). We call this output-possibilistic security and refer to Definition 9.1 for a formal account.

6.7. Restricted Shuffling

The formal model as defined by Mizuki and Shizuya [MS14a] allows shuffling operations with arbitrary permutation sets and an arbitrary real-valued probability distribution on them. This is arguably a very strong assumption, allowing for similarly strong impossibility results. However, we will later present a protocol in Section 8.3 for computing arbitrary n-ary Boolean functions with $2n$ cards, using a shuffle that – while allowed in the formalism by [MS14a] – is of questionable practicality. We could not conceive of a way to perform it with actual people and actual cards such that the players do not learn anything about the permutation that was done in the shuffle. It somehow does all the "hard work" in computing the function in one lucky case to be hit, while all the other visible sequences after the following turn will force the protocol to restart. One may argue that at least for possibility results, one should aim for less.

In this thesis, we identify a natural subclass of these general shuffles, which we dubbed "practicable shuffles" in the introduction. While this name is suggestive, it is also debateable due to its subjective nature. Hence, we prefer the more exact definition, namely of shuffles that are

- *uniform*, i.e., where the probability distribution on the permutations in the shuffle is the uniform distribution, and

- *closed*, meaning that the permutation set really is a subgroup of S_ℓ. (The name originates from the set being closed under the composition operation.)

Together, the properties insure that when shuffling twice in this way, the probability distribution of resulting card sequences is the same as when shuffling once, as, e.g., closedness implies $\Pi^2 \coloneqq \{\pi_1 \circ \pi_2 \mid \pi_1, \pi_2 \in \Pi\} = \Pi$.

Note that a good real-world implementation of the shuffles is important to back up the claim that the commitments used are binding. When viewing single face-down cards as commitments to their symbols, the bindingness is immediate, if there is no possibility of the players to exchange the card by a new one. However, when viewing multiple face-down cards as a commitment to a bit value, the perfect *bindingness* of the commitment (as assumed in the card-based cryptographic literature), i.e., that no player can for example invert the respective bit before it is revealed, depends mostly on the secure implementation of the shuffle actions. For this, we have to ensure that in these steps nobody can swap the cards of a bit if it is not an allowed action.

The proposed *uniform closed shuffles* can be easily implemented in the honest-but-curious setting by the parties taking turns in performing a random permutation from the specified permutation set, while the other parties are not looking. While this is usually sufficient for didactic settings, we will describe an *actively secure* implementation of uniform closed shuffles in Chapter 12, enforcing players to only perform permutations from the allowed set. This will provide evidence to the assumed bindingness in card-based protocols.

Also, note that uniform closed shuffles *suffice to compute any function*, as almost all existing Mizuki–Shizuya protocols, e.g., [CK94; dB90; ICM15; MAS13; M16a; MKS12; MS09; MS14b; NHMS15; NR98; S01; AHMS18] and the protocols in Figures 8.5 and 8.6, use only these. This list contains protocols for AND, NOT and COPY, hence allowing arbitrary circuits. (More general shuffles appear in [CHL13; NNH$^+$18] and Sections 8.1 and 8.2 only for the purpose of using less cards.[9] In this sense, our results on lower bounds on the number of cards will show that restricting to these shuffles really does affect how many cards are needed to compute AND or COPY.)

Other authors suggest to further restrict the class to shuffles of a few basic types, namely random cuts, random bisection cuts and unconstrained (S_k-)shuffles, and their generalizations to piles (i.e., instead of shuffling cards, one shuffles piles that are somehow forced to stay together). Arguably, random cuts are the most basic type of shuffling, and we will defend this view in Chapter 12. Moreover, Mizuki and Sone [MS09] and Ueda et al. [UNH$^+$16] argue for nice ways to implement the random bisection cut. While we feel that it is natural to consider uniform and/or closed shuffles for meaningful impossibility results, we agree to prefer these even more restricted shuffles to devise really practicable protocols.

Formally, a shuffle is a *random cut* if its permutation set is a group, generated by an element π *which is a single cycle* $(x_1\ x_2\ \ldots\ x_l)$. A shuffle is a *random bisection cut* if its permutation set is generated by a permutation π which is the composition of

[9] Note that non-uniform and/or non-closed shuffles used therein have been implemented in [NHMS16] using sliding cover boxes, but such an approach requires extra tools and the security that we achieve using our transformation in Chapter 12 is not achieved against attackers that reorder the boxes in an illicit way in their implementation.

pairwise disjoint cycles *of length* 2. Finally, an S_k*-shuffle* for $k \leq \ell$ is a shuffle with permutation set S_k. Pile shuffles will be further explored in Chapter 12.

Algebraic Properties of Closed Shuffles

As the shuffle set Π of a *closed* shuffle is a subgroup of S_ℓ, it also defines a group action on the card sequences, allowing us to exploit their algebraic structure. This structure is captured in the orbits associated with the group action of Π.

For this, let G be a group acting on a set X, as in Definition 6.1. Then we define the *orbit of an* $x \in X$ as $G(x) \coloneqq \{g(x) \colon g \in G\}$, i.e., all elements in X that are reachable from x via some $g \in G$. Note that orbits $G(x), G(y)$ of $x, y \in X$ are either disjoint or equal. Hence the orbits form a partition of X, called the *orbit partition* of X through G. (A partition of X is a set of disjoint subsets from X, such that their union is X.) In our setting, $G = \Pi \subseteq S_\ell$ is a permutation group of a shuffle and $X = \mathsf{Seq}_\ell^{\mathcal{D}}$ is the set of sequences on a deck $\mathcal{D}$, and Π acts on X naturally via reordering the cards according to $\pi \in \Pi$ as described above. We will use orbit partitions to analyze protocols using closed shuffles and prove lower bounds on the number of cards in such protocols in Sections 9.7 and 9.10. Moreover, we give a criterion for the secure application of so-called sort protocols in terms of orbit partitions, cf. Definition 8.1. For a shuffle with permutation set Π we also say that the orbit partition of Π is the orbit partition of the shuffle.

As an example, consider the orbit partition on the sequences $\mathsf{Seq}^{\mathcal{D}}$ of deck $\mathcal{D} = [\clubsuit, \clubsuit, \heartsuit, \heartsuit]$ that $\langle \pi = (1\ 2)(3\ 4) \rangle$ generates (written as strings):

$$\{\{\heartsuit\heartsuit\clubsuit\clubsuit\}, \{\heartsuit\clubsuit\heartsuit\clubsuit, \clubsuit\heartsuit\clubsuit\heartsuit\}, \{\heartsuit\clubsuit\clubsuit\heartsuit, \clubsuit\heartsuit\heartsuit\clubsuit\}, \{\clubsuit\clubsuit\heartsuit\heartsuit\}\}.$$

That is, e.g., because π maps $\heartsuit\clubsuit\heartsuit\clubsuit \mapsto \clubsuit\heartsuit\clubsuit\heartsuit$ and vice versa, but leaves $\heartsuit\heartsuit\clubsuit\clubsuit$ and $\clubsuit\clubsuit\heartsuit\heartsuit$ fixed. Sometimes, it is useful to speak of all permutations that leave a given sequence fixed, e.g., in that they only exchange cards with indistinguishable symbols. Obviously, the set of these permutations includes the identity permutation, and in fact it forms a group. Formally, the *stabilizer subgroup* $\mathsf{Stab}_x(G)$ of a group G with respect to an $x \in X$, is defined as $\mathsf{Stab}_x(G) = \{g \in G : g(x) = x\}$, i.e., the group of all $g \in G$ that *fix* x, meaning that $g(x) = x$.

We will later make use of the following interesting fact about orbit partitions and stabilizer subgroups:

Lemma 6.1 (Orbit-Stabilizer theorem)**.** *Let G be a group, X a nonempty set and let G act on X as specified above. Then, $|G(x)| = |G|/|\mathsf{Stab}_x(G)|$ for all $x \in X$.*

Proof. The simple proof is given e.g., in [DM96, Thm. 1.4A]. □

6.8. Variants of the Computational Model

The computational model defined in Section 6.3 includes an rflip action that is meant as a randomized version of the turn action. Note however that in contrast to shuffling, this does not introduce uncertainty, as the turned cards are publicly observable. While

having public randomness in the formalism seems useful (albeit we are not aware of any protocols in the literature that make use of it), we think that in this case, it is natural to choose more generally which action to perform (instead of only which set of cards to turn) based on, e.g., whether a coin flipped by the players ends up heads or tails.

For this introduce an additional class of actions in protocols, that produces public randomness: The action $\mathsf{rand}(p_1, \ldots, p_i)$ for real numbers $p_1, \ldots, p_i \in (0, 1)$ summing to 1 appends a value $x \in \{0, 1, \ldots, i\}$ to the visible sequence trace[10] with $\Pr[x = j] = p_j$.

Note that public randomness needs to be handled with care when assuming our security notion above. An example for this would be a protocol that starts with a commitment to a bit and outputs either this bit unchanged with probability $1/3$ or its negation with probability $2/3$. This can be implemented by a $\mathsf{rand}(1/3, 2/3)$ action and, depending on the outcome, exchanging the two cards or leaving them as-is, followed by a result action. (This is also possible with a shuffle, but requires a non-uniform probability distribution on the permutations, something we are inclined to prohibit as discussed in Section 6.7.) However, this protocol is *not secure* since the result of the random experiment is part of the visible sequence trace (it is *public*) which introduces a non-trivial relationship between output and visible sequence trace precluding independence.

Let us turn to a simplification of the model. Shuffling with face-up cards or with distinct backs, i.e., with the current visible sequence v being non-trivial, may result in several visible sequences v', depending on how the cards are permuted. It is easy to see that such a *branching shuffle* can be simulated with a rand action, determining which v' is obtained, followed by a shuffle action restricted to those permutations that transform v into v'. In the following lemma and the following discussion, we argue that this still works for uniform and/or closed shuffles, i.e., if the original branching shuffle was closed (uniform), the restricted shuffles can be implemented using closed (uniform) shuffles as well, possibly accompanied by a deterministic permutation action.

Lemma 6.2. *Let Π be any closed permutation set and v and v' visible sequences. Assume $\Pi_{v \mapsto v'} := \{\pi \in \Pi \colon \pi(v) = v'\}$ is non-empty. Then for any $\pi_{v \mapsto v'} \in \Pi_{v \mapsto v'}$, the set $\Pi_v := \Pi_{v \mapsto v'} \circ \pi^{-1}_{v \mapsto v'}$ is closed.*

In particular, we can implement $(\mathsf{shuffle}, \Pi_{v \mapsto v'})$ using $(\mathsf{perm}, \pi_{v \mapsto v'})$ followed by the closed shuffle $(\mathsf{shuffle}, \Pi_v)$.

Proof. Let $x = \varphi_1 \circ \pi^{-1}_{v \mapsto v'}$ and $y = \varphi_2 \circ \pi^{-1}_{v \mapsto v'}$ be two elements of Π_v, in particular $\varphi_1, \varphi_2 \in \Pi_{v \mapsto v'}$. We have $(\varphi_1 \circ \pi^{-1}_{v \mapsto v'} \circ \varphi_2)(v) = v'$, so $\varphi_1 \circ \pi^{-1}_{v \mapsto v'} \circ \varphi_2 \in \Pi_{v \mapsto v'}$. This implies:

$$x \circ y = \varphi_1 \circ \pi^{-1}_{v \mapsto v'} \circ \varphi_2 \circ \pi^{-1}_{v \mapsto v'} \in \Pi_{v \mapsto v'} \circ \pi^{-1}_{v \mapsto v'} = \Pi_v. \qquad \square$$

For uniform shuffles, the analogous claim is even easier, as a uniformly random variable conditioned to be contained in some subset is still uniform on that subset. For uniform *and* closed shuffles these arguments can be combined.

[10] We refrain from extending the definition of a visible sequence trace ("view") to contain $x \in \{0, 1, \ldots, i\}$, as the required adaptations are only technical and straightforward.

From the discussion above, we can derive the following corollary, for general protocols and for protocols restricted to uniform and/or closed shuffles.

Corollary 6.1 (We can assume all cards are face down)**.** *It suffices to look at protocols which start with face-down cards and which, after turning any card, directly turns it face-down again.*

Note, however, that branching shuffles are useful, if one wants to restrict shuffles in other ways. For example, if we are interested in protocols which only use random cuts, the analogous statement of Lemma 6.2 is false, because the resulting Π_v will in general not again be a random cut, even if Π is a random cut. Because of this, we use cards with different backs in Figures 7.4 and 12.2 later on. To give a counterexample, let the back alphabet $B = \{?, \#\}$ and assume one is given a card sequence of even length with alternating backs, e.g., with visible sequence $(?, \#, ?, \#, ?, \#)$. Then there are exactly two configurations this can end up, if we apply $(\mathsf{shuffle}, \Pi = \langle(1\ 2\ 3\ 4\ 5\ 6)\rangle)$, namely $(?, \#, ?, \#, ?, \#)$ and $(\#, ?, \#, ?\#, ?)$. The set of all permutations compatible with the first visible sequence is $\langle\pi\rangle$, with $\pi = (1\ 2\ 3\ 4\ 5\ 6)^2 = (1\ 3\ 5)(2\ 4\ 6)$. As π is not a single cycle, shuffling with this set is no longer a random cut. Hence, we cannot implement this shuffle in the way above using only random cuts and only '?' as back symbols.

Finally, note that if one wants to restrict to random bisection cuts or more generally shuffles with groups of order 2, branching shuffles also do not help, because in the case of branching, one completely learns the chosen permutation from the resulting visible sequence, and can then implement it with just a rand operation, followed by perm.

6.9. Discussion

All definitions and efforts to pinning-down the right notions bring with them the danger of excluding other worthwhile options. Let us briefly discuss the choices that we have not covered.

Most notably, concerning the modeling of cards and the available actions, the above definitions ignore a possibility present in many common card games, namely to *rotate* the typical rectangular cards by 180 degrees, as pointed out in [MS14b, Sect. 4]. Hence, when actually running a card-based protocol with real cards, one either has to assume that both back and front of the cards are rotation-invariant, i.e., one cannot tell the orientation as it looks the same both ways, or if not, the back side has to tell the orientation to detect whether cards have been rotated. Otherwise, a player could input their cards upside-down to "mark" them and to subsequently learn the position of the other players' cards and their value. On the flip-side, [MS14b; SNN$^+$16] point out how one can exploit cards with rotationally symmetric backs to input single-card commitments via $\boxed{\heartsuit}$ or $\boxed{\clubsuit}$ depending on the input bit. Using this *encoding by card orientation*, the authors devise two-card XOR, three-card COPY and three-card AND protocols, the latter featuring a so-called "tornado shuffle". Shinagawa et al. [SMS$^+$15a] and Shinagawa and Mizuki [SM18] developed the idea further to allowing regular n-gonal cards to represent values in $\mathbb{Z}/n\mathbb{Z}$ and to compute on these values. Moreover, [MWS15,

Sect. 3.4] proposes a simple AND protocol with a single square-shaped, rotate-able card which one can partially reveal by folding half of it up.

Another route that one may go is to define the cards' *hidingness* as less perfect, as in the setting of *tamper-evident seals* introduced by Moran and Naor [MN10a]. Here, players may tear open envelopes or scratch off scratch-off-cards (corresponding to turning cards in our setting), but with the consequence that this is detectable, leading to an abort of the protocol. We assume that players can observe an adversary turning cards during execution, as card-based cryptographic protocols are typically performed with all players around a table. In this setting, card-based cryptography cannot really protect from illegitimate card turning, besides generic methods, such as compiling the circuit to be computed into a *private circuit*, as defined by Ishai, Sahai, and Wagner [ISW03]. Hence, this is excluded from the model. The perfect hidingness of card-based commitments is also based on the indistinguishability of the cards' backs and of cards using the same symbols. As in private circuits, Mizuki and Shizuya [MS14b] propose to strengthen this assumption by handling scuff marks on the cards via XOR-secret sharing the value in many bits.

When discussing the available actions in a protocol, one may introduce formal privatePerm or inputPerm operations, which allow for a player to permute the cards in a way unobserved by the other players, but restricted in its choice. This may be done, e.g., to implement shuffling (as argued above, this is at least as powerful as allowing uniform closed shuffles) or to input bits (requiring that players are aware of their inputs). We will cover these protocols in Chapter 12. In the same way, privateTurn operations might be of use, but only if players are modeled that can do actions dependent on private knowledge, not verifiable by other players. Hence, this is excluded in the definition given above. Moreover, one may conceive that players may provide inputs later in the protocol, as in reactive secure multiparty computation by a special inputCards action. In our exposition we focus on the case of secure function evaluation where all inputs are fixed in the beginning, as one can easily update this to the reactive case, if needed.

In [KKW+17, Sect. 4], the authors use a variation of the computational model that does not use state machine semantics as in Section 6.3, but possibly infinite, directed trees, where the actions are prescribed to the nodes of the tree, and the outgoing edges are chosen according to the observed visible sequence, similar to the diagrams that we will introduce in Chapter 7. Given a protocol defined in this way, one can eliminate the occurrences of rflip and rand actions by deterministic replacements.

Conclusion. In this chapter, we have given a unified presentation that combines many of the different notions and implicit intuitions scattered throughout the literature into one consistent framework. For this, we refined and strengthened what is to be understood by security in card-based protocols, which has been lacking in sufficient generality so far. The definitions are general enough to account for both committed and non-committed format protocols and for more flexible encoding as in the case of a standard deck, and will be further extended in Chapter 12 to also encompass protocols requiring input awareness, in that the players input their bits by performing a permutation.

Moreover, we identify uniform closed shuffling as a natural restriction to the more general shuffling allowed in the formalism, offering a conceptually simple implementation in the real world, e.g., by players taking turns in shuffling. We see this as a middle ground between too strong and too weak (im-)possibility results, which will be later backed up by a protocol using shuffles, for which no implementation has been proposed, yet. Additionally, this class allows to exploit the rich algebraic structure that orbit partitions of the corresponding shuffle groups induce – a fact that will be important in Chapter 9.

Finally, we propose variants and simplifications to the computational model, e.g., that shuffling with open cards or using different backs is unnecessary, unless one wants to restrict shuffling to specific classes, such as random cuts. In what follows, we will use this as a basis to give rigorous possibility and impossibility proofs for card-based protocols.

7. Diagrams for Secure Protocols

This chapter is mainly based on [KWH15; KKW+17] and introduces state diagrams, which are representations of protocols describing all possible runs simultaneously in a way that makes correctness and security of the protocol directly recognizable.

They have been formally introduced in [KWH15] and have since found widespread use in the literature, e.g., [FAN+17; MS17; AHMS18; MK18; MUH+18], where they are also called KWH diagrams or trees. In essence, they are labeled graphs, which specify for each step of a protocol run, which card sequences are possible at this point in time, and how likely they are, in terms of the probabilities for the inputs.

The way it is defined, a state diagram is not only a structured way to describe protocols, but also a direct witness for a main aspect of the security of the protocol, as protocols violating this security aspect do not even admit well-formed state diagrams. We will make this precise below. Moreover, correctness and the security w.r.t. Definition 6.2 are verifiable via a simple criterion to be checked at the leaf nodes of the diagram. What is more, state diagrams allow for structural insights and play a central role in our proofs for the impossibility of AND and COPY protocols for certain decks and requirements in Chapter 9.

Let us start with an example of how we represent states in such a diagram:

$\mathsf{symb}(\llbracket b_1 \rrbracket \,\|\|\, H)$	X_{b_1}
$\mathsf{symb}(\llbracket b_2 \rrbracket \,\|\|\, H)$	X_{b_2}
$\vdots$	$\vdots$
$\mathsf{symb}(\llbracket b_{2^k} \rrbracket \,\|\|\, H)$	$X_{b_{2^k}}$

♡♣♡♣♣♡	X_{11}
♡♣♣♡♣♡	X_{10}
♣♡♡♣♣♡	X_{01}
♣♡♣♡♣♡	X_{00}

On the left we depict the general *start state* of a protocol with helping sequence H, computing a function $f\colon \{0,1\}^k \to \{0,1\}^m$ w.r.t. deterministic input encoding $\llbracket\cdot\rrbracket$, where $\{b_1, \ldots, b_{2^k}\} \coloneqq \{0,1\}^k$. On the right, we have the start state of the six-card AND protocol of Mizuki and Sone [MS09], cf. Example 6.1. To simplify notation, we write down only the symbol sequences[1], not the full cards. Moreover, we write all sequences as strings. For example, in the state on the right, the first line can be read as follows: The current card sequence is (?/♡, ?/♣, ?/♡, ?/♣, ?/♣, ?/♡) with symbolic probability X_{11}, i.e., the probability that this sequence is the actual sequence on the table is exactly the probability of $(1,1)$ being the input to the protocol. Similarly, in the general start state on the left, the first line can be read as: the sequence $\llbracket b_1 \rrbracket \,\|\, H$ is the actual sequence with the probability of b_1 being chosen as input.

[1] This is because we almost always use back alphabet $\{?\}$ and keep cards essentially face-down due to Section 6.8, so no ambiguities will arise. This simplifies notation significantly.

For a more complex state, let us look at the situation of the AND protocol after the shuffle of Example 6.1, which is as follows (also see the full graph in Figure 7.1 for context):

♡♣♡♣♣♡	$\frac{1}{2}X_{11}$
♣♡♣♡♡♣	$\frac{1}{2}X_{11}$
♡♣♣♡♣♡	$\frac{1}{2}X_{10} + \frac{1}{2}X_{00}$
♣♡♣♡♣♡	$\frac{1}{2}X_{10} + \frac{1}{2}X_{00}$
♣♡♡♣♣♡	$\frac{1}{2}X_{01}$
♡♣♣♡♡♣	$\frac{1}{2}X_{01}$

Here, we have introduced uncertainty by a shuffle operation which applies the identity permutation id or (1 2)(3 5)(4 6), each with probability $\frac{1}{2}$. For example, ♡♣♡♣♣♡ (first line in the state) is on the table if the input was $(1, 1)$ and id has been chosen in the shuffle, resulting in a remaining total probability of $\frac{1}{2}X_{11}$. On the other hand, ♡♣♣♡♣♡ (third line in the state) can arise via input $(1, 0)$ and choosing id, or by input sequence ♣♡♣♡♣♡ (if inputs were $(0, 0)$) on which (1 2)(3 5)(4 6) is applied, resulting in a symbolic probability of $\frac{1}{2}X_{10} + \frac{1}{2}X_{00}$. We will later see more systematically how we can derive such states following a shuffle action.

Let us start with a formal definition of a state. For this, let $\mathbb{R}[X_b \colon b \in \{0,1\}^k]_1$ denote the *homogeneous polynomials of degree* 1 with variables X_b for $b \in \{0,1\}^k$, i.e., the polynomials from $\mathbb{R}[X_b \colon b \in \{0,1\}^k]$ (with coefficients in $\mathbb{R}$ and variables $\{X_b \colon b \in \{0,1\}^k\}$) such that their monomials all have degree 1. Note that this is an abelian group w.r.t. addition and includes the zero polynomial, denoted as 0 in the following. We say that a polynomial $P \in \mathbb{R}[X_b \colon b \in \{0,1\}^k]_1$ is *non-negative*, if all of its coefficients are non-negative, i.e., greater or equal to 0, and write $P \geq 0$ for short.

Definition 7.1 (State)**.** Let $\mathcal{D}$ be a deck and $\{0,1\}^k$ be a set of inputs. A *state* μ *with deck* $\mathcal{D}$*, variables from* $\{0,1\}^k$ *and sequence length* $\ell \leq |\mathcal{D}|$ is a map $\mu \colon \mathsf{Seq}_\ell^{\mathcal{D}} \to \mathbb{R}[X_b \colon b \in \{0,1\}^k]_1$ such that

1. $\sum_{s \in \mathsf{Seq}_\ell^{\mathcal{D}}} \mu(s) = \sum_{b \in \{0,1\}^k} X_b$ (representing that μ can be seen as a probability distribution on card sequences, with coefficients of the convex combination summing to 1, which is formally given as $\sum_{b \in \{0,1\}^k} X_b$, cf. the discussion below),

2. for all $s \in \mathsf{Seq}_\ell^{\mathcal{D}}$, $\mu(s)$ is non-negative (as before, convex combinations require its coefficients to be non-negative), and

3. all $s \in \mathsf{Seq}_\ell^{\mathcal{D}}$ with $\mu(s) \neq 0$ have the same visible sequence.

We say that a state μ *contains* a sequence of symbols s (or s *is in* μ for short) if $\mu(s) \neq 0$, and write $\mathsf{supp}(\mu)$ (as in *support*) for the set of all such sequences. Let $|\mu| := |\mathsf{supp}(\mu)|$. We write $\mathsf{vis}(\mu)$ for the (unique) visible sequence of the sequences in μ. Moreover, we will often leave $\mathcal{D}$ and $\{0,1\}^k$ implicit, if no ambiguities can arise.

As our aim was to use this formalism to express a (symbolic) probability distribution on sequences being the current actual sequence, we require the first two conditions in

the definition of a state. We call them *convexity conditions*. They allow us to interpret these polynomials as probabilities for the respective sequence to be the actual sequence at this time in the protocol, depending on the (symbolic) probabilities of the inputs b, given as variables X_b for $b \in \{0,1\}^k$.

Formally, we want that $\sum_{b\in\{0,1\}^k} X_b = 1$, i.e., that the sum of the probabilities of all input sequences is 1, but refrain from introducing the basic polynomial ring as a factor ring modulo $\sum_{b\in\{0,1\}^k} X_b - 1$ for ease of exposition. This lets us interpret the first part of the convexity condition as having the coefficients of a convex combination summing to 1. Importantly, the convexity conditions imply that the polynomials over the variables X_b for $b \in \{0,1\}^k$ are of the form $\sum_{b\in\{0,1\}^k} \beta_b X_b$, for $\beta_b \in [0,1] \subseteq \mathbb{R}$.

7.1. Constructing State Trees from Protocols

We now describe how to construct state trees from protocol descriptions. We will later argue how to identify states which are essentially equal, introducing backwards edges and cycles to the state diagrams for more compact, often finite graphs. This section is mainly taken from [KKW+17, Sect. 3.1] with adaptations and extensions. See Figure 7.1 as an illustration of the more general rules. In the following, v and v' are always visible sequence traces of the prefix of a protocol run, with v' being the trace that arise from v by adding one visible sequence v^+, i.e., $v' = v \parallel v^+$.

For each v, the state tree contains a state μ_v. In our model, each state is associated with a unique action that the protocol prescribes for this situation. We annotate the state (essentially its outgoing edges) with this action. If the action leads to an extension of v to v' by appending the visible sequence v^+, then $\mu_{v'}$ is a *child* of its *parent* μ_v. A turn action, or a shuffle action on face-up cards or cards with distinct back symbols, may result in several children which are *siblings* of each other and the edge to each child is additionally annotated with the respective v^+. We call a shuffle on cards with a non-trivial visible sequence that has multiple children a *branching shuffle*. If the action is a result action, then μ_v is a *leaf* or a *final state*.

From the perspective of an observer of the protocol who does not know the input I or the permutations chosen during shuffle actions, the *actual sequence* S_v lying on the table in a particular run when the state μ_v is reached, is unknown. We interpret S_v as a random variable and annotate μ_v with the corresponding probabilities, i.e., with $\mu_v(s) := \Pr[S_v = s \,|\, v]$ where s is any card sequence from $\mathsf{Seq}_\ell^{\mathcal{D}}$ and v stands for the event that v is a prefix of the visible sequence trace of the complete protocol run. We can then rewrite this as:

$$\begin{aligned}
\mu_v(s) = \Pr[S_v = s \,|\, v] &= \sum_{i\in\{0,1\}^k} \Pr[S_v = s, \mathsf{val_i}(I) = i \,|\, v] \\
&= \sum_{i\in\{0,1\}^k} \Pr[S_v = s \,|\, \mathsf{val_i}(I) = i, v] \cdot \Pr[\mathsf{val_i}(I) = i \,|\, v] \\
&= \sum_{i\in\{0,1\}^k} \underbrace{\Pr[S_v = s \,|\, \mathsf{val_i}(I) = i, v]}_{=:p_{v,i,s}} \cdot \underbrace{\Pr[\mathsf{val_i}(I) = i]}_{=:X_i} = \sum_{i\in\{0,1\}^k} p_{v,i,s} X_i.
\end{aligned}$$

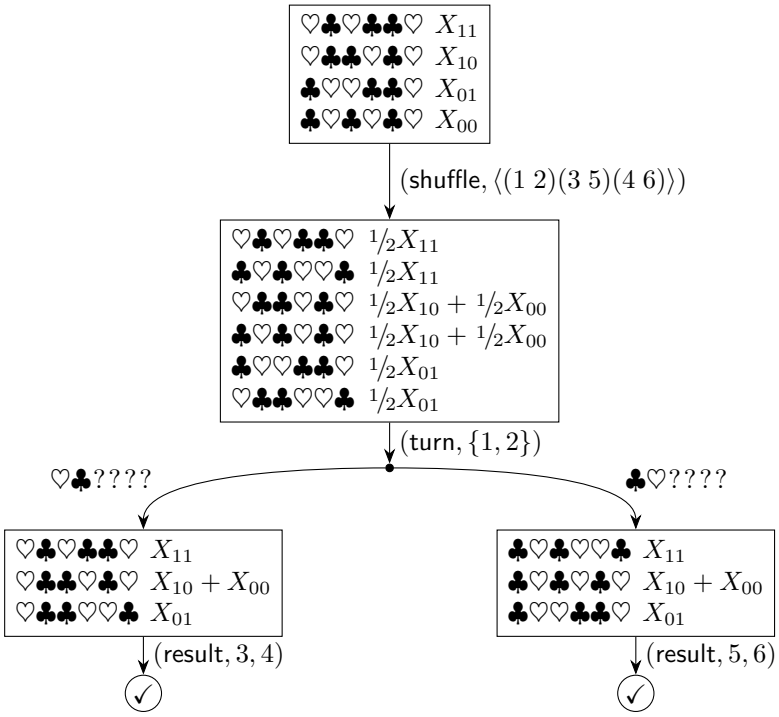


Figure 7.1.: The six-card AND protocol of [MS09].

At the second line-break, we exploited the independence of visible sequence trace and input in secure protocols and introduced two abbreviations. Note that $p_{v,i,s}$ is constant, but X_i is a variable since we have not actually defined a specific probability distribution on the inputs (nor will we). We treat the result as a formal sum, making μ_v a map from sequences of symbols to polynomials, as in Definition 7.1.

Derivation Rules for States

Let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a secure protocol computing a Boolean function $f\colon \{0,1\}^k \to \{0,1\}^m$ w.r.t. input encoding $[\![\cdot]\!]_{\mathrm{i}}$ and output encoding $[\![\cdot]\!]_{\mathrm{o}}$. We now describe how the tree of all states can be computed for $\mathcal{P}$ inductively starting from the root.

Start state. First note that the start state is unique, which was, besides simplicity, why we required all sequences in U to be face-down and share the same visible sequence v_0. The distribution of S_{v_0} is the input distribution, with the helping card sequences put to the right, i.e.,

$$\mu_{v_0}(s) = \sum_{b,r_{\mathrm{i}} \text{ with } s=[\![b;r_{\mathrm{i}}]\!]_{\mathrm{i}} \,\|\, H} \Pr[R_{\mathrm{i}} = r_{\mathrm{i}}] \cdot X_b. \tag{7.1}$$

(Note that we assumed in the definition of encoding that the b is unique, but there may be several random values leading to the same sequence). Here, we already assume

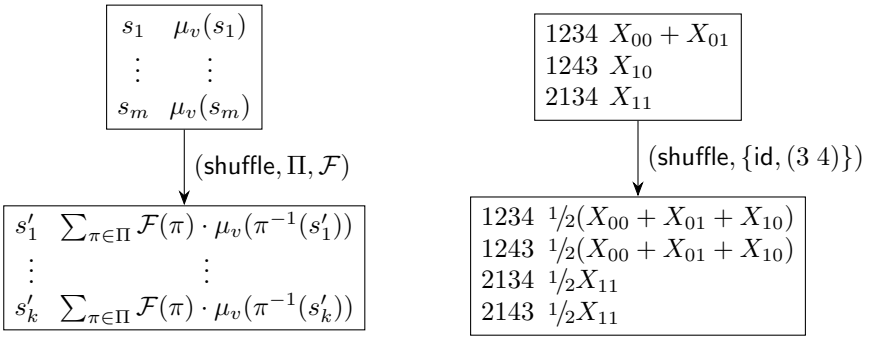


Figure 7.2.: A non-branching shuffle operation. The general rule is on the *left*, an example from a two-bit input protocol is given on the *right*.

that R_i is stochastically independent of the input value to be encoded[2]. If the input encoding is deterministic, then this simplifies to

$$\mu_{v_0}(s) = \begin{cases} X_b, & \text{if } s = [\![b]\!]_\mathrm{i} \parallel H, \\ 0, & \text{otherwise.} \end{cases} \tag{7.2}$$

This is also the state displayed on p. 61 (left), which we have seen at the beginning of this chapter. It is clear that this fulfills the definition of a state, as in Definition 7.1.

Shuffle action. Assume for a state μ_v the action $(\mathsf{shuffle}, \Pi, \mathcal{F})$ is prescribed. Let us first assume that before the shuffle the visible sequence is the trivial visible sequence $(?, \ldots, ?)$. Then no non-trivial information is revealed upon shuffling, and we obtain v' by appending a trivial visible sequence to v. Then $\mu_{v'}$ is the unique child of μ_v, fulfilling

$$\begin{aligned} \mu_{v'}(s) &= \Pr[S_{v'} = s \,|\, v'] \\ &= \sum_{\pi\in\Pi} \mathcal{F}(\pi) \cdot \Pr[S_v = \pi^{-1}(s) \,|\, v] = \sum_{\pi\in\Pi} \mathcal{F}(\pi) \cdot \mu_v(\pi^{-1}(s)). \end{aligned}$$

This just takes into account all sequences $\pi^{-1}(s)$ in μ from which s may have originated via some π as well as their corresponding probabilities. See Figure 7.2 for an example situation in a state tree. Note that perm actions are simply a special case, see Figure 7.3 for an example.

If the visible sequence before the shuffle is non-trivial, there are several child states. For this, let Π_{v^+} be the subset of Π that leads to some visible sequence v^+ with corresponding state $\mu_{v'}$. If $\mathcal{F}(\cdot \mid v^+)$ denotes the probability distribution on Π_{v^+} conditioned on the fact that v^+ is observed, we have that

$$\mu_{v'}(s) = \sum_{\pi\in\Pi_{v^+}} \mathcal{F}(\pi \mid v^+) \cdot \mu_v(\pi^{-1}(s)). \tag{7.3}$$

[2]This is the precondition to security as in Definition 6.2 and constitutes the only interesting case for us, if we want our state diagrams to be a nice representation that allows for showing security by simple, local checks.

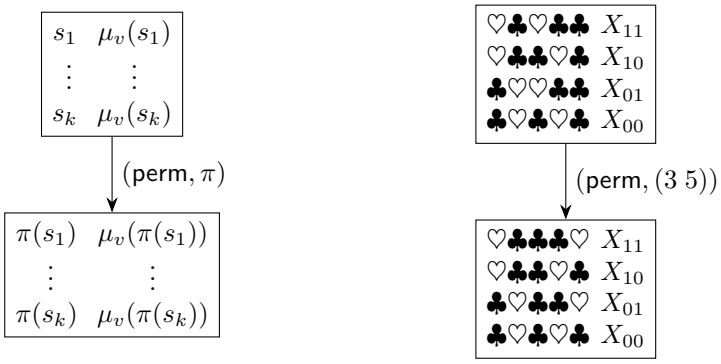


Figure 7.3.: A permutation operation. The general rule is on the *left*, an example from den Boers protocol [dB90] is given on the *right*. In the example, the helping card is sorted in between the two bits while at the same time inverting the order of the second bit, leading to a situation where the hearts are cyclically adjacent if and only if the input was $(1, 1)$.

In other words, the probability for the sequence s in the new state $\mu_{v'}$ is obtained by considering all sequences $\pi^{-1}(s)$ from which s may have originated through some $\pi \in \Pi_{v^+}$ and summing the probability of those sequences in μ_v, weighted with the probabilities that the corresponding π is chosen. An example of such a *branching shuffle*, is given in Figure 7.4. Note that as μ_v is assumed to be a state, also $\mu_{v'}$ fulfills the definition, as (7.3) is a convex combination of several states as follows: Starting from μ_v, we look at all states $\mu_v(\pi^{-1}(\cdot))$ for the different $\pi \in \Pi$ and then use the coefficients from $\mathcal{F}$ to form a convex combination of these states.

Turn action. Let μ_v be a state with a turn action with turn set $T \subseteq \{1, \ldots, \ell\}$ that possibly results in the visible sequence v^+, which appended to v, yields v'. The child $\mu_{v'}$ of μ_v contains the sequences s whose symbol sequences coincide with a symbol sequence from a sequence in μ_v but with the cards at T having been turned via $\mathsf{swap}(\cdot)$, and that are *compatible* with v^+ at T, i.e., for which $\mathsf{vis}(s)[T] = v^+[T]$ holds. Then, the probability to reach $\mu_{v'}$ from μ_v is:

$$\Pr[v' \,|\, v] = \Pr[S_v \in \mathsf{supp}(\mu_{v'}) \,|\, v] = \sum_{s \in \mathsf{supp}(\mu_{v'})} \Pr[S_v = s \,|\, v] = \sum_{s \in \mathsf{supp}(\mu_{v'})} \mu_v(s). \tag{7.4}$$

Recall that the right hand side is a polynomial of the form $\sum_{b \in \{0,1\}^k} a_b X_b$ where X_b is a placeholder for the input probability $\Pr[\mathsf{val}_{\mathsf{i}}(I) = b]$. By security, input and visible sequence trace are independent, in particular no matter how the variables $(X_b)_{b \in \{0,1\}^k}$ are initialized, the polynomial $\Pr[v' \,|\, v]$ evaluates to the same real number. Thus, all a_b are equal to a constant $\beta_{v'} \in [0, 1]$ and using $\sum_{b \in \{0,1\}^k} X_b = 1$ we have:

$$\sum_{s \in \mathsf{supp}(\mu_{v'})} \mu_v(s) = \beta_{v'} \sum_{b \in \{0,1\}^k} X_b = \beta_{v'}.$$

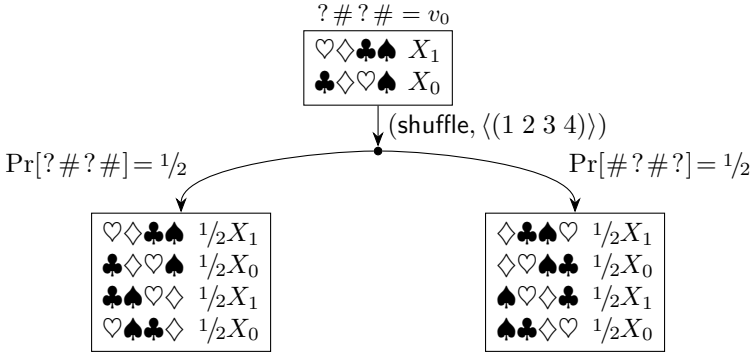


Figure 7.4.: An example of a branching shuffle, due to differing backs. Here, the initial visible sequence v_0 is displayed on top of the start state. Due to the alternating backs, we have partial permutation sets $\Pi_{?\#?\#} = \{\mathsf{id}, (1\ 2\ 3\ 4)^2\}$ and $\Pi_{\#?\#?} = \{(1\ 2\ 3\ 4), (1\ 2\ 3\ 4)^3\}$.

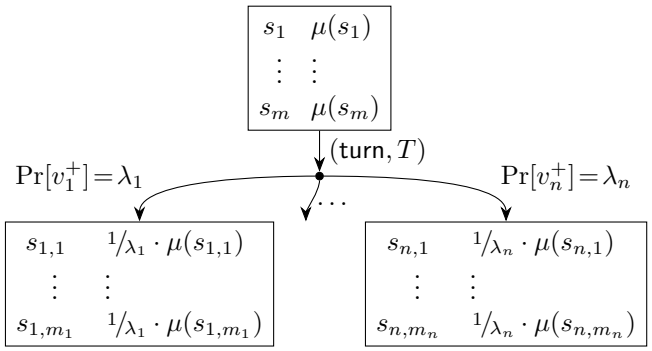


Figure 7.5.: A turn operation. Here, $v_1^+, \ldots, v_n^+$, are the possible observations by turning the cards at positions in T. For each $i \in \{1, \ldots, n\}$ the $s_{i,1}, \ldots, s_{i,m_i}$ are the sequences from $s_1, \ldots, s_m$ which are compatible with v_i^+. Note that in secure protocols, the probability of observing v_i^+ is constant.

Moreover, for $s \in \mathsf{supp}(\mu_{v'})$ we have (using Bayes' formula):

$$\mu_{v'}(s) = \Pr[S_{v'} = s \,|\, v'] = \frac{\Pr[S_{v'} = s \,|\, v]}{\Pr[v' \,|\, v]} = \frac{\Pr[S_v = s \,|\, v]}{\beta_{v'}} = \frac{\mu_v(s)}{\beta_{v'}}. \tag{7.5}$$

Hence, (when reducing to symbol sequences,) $\mu_{v'}$ is a restriction of μ_v to sequences compatible with v', scaled by $\frac{1}{\beta_{v'}}$. Therefore, $\mu_{v'}$ fulfills the definition of a state. Note that the security of the protocol is important for this implication. For reference, see Figure 7.5. The randomized flip action can be defined analogously.

Capturing the observation we made for secure protocols, we define

$$T \text{ is } \textit{turnable} \text{ in } \mu \;:\Leftrightarrow\; \forall s \in \mathsf{Seq}^{\mathcal{D}}_{|T|} \colon \exists \beta \in [0,1] \colon \sum_{s' \text{ with } s'[T]=s} \mu(s') = \beta \sum_{b \in \{0,1\}^k} X_b. \tag{7.6}$$

Restart action. States in which a restart action happens have a single child. The subtree at this child is equal to the entire tree.

Result actions. Consider any leaf state μ_v with action $(\mathsf{result}, p_1, p_2, \ldots, p_r)$. The output $O = (S_v)_{p_1,\ldots,p_r}$ is the projection of S_v to the components $p_1, \ldots, p_r$ (in this order). We can easily obtain from μ_v the probabilities $\Pr[O = o \,|\, v]$ for any card sequence $o \in \mathsf{Seq}_r^{\mathcal{D}}$. For protocols computing a function w.r.t. output encoding $[\![\cdot]\!]_{\mathrm{o}}$ we can also derive, for a fixed b, $\Pr[\mathsf{val}_{\mathrm{o}}(O) = b \,|\, v]$. By security as in Definition 6.2, the visible sequence trace v is irrelevant for the latter, which implies that we obtain the same polynomial $\Pr[\mathsf{val}_{\mathrm{o}}(O) = b]$, regardless of which leaf state we examine. If we have full-sequence security, the visible sequence trace v is irrelevant also for $\Pr[O = o \,|\, v]$, which implies that we again obtain the same polynomial $\Pr[O = o]$ for a fixed o, regardless of the leaf state. In case of a randomized output encoding we will derive a more formal rule to verify security in (7.7) of Section 7.5.

To construct a state tree via these rules, we started by formally specifying the start state and then, how subsequent states are derived from a given state when performing the prescribed action. These rules can also be seen as an inductive proof that our definition of a state is sound in secure protocols, as the probabilities are from $\mathbb{R}[X_b \colon b \in \{0,1\}^k]_1$ retaining the convexity conditions as claimed.

On an Interpretation of the States. If s is a sequence in μ_v and $\mu_v(s)$ the corresponding polynomial, substituting 1 for the variable X_b and 0 for the other variables in $\mu(s)$, yields the probability that s is the current sequence, given the input b and any information observed so far. Accordingly, we can use our notions to analyze player knowledge in multiparty computations where an agent has partial information about the input.

Example. As an illustration of our method, let us walk through the states of the six-card AND protocol from above, see Figure 7.1 on page 64, as in [KWH15].

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

- The shuffle introduces uncertainty. Consider for instance the case that the input was "10". Then, before the shuffle, we must have had the sequence $s = (?/\heartsuit, ?/\clubsuit, ?/\heartsuit, ?/\clubsuit, ?/\clubsuit, ?/\heartsuit)$. It was either permuted by id or by $\pi = (1\ 4)(2\ 5)(3\ 6)$, yielding either s itself or $s' = (?/\clubsuit, ?/\clubsuit, ?/\heartsuit, ?/\heartsuit, ?/\clubsuit, ?/\heartsuit)$, both with probability $1/2$. This explains the coefficients of X_{10} in the polynomials for s and s'.

- The turn step can yield two possible visible sequences: $\heartsuit\clubsuit????$ and $\clubsuit\heartsuit????$. Crucially, the probability of observing $\clubsuit\heartsuit????$ is the same for each possible

input, so no information about the actual sequence is leaked: If ♣♡???? would be observed slightly more frequently for, say, the input "01" than for the input "10", then observing ♣♡???? would be weak evidence that the input was "01". In the case at hand, however, the probability for the right branch is $1/2$ for each input, as the sum of the polynomials of the sequences branching right is $1/2(X_{11} + X_{10} + X_{01} + X_{11})$.

After the turn our knowledge has changed, for instance, if we have observed ♡♣???? and know that the input was "11" then we know beyond doubt that the symbol sequence must then be ♡♣♡♣♣♡, explaining the coefficient 1 of X_{11}.

- The output given by the result actions is correct: For all polynomials containing X_{11} with non-zero coefficient, the corresponding sequence has ($?/♡$, $?/♣$) at the specified positions and for all polynomials containing one of the other variables with non-zero coefficient, the corresponding symbol sequence has ($?/♣$, $?/♡$) there.

Note that "mixed" polynomials with non-zero coefficients for variables of both types (for output 1 and output 0) cannot occur in a final state of a protocol computing a *deterministic* function. If they occur at some other vertex μ in the tree, a restart would be necessary in the subtree starting from μ, as we cannot guarantee correctness otherwise.

For randomized functions, "mixed" polynomials may actually occur. Hence, let us give a definition of correctness which also includes randomized functions:

Definition 7.2 (Correctness)**.** Let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a secure card-based protocol computing a function $f\colon \{0,1\}^k \to \{0,1\}^m$ w.r.t. output encoding $[\![\cdot]\!]_o$. As above, consider any leaf state μ_v with action $(\mathsf{result}, p_1, p_2, \ldots, p_r)$. The output $O = (S_v)_{p_1,\ldots,p_r}$ is the projection of S_v to the components $p_1, \ldots, p_r$ (in this order). We can then derive the probability of the output being $x \in \{0,1\}^m$ given input $i \in \{0,1\}^k$ in μ_v as $F_x(b) := \Pr[\mathsf{val}_o(O) = x \,|\, \mathsf{val}_i(I) = i, v]$ by setting $X_i = 1$, and $X_{i'} = 0$ for all $i' \neq i$ in the polynomial of symbolic probabilities for the output $\Pr[\mathsf{val}_o(O) = x \,|\, v]$. If the protocol is *correct*, then this probability will equal the probability of $f(i) = x$ in all leaf states.

If f is a deterministic function, this simplifies to: the card sequence at the result positions encodes value $x \in \{0,1\}^m$ if for all X_i occurring in $\mu_v(s)$ for a sequence s it holds that $f(i) = x$.

7.2. Formally Defining State Diagrams

Having derived how states arise from other states via the available actions, we can now define what we mean by a state diagram of a secure card-based (Mizuki–Shizuya) protocol. This is captured in the following definition.

Definition 7.3. Let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a secure card-based protocol computing a function $f\colon \{0,1\}^k \to \{0,1\}^m$. A *state diagram of* $\mathcal{P}$ is a 5-tuple $(G, \mu_0, \lambda_a, \lambda_v, \lambda_p)$, where

- G is a connected, directed graph with vertex set $V(G)$ and edge set $E(G)$, where the vertices are annotated with states of deck $\mathcal{D}$, variables from $\{0,1\}^k$ and sequence length ℓ as defined above. (We leave this annotation map implicit and directly treat the states as vertices in the following.)
- $\mu_0 \in V(G)$ is the *start state* of $\mathcal{P}$, representing the situation at the beginning of a protocol run,
- $\lambda_a \colon V(G) \to \mathsf{Action}_\ell$ is a vertex labeling of G, specifying the action to be performed next after reaching this state (with the condition given below that the actions coincide with those specified by action function A of $\mathcal{P}$). If t is the type of the assigned action, we also say that the state is a state *with* t*(-action).*
- $\lambda_v \colon E(G) \to \mathsf{Vis}_\ell^{\mathcal{D}}$ is an edge labeling of G, that annotates to each outgoing edge of a vertex μ a visible sequence that may result from doing $\lambda_a(\mu)$, to denote that if this visible sequence is observed in a protocol run, the corresponding edge has to be taken to reach the new state,
- $\lambda_p \colon E(G) \to [0,1]$ is an edge labeling of G, annotating to each edge the probability of it being taken during a protocol run. This will allow us to interpret the state diagram as a transition graph of a (homogeneous) Markov chain. We will give more details on this below.

Importantly, we require *well-formedness*, i.e.,

1. μ_0, and states with an incoming edge from a state with $\mathsf{restart}$, satisfy (7.1),
2. states with an incoming edge from a state with perm or $\mathsf{shuffle}$ obey (7.3),
3. states with an incoming edge from a state with turn obey the relation in (7.5),
4. all sequences of a state have the same visible sequence as the (unique) visible sequence assigned to all incoming edges via λ_v,
5. the probability assigned to each edge via λ_p is 1 if it is the only outgoing edge, and given as in (7.4), otherwise.
6. In all leaf nodes of the protocol for the positions given by the result action, we have correctness as given in Definition 7.2.

If the graph of a state diagram is a tree, we also call it a *state tree.*

As we have already seen several examples of state diagrams in the figures given above, we refer to Figure 7.6 (and later Figure 7.7) for an explanation how these (intuitive) drawings of state trees translate to our formal definition. Moreover, for conciseness, we may omit states and write multiple non-branching actions at a single edge. For example, when turning back cards revealed by a turn, nothing interesting happens, so that we usually turn them back again, without specifying the state again. Moreover, when all outgoing edges have the same probability, we may omit its specification.

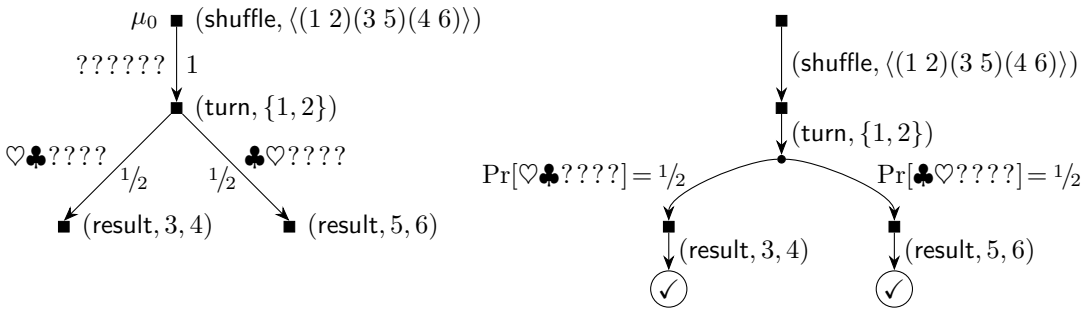


Figure 7.6.: The state tree of the six-card AND protocol of [MS09] in two variants. On the *left*, we show the graph of the state tree with vertices depicted as small black squares, with action annotation λ_a to the right of the vertices, and visual sequences from λ_v and probabilities of taking the respective edge from λ_p on the left or right of the corresponding edges. On the *right* we propose a better-readable form of depicting the same graph by leaving out visible sequences and probabilities when they are trivial or 1, respectively. This leaves space to write the action annotation at the outgoing edge, which is more intuitive. Vertices with more than one outgoing edge are depicted by a short edge with the respective action, followed by a branching where edges are labeled with visible sequences v and their probabilities p, written in form $\Pr[v]=p$. The result and restart actions are depicted by a short outgoing edge pointing to a circled checkmark or circular arrow, respectively. The topmost vertex is always the start state.

7.3. Identifying Recurring States

Using our construction in Section 7.1, it may well happen that one arrives at an infinite tree. This occurs when there is a positive probability of looping in the protocol, such as in Las Vegas protocols. As these occur quite often, it is useful to introduce backwards edges, which allows to draw all relevant protocols from the literature as a simple finite graph. For an example of a graph that stays infinite, we refer to Figure 7.8. (We draw it in a finite way by introducing a variable n that is increased by taking the backwards edge.)

We do this by identifying vertices of the tree at which the exact same (randomized) behavior is supposed to follow. For this, we first introduce a equivalence relation on finite visible sequence traces. Let v, $\tilde{v}$ be such visible sequence traces, we say that v and $\tilde{v}$ are *equivalent*, or $v \sim \tilde{v}$, if the protocol run (prefix) with visible sequence trace v ends in the same state from Q, as the run with $\tilde{v}$, and the last visible sequence in both traces is the same. This will guarantee that the next action to be prescribed by the action function A in both situations is equal. (Note that this does not yet say anything about the steps afterwards, as the probabilities in the states can be different. This will be accounted for next.)

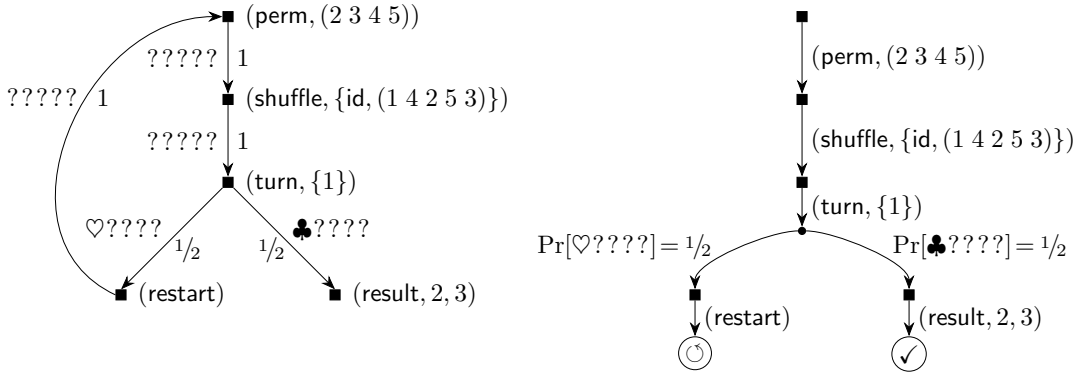


Figure 7.7.: The state tree of the five-card AND protocol of [CHL13], in two variants as in Figure 7.6, with the "raw" graph on the *left*, and the better-readable form on the *right*. This protocol uses all four types of actions and therefore serves as an extended example of state trees. The drawing uses the same conventions as Figure 7.6.

Now, we identify two states μ_v, $\mu_{\tilde{v}}$ if i) $v \sim \tilde{v}$, and ii) for all sequences $s \in \mathsf{Seq}_\ell^{\mathcal{D}}$ it holds: $\mu_v(s) = \mu_{\tilde{v}}(s)$, i.e., the state is equal (as a map)[3]. By abuse of notation, we will also write $\mu_v \sim \mu_{\tilde{v}}$ and say that μ_v is *equivalent* to $\mu_{\tilde{v}}$ in this case. For example, in Figure 7.7, we identify the state following the restart action with the start state μ_0, and draw a corresponding backwards edge.

When defining state diagrams in Definition 7.3, we took care to also account for graphs with cycles, hence one can verify that the resulting graphs by identifying states as described above are state diagrams again. This is because all the rules still hold when identifying equivalent states.

As a corollary to the preceding discussion and definitions, we summarize the content of the previous and this section in the following proposition:

Proposition 7.1. *Every secure protocol $\mathcal{P}$ has a unique state diagram (with the equivalence relation applied) that can be obtained as described above. If the graph of the state diagram is finite, one can derive a description of a protocol from the diagram with the same computational behavior, i.e., for each input produces the same distribution of visible sequence traces and output sequences.*

Proof. Uniqueness is immediate, as the derivation rules and the equivalence relation do not leave any choice. For the second statement, we want to derive a protocol $\mathcal{P}' = (\mathcal{D}', U', H', Q', A')$ with the same computational behavior. The deck $\mathcal{D}'$, the set of input sequences U' and the helping sequences H' can be directly deduced from the start state and possibly the visible sequences at the edges of the diagram (to aid this, when the visible sequence uses a back alphabet with more than one element, we may write it

[3]This second condition is arguably more important. It happens rarely that the states are equal as maps, but a different action is to be performed next. If this occurs, one could well unify these actions to the action that would reduce the overall expected running time of the protocol, as in randomized flip operations.

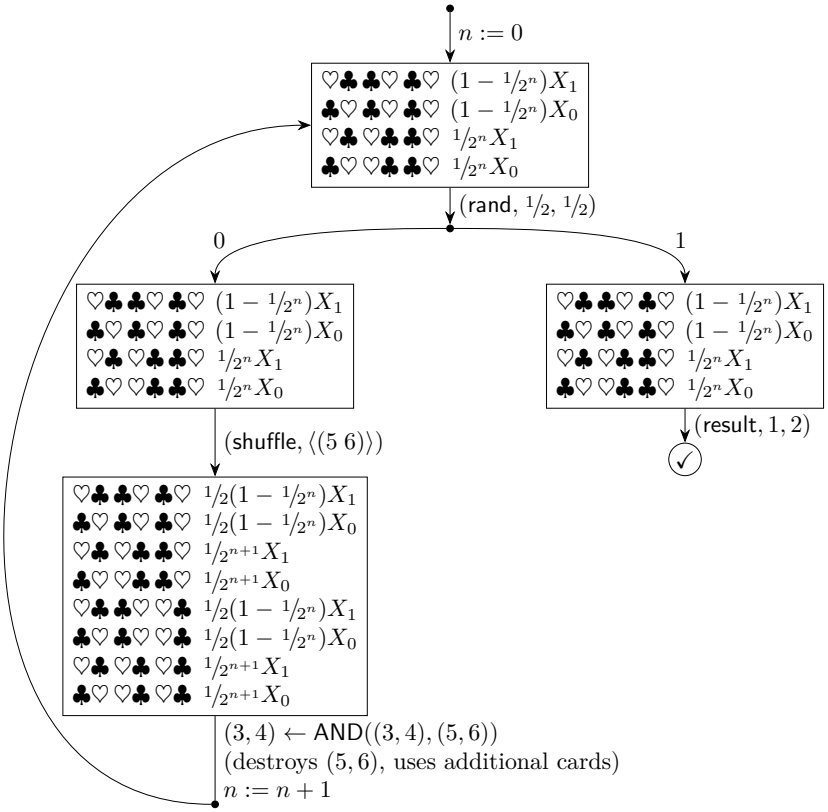


Figure 7.8.: The state diagram of a protocol computing the identity function $x \mapsto x$ in Las Vegas fashion. While this does not look like a particularly useful protocol, it serves as an example that state diagrams can be infinite. While our depiction of this diagram is finite, this is only because we introduced n as a variable in the state descriptions. As n increases during each run in this tree cycle, the states on the cycle are strictly distinct.

above the start state as in Figure 7.4). While there may be a choice in which cards to assign to the sequences in U' and to H', we assume that, starting from the right, all constant columns in the start state are assigned to H', as there might be little reason to use constant positions in the input encoding. The set Q' consists of the vertices of the graph (after identification of equivalent states), with the inital state q'_0 being μ_0. The action function can be derived from the action labeling λ_a to the states in the diagram, and their visible sequence, and the states to which the state connected via outgoing edges. The $q \in Q'$ following a leaf state is set as the final state $q = q'_{\text{fin}}$. □

Hence, (finite) state diagrams present a useful alternative way to describe secure protocols, from which security and correctness can be more directly observed. Note that we do not claim that Q' will equal to Q of the original protocol from which the state diagram was generated, as one can easily find counterexamples. For example, if Q

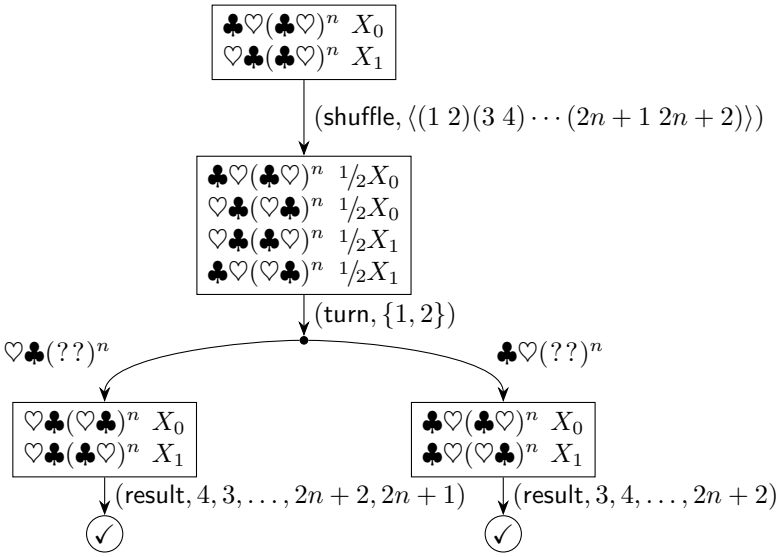


Figure 7.9.: The state tree of the $(2n+2)$-card COPY protocol of [MS09]. It uses only a random bisection cut.

contains states that are never reached, the described process may result in smaller Q'. Also, as the states contain additional information of the different runs of a protocol, Q' may be well larger. For this, consider the protocol that is given a long string of bits encoded as cards, and randomizes each of them by shuffling the two cards at the beginning of the sequence and then cyclically shifting by two, until it has randomized all bits. This may have a very concise description as a Mizuki–Shizuya protocol, but in each step, the state changes and hence needs to be represented by distinct $q \in Q'$ in the construction.

The claim that security is immediate by local checks at the branching of a turn and at the leaf states will be backed up more formally in the next section.

Figures 7.1, 7.9 and 7.10 depict state trees of AND and COPY protocols that are identified as card-minimal w.r.t. certain restrictions to running time behavior and used shuffles in Chapter 9.

Running Time in State Diagrams. Let us reinterpret our running time definitions of card-based protocols in terms of properties that one can observe when looking at the state diagram. We have that a protocol's running time is finite, when its state diagram is a finite tree. A *path in a state diagram* p starts at the start state μ_0 and proceeds along outgoing edges according to a maximal (and possibly infinite) visible sequence trace v. If v is finite, p ends in a leaf. The *probability* of p is $\Pr[v]$, which is a constant (independent of the distribution of inputs) by our discussion of security before. If the state diagram is neither finite nor acyclic, but the expected length of p is finite, we call the protocol *Las Vegas*.

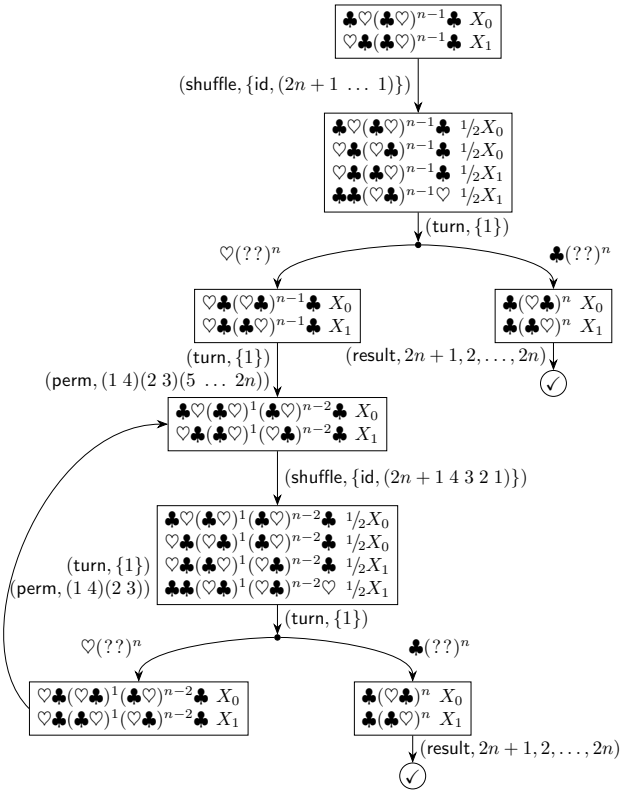


Figure 7.10.: A variant of the $(2n+1)$-card COPY protocol of Nishimura et al. [NNH+18, Sect. 5], with less permuting from [KKW+17]. After the perm in the first ♡-branch, the protocol resembles a 2-COPY protocol [NNH+15] on the first four and the last card. The parenthesis $(\cdot)^1$ are to emphasize this symmetry. The shuffle steps in this protocol are uniform, but non-closed. As they consist of the identity and exactly one odd-length cycle, they can be performed using an "unequal division shuffle". A proposed implementation of these shuffles using sliding cover boxes or envelopes can be found in [NNH+18, Sect. 6].

7.4. A Locally-Verifiable Security Criterion

To check security in a state diagram, first note that shuffle actions never reveal new critical information: When shuffling with face-up cards, the shuffle may reveal information about which permutation was used to shuffle, but this information is a fresh random variable independent of all previous information.

Considering turns or randomized flips, we already identified the condition before: A turn in state μ_v does not violate security if for each visible sequence v^+ that may result from the turn, the probabilities of all s in μ_v compatible with v^+ add up to a constant (a multiple of $\sum_{b\in\{0,1\}^k} X_b$), since this exactly means that the probability to observe a visible sequence does not depend on the inputs. As this was a precondition for the derivation rule of turns, being able to construct a diagram by the rules above is

a witness to V being independent of $\mathsf{val}_\mathrm{i}(I)$ in the protocol. In this sense, Figure 7.1 is an alternative proof for the security of the six-card AND protocol of [MS09], as AND is deterministic and the protocol uses deterministic input and output encodings.

In what follows, we will focus on the case of using a deterministic input encoding. The case of randomized input encodings will be considered in the next section. By checking turnability above, we already have that V is independent of $(\mathsf{val}_\mathrm{i}(I), \mathsf{val}_\mathrm{o}(O))$. Let us now verify that also (V, R_o) is independent of $(\mathsf{val}_\mathrm{i}(I), \mathsf{val}_\mathrm{o}(O))$. As randomness in the output encoding is only relevant at the end of a protocol run, we focus on the leaf states. We want, similarly as in turnability, that the probability of each choice of output randomness is constant, i.e., it does not depend on the X_b as before. We will make this precise.

For this purpose, let rd_o be a function that maps a card sequence Γ to the set of choices for randomness that can bring about Γ w.r.t. (randomized) encoding $[\![\cdot]\!]_\mathrm{o}$. If $[\![\cdot]\!]_\mathrm{o}$ is not randomized, $\mathsf{rd}_\mathrm{o}(\cdot)$ is set to $\emptyset$ for all function inputs. For a sequence not in the image of the encoding, the output will be an error symbol $\perp$. Formally, if there is an b, r, such that $\Gamma = [\![b; r]\!]_\mathrm{o}$, we have $r \in \mathsf{rd}_\mathrm{o}(\Gamma)$. (Remember that our encoding is injective, so this b is unique for all such r.) Let $\mathcal{R}$ be a finite set containing all choices for randomness in $[\![\cdot]\!]_\mathrm{o}$, and let

$$R_{r_\mathrm{o}} \coloneqq \Pr[R_\mathrm{o} = r_\mathrm{o} \,|\, v] = \sum_{\Gamma \text{ with } r_\mathrm{o} \in \mathsf{rd}_\mathrm{o}(\Gamma)} \Pr[R_\mathrm{o} = r_\mathrm{o}, O = \Gamma \,|\, v]$$

be the probability that randomness r_o is used for the output in leaf state μ_v. Then, we can make sense of the following definition:

$$T \text{ is } \textit{outputable} \text{ in } \mu_v \;:\Leftrightarrow\; \forall r_\mathrm{o} \in \mathcal{R} \colon \exists \beta \in [0,1] \colon R_{r_\mathrm{o}} = \beta \sum_{b \in \{0,1\}^k} X_b. \tag{7.7}$$

Checking this in every leaf state, guarantees that the output randomness is independent of the inputs and outputs.

Examples. Let us reconsider the security of the five-card trick on the left of Figure 7.11. As their are no turns and only one leaf, it suffices to check that R_i for $i \in \mathbb{Z}/5\mathbb{Z}$ is independent of the inputs. Closely observing the leaf state reveals that we even have a uniform distribution, as $R_i = 1/5$ for all $i \in \mathbb{Z}/5\mathbb{Z}$.

As a second example, consider the eight-card AND protocol for standard decks on the right of Figure 7.11. While it has a deterministic input encoding, we see from the added output sequences (the dashed "states" following the result action), that the output basis admits two choices, namely $\{?_5, ?_6\}$ and $\{?_7, ?_8\}$. First note, that all turn actions are secure by the conditions above. Now, checking our outputability criterion, gives

$$R_{\{5,6\}} = 1/2 X_0 + 1/2 X_1 = 1/2, \text{ and } R_{\{7,8\}} = 1/2 X_0 + 1/2 X_1 = 1/2,$$

for both leaf states, hence satisfying the criterion. In total, this shows that the protocol is secure.

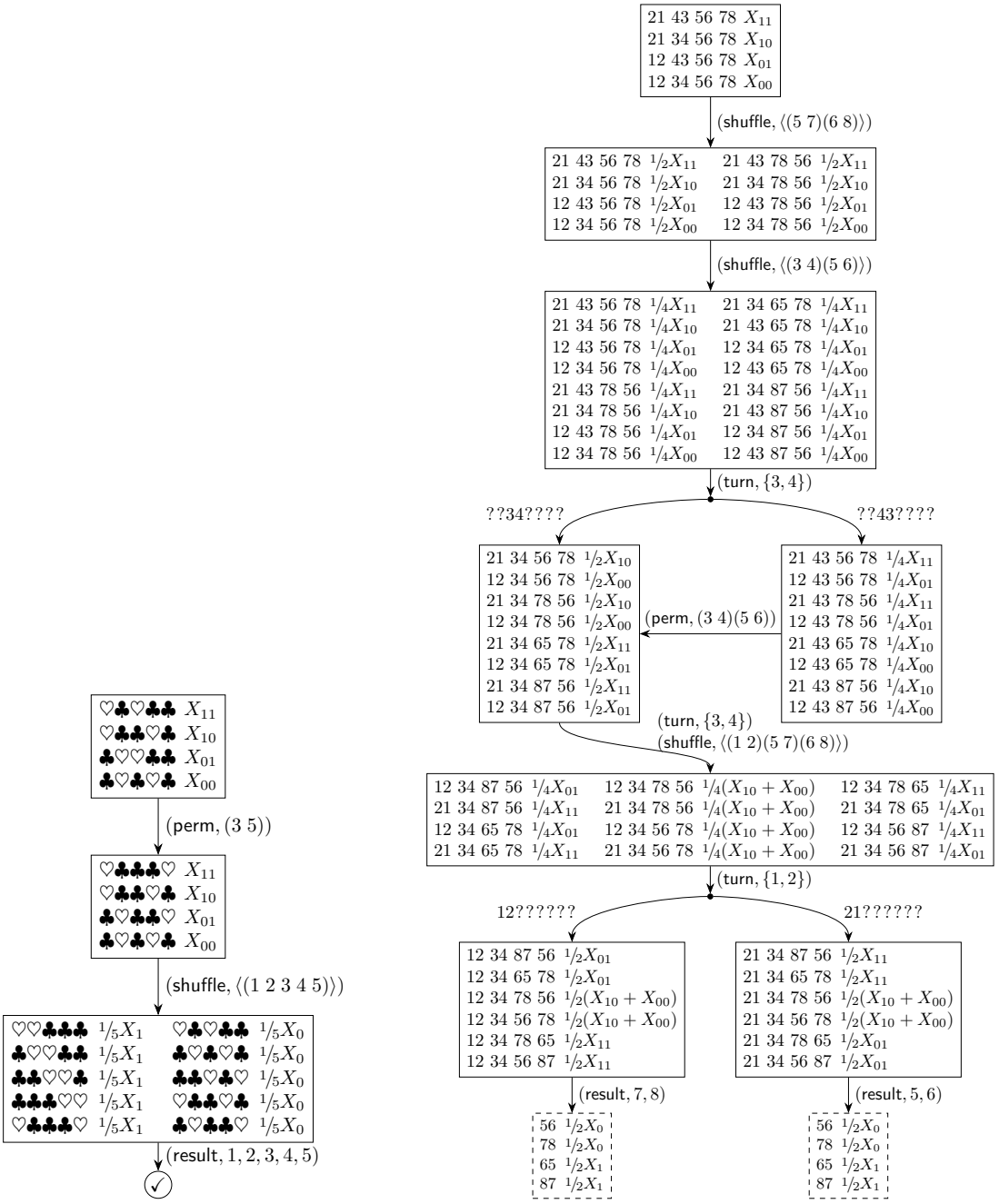


Figure 7.11.: On the *left* we give state tree of the five-card trick due to den Boer [dB90]. One can see that all choices for the randomness have the same probability $\frac{1}{5}$, hence the security criterion for the leaf nodes is fulfilled. For conciseness, we specify the state information in the final state in two columns. On the *right*, we have the finite-runtime eight-card AND protocol for standard decks of Mizuki [M16b]. Note the randomized output encoding. For conciseness, we specify the state information in some states in multiple columns.

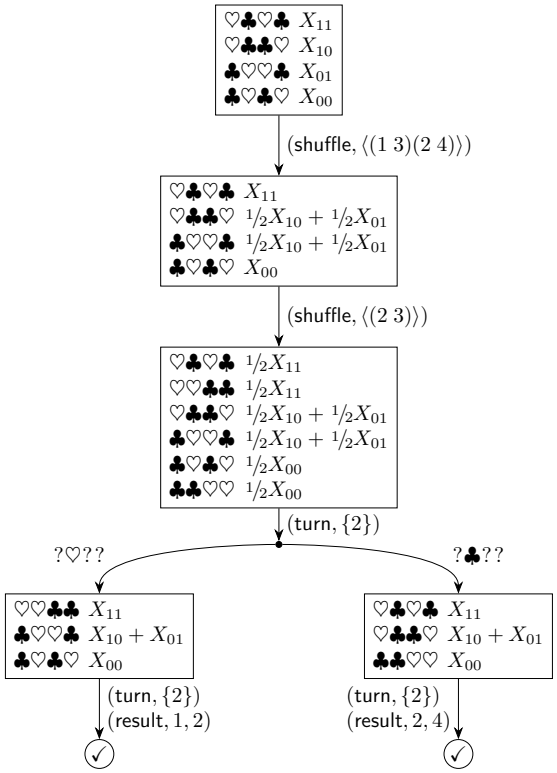


Figure 7.12.: The four-card AND protocol by Mizuki, Kumamoto, and Sone [MKS12] with non-standard (randomized) output encoding $[\![\cdot]\!]_=$ that encodes as follows: $[\![1;\clubsuit]\!]_= = (?/\clubsuit, ?/\clubsuit)$, $[\![1;\heartsuit]\!]_= = (?/\heartsuit, ?/\heartsuit)$, $[\![0;\clubsuit]\!]_= = [\![0;\heartsuit]\!]_= = (?/\clubsuit, ?/\heartsuit)$.

7.5. The Case of Randomized Input Encodings

Recall that our definition of the start state in (7.1)

$$\mu_{v_0}(s) = \sum_{b, r_\mathrm{i} \text{ with } s = [\![b; r_\mathrm{i}]\!]_\mathrm{i} \,\|\, H} \Pr[R_\mathrm{i} = r_\mathrm{i}] \cdot X_b.$$

allowed for multiple sequences to be possible for one input $b \in \{0,1\}^k$. As for security we already require $R_\mathrm{i} \perp \mathsf{val}_\mathrm{i}(I)$, i.e., $\Pr[R_\mathrm{i} = r_\mathrm{i} \mid \mathsf{val}_\mathrm{i}(I) = b] = \Pr[R_\mathrm{i} = r_\mathrm{i}]$, we do not introduce R_i conditioned on b. For the discussion that follows, it is useful to introduce the term $\Pr[R_\mathrm{i} = r_\mathrm{i}]$ as an additional variable $R_{r_\mathrm{i}} := \Pr[R_\mathrm{i} = r_\mathrm{i}]$. The security then should hold for any probability distribution on the input randomness, as long as it is independent of the inputs.

This section will give two examples of protocols with randomized input encoding and will shed some light on this case. Let us begin with an example that is given an input bit either in basis $\{?/\clubsuit, ?/\heartsuit\}$ or in basis $\{?/\spadesuit, ?/\diamondsuit\}$ in encoding $[\![\cdot]\!]_>$ assuming $\spadesuit < \diamondsuit$, and should output the same bit securely in fixed basis $\{?/\clubsuit, ?/\heartsuit\}$. It is given in Figure 7.13. Observe that the probabilities for the different branches in the turn action may depend

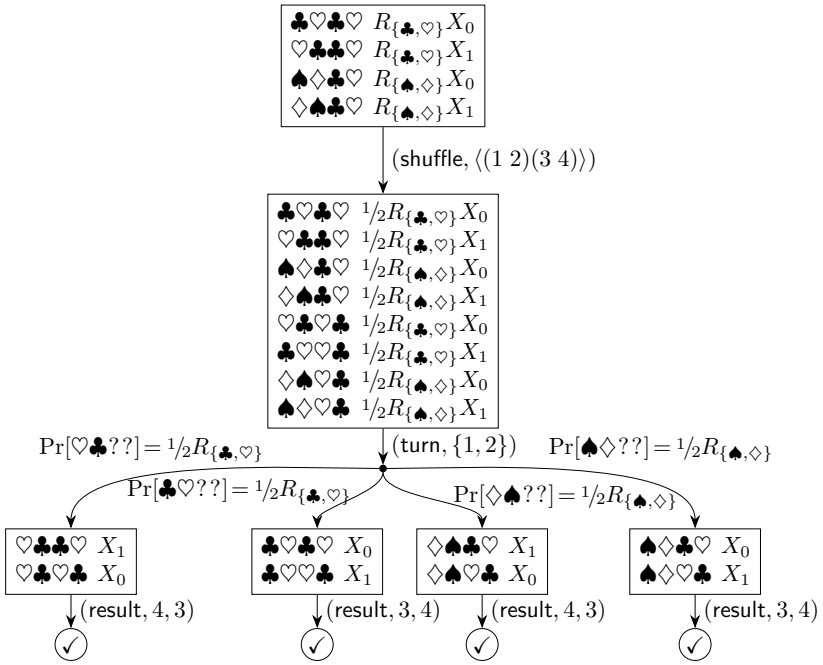


Figure 7.13.: A base convert protocol where the input bit is given either in basis $\{{}^{?}/_{♣}, {}^{?}/_{♡}\}$ or $\{{}^{?}/_{♠}, {}^{?}/_{♢}\}$ and converts it into a bit in fixed basis $\{{}^{?}/_{♣}, {}^{?}/_{♡}\}$. In this process the input basis is leaked, as the cards are turned, which is allowed for secure protocols.

on R_{r_i}, as evidence on the used input randomness is allowed to be contained in V in secure protocols, that are not full-sequence secure.

As a second example, we want give an interpretation of a five-card Las Vegas AND protocol in committed format due to Abe et al. [AHMS18], as a sequential composition of two protocols: First the protocol starts with the five-card trick, given on the left of Figure 7.11 as a state diagram. The output encoding is then $[\![\cdot]\!]_{♡♡}$ as in (6.1) and uses $\mathbb{Z}/5\mathbb{Z}$ as randomness domain. The second part of the protocol then converts the output from this encoding to the standard encoding $[\![\cdot]\!]_{>}$ in a Las Vegas fashion. It is given in Figure 7.14.

Note that this protocol suffix has additional preconditions on the distribution of R_i (apart from it being independent of the input), in order to guarantee the security of the protocol. Here, $R_0 = R_1 = R_2$, which holds for outputs directly coming from the five-card trick, as there we even have $R_i = {}^1/_5$ for all $i = 0, \ldots, 4$. If these preconditions were violated, there may be a way to establish them by an additional randomization. In our case, a random cut with permutation group $\Pi = \langle(1\ 2\ 3\ 4\ 5)\rangle$ would suffice.

On Full-Sequence Security

For full-sequence security, if the input encoding is not deterministic, we want that at any turn the probability of the respective visible sequences should additionally be independent of the R_{r_i}. If one wants to avoid handling two variables (X and R), it is

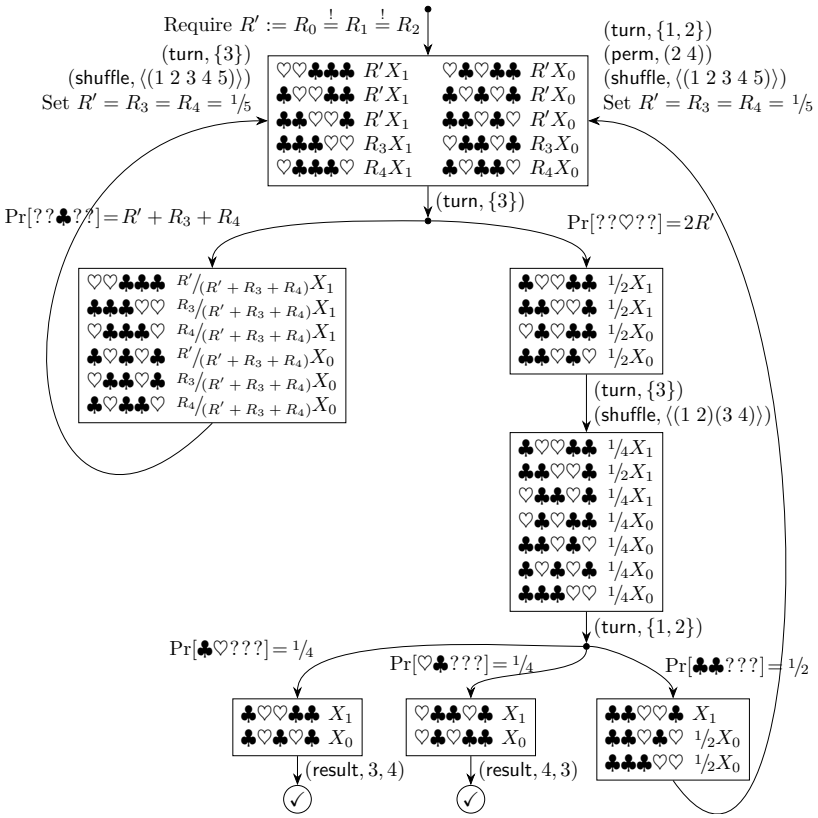


Figure 7.14.: Protocol to convert the output encoding $[\![\cdot]\!]_{\heartsuit\heartsuit}$ of den Boer's protocol (cf. Figure 7.11, left) to the standard encoding $[\![\cdot]\!]_{>}$. For security of the protocol, we require $R_0 = R_1 = R_2$, which is the case if comes directly out of the five-card trick protocol. If one appends this protocol directly to den Boer, the total protocol is a slightly modified variant of the protocol by [AHMS18]. Note that the total protocol only uses random cuts and random bisection cuts.

better to use variables X_Γ for $\Gamma \in U$ instead of X_b and then ensure the above security conditions for states with with variables in this set. This will completely protect the input sequence from leaking information in the visible sequence trace. For the outputs, we refer to the discussion of result actions on page 68.

7.6. On the Choice of Cards for Input and Output

This section is mainly taken from [KSK19]. We essentially show that the choice of (deterministic) input bases (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. Throughout this section we use the standard encoding $[\![\cdot]\!]_{>}$.

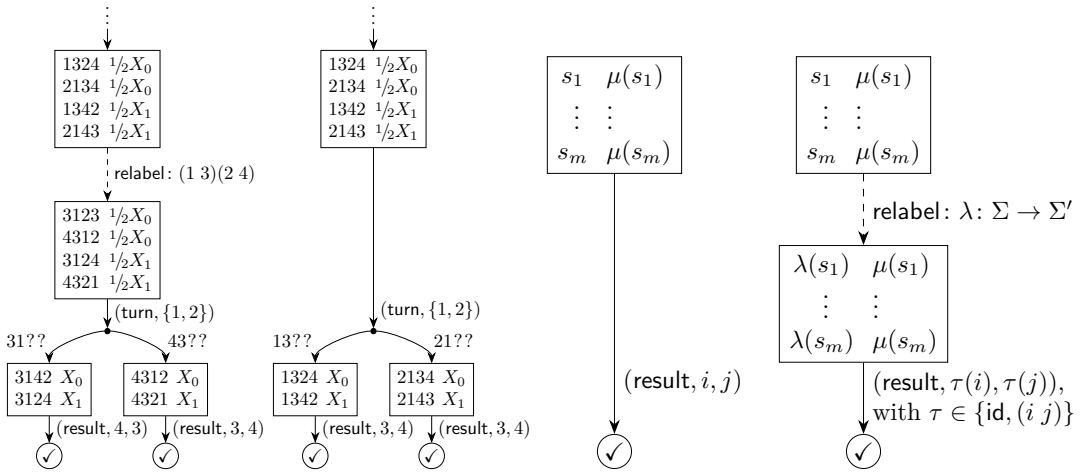


Figure 7.15.: The *left* side shows an example of the relabel action, swapping the card symbols of 1 and 3, and of 2 and 4, respectively. This action is for abbreviated writing only, it does not actually relabel the physical cards, which seems impossible without learning their symbols. Hence, the tree on the very left is virtually translated to one on its 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. On the *right* we have 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.

In standard deck protocols, the protocol description usually specifies Alice's cards to be of symbols $1, 2$, and Bob's to be of symbols $3, 4$. 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 $\lambda\colon \Sigma \to \Sigma'$. (In case of the decks being a subset of $\mathbb{N}$, we may use usual permutation notation.) 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, and a relabeling of a protocol/state (sub)tree is the relabeling of all its states as described by Figure 7.15.

We say that a leaf state μ_v *has deterministic output basis* $\{c_1, c_2\}$, if $R_{\{c_1,c_2\}} = 1$ and $R_x = 0$ for all $x \neq \{c_1, c_2\}$. Similarly, a protocol $\mathcal{P}$ *has deterministic output bases* if all its leaf states have a deterministic output basis. (In secure protocols without full-sequence security these may differ.)

Lemma 7.1. *If $\mathcal{P}$ is a committed format protocol with deterministic output bases, one can relabel the cards without affecting the functioning.*

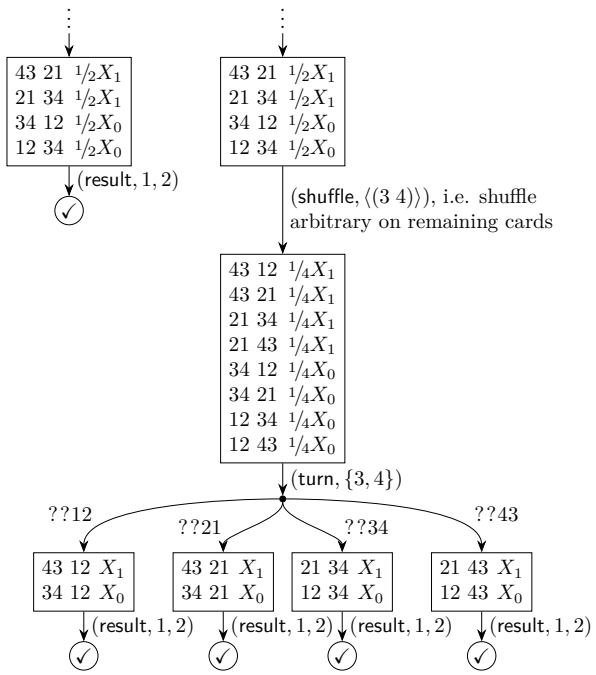


Figure 7.16.: Example of making the basis deterministic, cf. Lemma 7.2. On the left you see a tree part with one-bit output and randomized basis, i.e., the output basis may be $\{{}^{?}/_{1}, {}^{?}/_{2}\}$ or $\{{}^{?}/_{3}, {}^{?}/_{4}\}$, each with a probability of $1/2$. The situation is similar as in Figure 7.11 on the right. We can make the output basis known by splitting up the state via an S_k-shuffle (here: $k = 2$) on the remaining cards (erasing their order), turning them and then doing result. By what is visible in the turn, one can derive the output basis.

Note that the deterministic output basis restriction is important, because if we have a randomized output encoding such as in Figure 7.16 on the left, a relabeling might affect the monotonicity of the encoding of only some of the possible output bases. In this case, we make use of the following lemma, which is illustrated by the mapping of Figure 7.16.

Lemma 7.2. *Every secure protocol with one-bit output can be transformed into a protocol with deterministic output bases, by inserting one (S_k-)shuffle and one turn action before any result operation at a leaf state that does not have a deterministic output basis.*

7.7. Conclusion

Apart from the impossibility for perfect copy of a single card in [MS14a], we are the first to give impossibility results for (committed-format) AND and COPY protocols in this thesis. This may be because of the sparsity of good ways to speak about card-based

protocols. We believe to have overcome this problem by introducing the state diagram formalism in this chapter.

The utility of the state diagram formalism is evident not only by its compact way of representing protocols, but most importantly by the fact that one can read it as a security and correctness proof for the represented protocol. In this thesis, we extended the version as published in [KWH15; KKW+17] to randomized input and output encodings and discussed security in the context of our strengthened formulation of the previous chapter.

Let us point out that state diagrams can also be interpreted as a transition graph of a homogeneous Markov chain, with the probabilities of the edges constituting the probability to switch to this state in the next step of the corresponding stochastic process. For this, we want the initial distribution to be concentrated on the start state and we can view the leaf states as absorbing states, i.e., as states that cannot be left again. Then, the expected running time of a protocol can also be seen as the stopping time of reaching an absorbing state. (States from which one cannot reach an absorbing state anymore will clearly lead to non-termination. If the protocol has at least one absorbing state which is reachable from the start state, we can repair the corresponding protocol by replacing the prescribed action at the state in question by a restart.)

Hence, we only recommend against using state diagrams in cases where one uses a lot of cards or have relatively long protocols which have a short natural-language description. (In these cases, drawing the diagram can be quite cumbersome.) Given, that state diagrams have found quite some adoption in the literature, and their good qualities, we will give (almost) all card-based protocols of the next chapter as a state diagram. We hope these will aid to understand "what is going on" during a protocol run.

8. New Card-based Protocols

In this chapter, we present several new card-based protocols in committed format, which need less cards than the protocols known before, and which are card-minimal w.r.t. certain requirements to the running time behavior and shuffle restrictions, as will be shown in Chapter 9.

In Section 8.1, we present two protocols for AND using only four cards in the two-color deck setting. These are restart-free Las Vegas and include shuffles that are only uniform or only closed, but not uniform closed. Section 8.2 contains two AND protocols, similar to the previous protocols, but which use one additional card to achieve a finite running time behavior. We present a two-color deck protocol for any k-ary Boolean function using only $2k$ cards in Section 8.3, but which exhibits a large restarting probability and makes use of non-closed shuffles, hence it is a merely theoretical result. Section 8.4 covers the case of a standard deck, where we present a new four-card AND protocol which is restart-free Las Vegas and uses only random cuts (which are uniform closed). Protocols for exchanging the basis of an encoding are given in Section 8.5. Finally, Section 8.6 describes a sub-protocol which allows to formulate many protocols from the literature as one or very few invocations of this protocol. It generalizes an idea by [HSN+17] and allows to apply a permutation "stored" in a card sequence, to a set of cards, by a "coupled sorting" mechanism.

8.1. Two-Color Four-Card AND Protocols

This section is based on [KWH15; K18]. We present two secure protocols to compute AND on two bits in committed format and without restarts. An algorithmic description is given in Protocol 8.1 and a representation as a state diagram (cf. Chapter 7), from which correctness and privacy can be deduced, is given in Figure 8.1. Note that we use a "placeholder" for two shuffle steps, with parameters $\Pi_1, \mathcal{F}_1$ and $\Pi_2, \mathcal{F}_2$, respectively, which are later instantiated in (8.1) and (8.2). This is because both protocols have the same formal states in their state diagram, but differ only in the prescribed shuffles, which hence have the same effect on the state they are applied, but the first version is closed (but not uniform) and the second version is uniform (but not closed).

Note that the first (closed-shuffle) variant was constructed in [KWH15], and the second (uniform-shuffle) variant as a variation of the first was constructed later in [K18]. This second variant was concurrently and independently discovered in [RI19].

Observe that the state diagram contains a cycle, i.e., it is possible to return to a state that was encountered before. This implies that the protocol is only Las Vegas. However, on the cycle there are two turn operations each of which have a chance of $1/3$ to yield a final state and therefore leave the cycle. The probability to return to a state on the

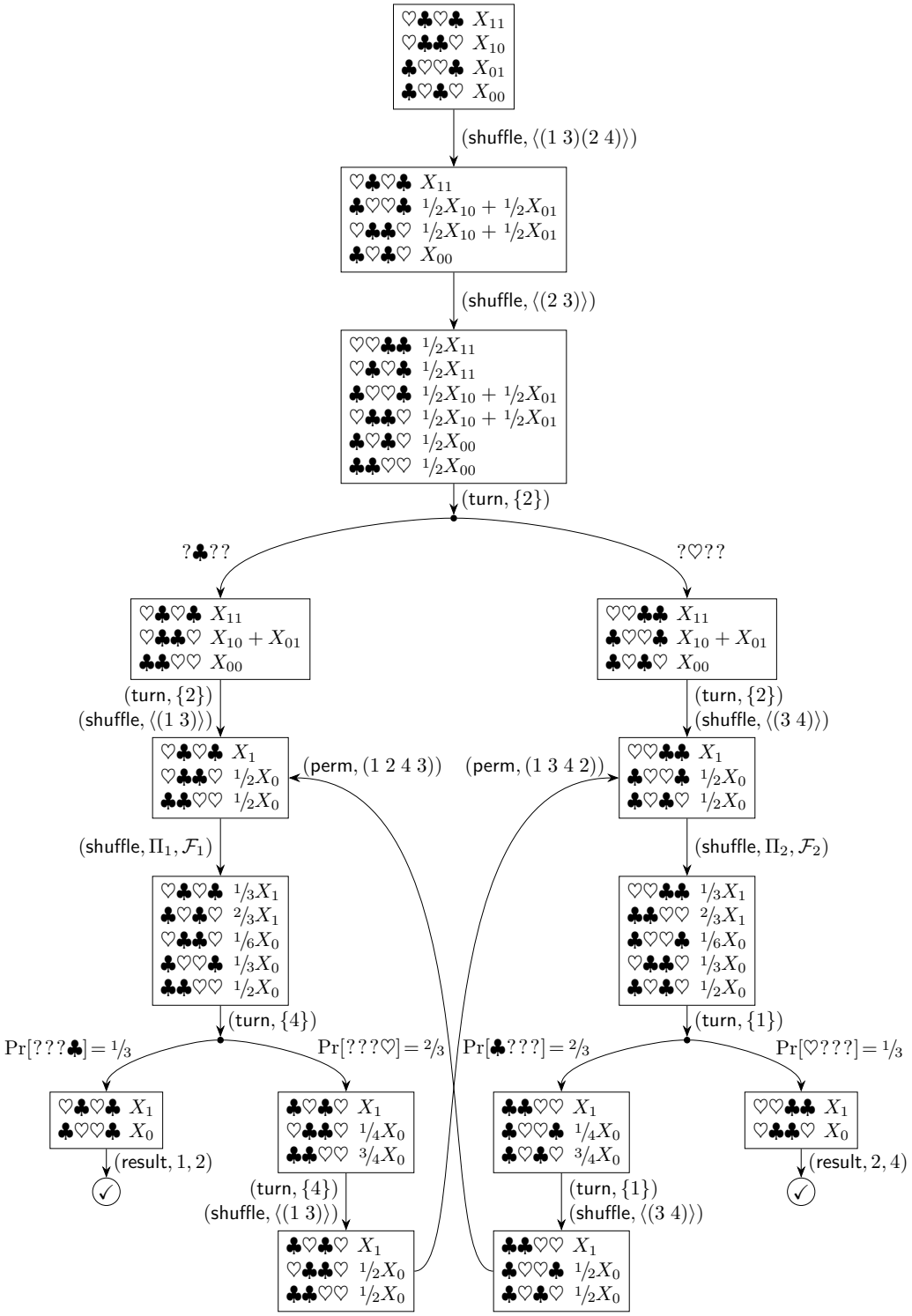


Figure 8.1.: The four-card Las Vegas AND protocol without restart operations from Protocol 8.1. The placeholders Π_i, $\mathcal{F}_i$ are given in (8.1) and (8.2). We use $X_1 \coloneqq X_{11}$ and $X_0 \coloneqq X_{00} + X_{10} + X_{01}$.

```
(shuffle, {id, (1 3)(2 4)})
(shuffle, {id, (2 3)})
(turn, {2})
if v = (?, ♣, ?, ?) then
    (turn, {2}) // turn back
    (shuffle, {id, (1 3)})
1   (shuffle, Π1, F1)
    (turn, {4})
    if v = (?, ?, ?, ♣) then
        (result, 1, 2)
    else if v = (?, ?, ?, ♡) then
        (turn, {4}) // turn back
        (shuffle, {id, (1 3)})
        (perm, (1 3 4 2))
        goto 2
else if v = (?, ♡, ?, ?) then
    (turn, {2}) // turn back
    (shuffle, {id, (3 4)})
2   (shuffle, Π2, F2)
    (turn, {1})
    if v = (♡, ?, ?, ?) then
        (result, 2, 4)
    else if v = (♣, ?, ?, ?) then
        (turn, {1}) // turn back
        (shuffle, {id, (3 4)})
        (perm, (1 2 4 3))
        goto 1
```

Protocol 8.1. Protocol to compute AND using four cards. Note that, because of the **goto** operations, no bound on the number of steps can be given. The placeholders Π_i, $\mathcal{F}_i$ are given in (8.1) and (8.2).

cycle is therefore $(2/3)^2 = 4/9$ and the probability to take the cycle k times is $(4/9)^k$. The expected number of times the cycle is taken is therefore $\sum_{k\geq 0}(4/9)^k = (1-4/9)^{-1} = 9/5$. In particular, the expected running time of the protocol is bounded. We summarize our result in the following theorem.

Theorem 8.1. *There is a secure restart-free Las Vegas protocol to compute AND on two bits in committed format using four cards in the two-color deck setting and using only closed shuffles.*

Proof. Figure 8.1, is a correct and secure protocol for

$$\begin{aligned} \Pi_1 &:= \langle (1\ 2)(3\ 4)\rangle, \quad \mathcal{F}_1\colon \mathsf{id} \mapsto 1/3, (1\ 2)(3\ 4) \mapsto 2/3, \\ \Pi_2 &:= \langle (1\ 3)(2\ 4)\rangle, \quad \mathcal{F}_2\colon \mathsf{id} \mapsto 1/3, (1\ 3)(2\ 4) \mapsto 2/3. \end{aligned} \tag{8.1}$$

While these shuffles are only closed, all other shuffles are uniform closed. □

The complex shuffles used here, have been implemented with sliding cover boxes by Nishimura et al. [NHMS16]. As discussed, we also have a uniform-shuffle variant:

Theorem 8.2. *There is a secure restart-free Las Vegas protocol for AND using four cards in the two-color deck setting and using only uniform shuffles.*

Proof. Figure 8.1, is a correct and secure protocol for

$$\Pi_1 \coloneqq \{\mathsf{id}, (1\ 2)(3\ 4), (4\ 3\ 2\ 1)\}, \quad \Pi_2 \coloneqq \{\mathsf{id}, (1\ 3)(2\ 4), (1\ 3\ 2\ 4)\}, \tag{8.2}$$

and $\mathcal{F}_1$ and $\mathcal{F}_2$ being the uniform distributions on Π_1, Π_2, respectively. While these shuffles are only uniform, all other shuffles used in the protocol are uniform closed. □

8.2. Finite-Runtime Two-Color Five-Card AND Protocols

This section is based on [KWH15; K18]. The two protocols presented here are very similar to the protocols of the previous section. Upon close observation of Figure 8.2, one discovers that for the most part – namely, for all steps except starting from the third state in the left (♣-)branch of the protocol – the only difference to the previous protocols is that the fifth card, with loss of generality a ♡, is put next to the cards as a helping card.

We make use of the fifth card just then to "break out" of the cycle of the four card protocols. This yields a protocol with finite running time and at most 13 steps in every execution. An algorithmic description is given in Protocol 8.2 and a representation as a state diagram is given in Figure 8.2. We summarize our result in the following theorem.

Theorem 8.3. *There is a secure finite-runtime protocol to compute AND on two bits in committed format using five cards in the two-color deck setting.*

Proof. Figure 8.2, is a correct and secure protocol for

$$\begin{aligned} &\Pi_1 \coloneqq \{\mathsf{id}, (5\ 4\ 3\ 2\ 1)\}, && \mathcal{F}_1\colon \mathsf{id} \mapsto {}^2\!/_3, (5\ 4\ 3\ 2\ 1) \mapsto {}^1\!/_3 \\ &\Pi_2 \coloneqq \langle (1\ 3)(2\ 4) \rangle, && \mathcal{F}_2\colon \mathsf{id} \mapsto {}^1\!/_3, (1\ 3)(2\ 4) \mapsto {}^2\!/_3. \end{aligned} \tag{8.3}$$

While Π_1 is neither closed nor uniform, Π_2 is closed but non-uniform and all other shuffles used in the protocol are uniform closed. □

The complex shuffles used here have been implemented with sliding cover boxes by Nishimura et al. [NHMS16].

As discussed before, a large part of the protocol is as in the four-card protocol, hence we can apply the same replacement of the shuffle to obtain a uniform shuffle in the right (♡-)branch of the protocol. What is more, we also propose a modification of the non-closed non-uniform shuffle in the left branch, to achieve a uniform probability distribution for the permutations in the shuffle. This is given in the following:

Theorem 8.4. *There is a secure five-card finite-runtime protocol for AND using only uniform shuffles.*

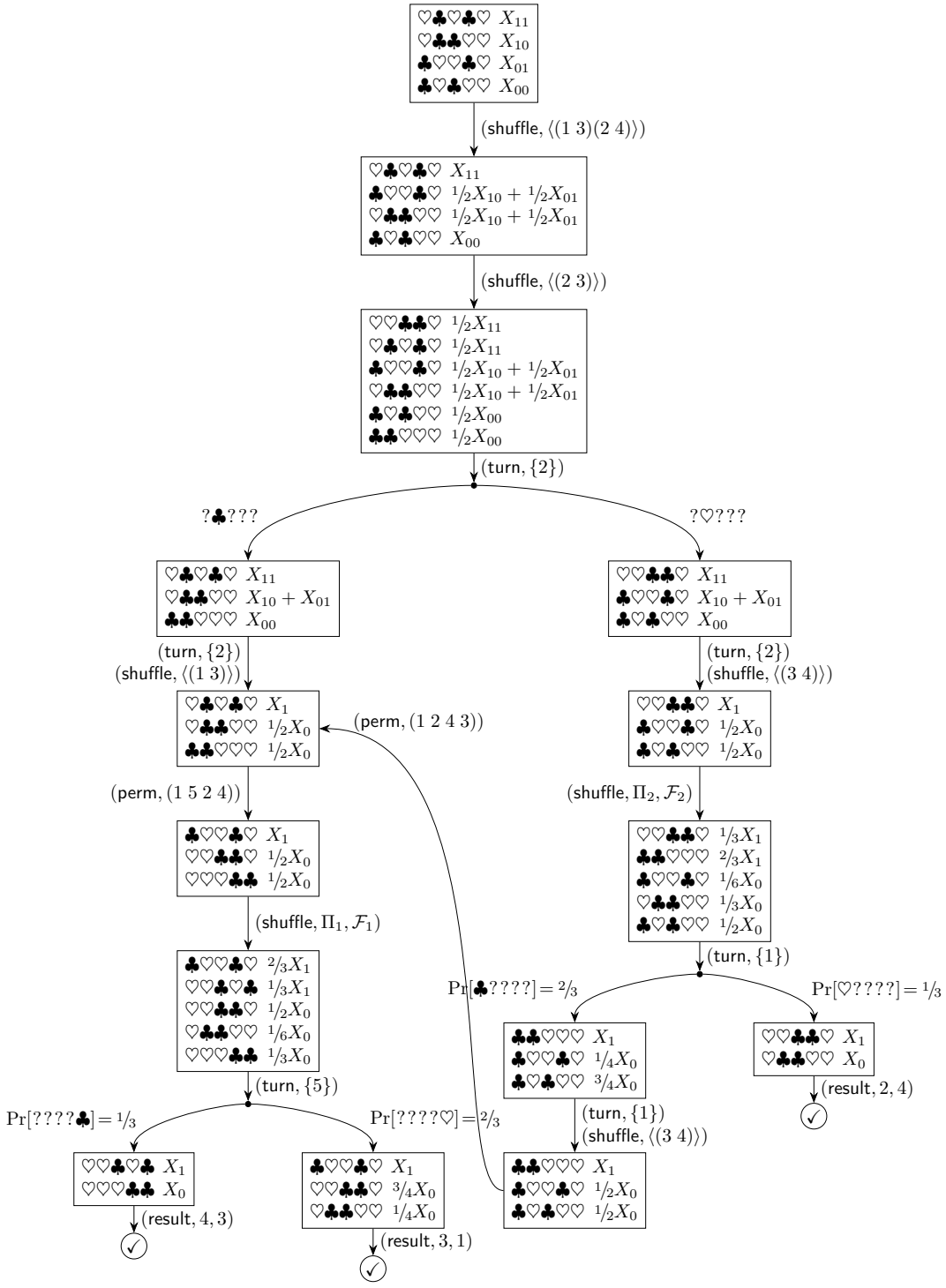


Figure 8.2.: The five-card finite-runtime AND protocol. Note that that fifth card allows to "break out" of the cycle of the previously seen four-card Las Vegas AND protocol. The placeholders Π_i, $\mathcal{F}_i$ are given in (8.3) and (8.4).

```
(shuffle, {id, (1 3)(2 4)})
(shuffle, {id, (2 3)})
(turn, {2})
if v = (?, ♣, ?, ?, ?) then
    (turn, {2}) // turn back
    (shuffle, {id, (1 3)})
⋆   (perm, (1 5 2 4)) // sort in the fifth card
    (shuffle, Π1, F1)
    (turn, {5})
    if v = (?, ?, ?, ?, ♣) then
        (result, 4, 3)
    else if v = (?, ?, ?, ?, ♡) then
        (result, 3, 1)
else if v = (?, ♡, ?, ?, ?) then
    (turn, {2}) // turn back
    (shuffle, {id, (3 4)})
    (shuffle, Π2, F2)
    (turn, {1})
    if v = (♡, ?, ?, ?, ?) then
        (result, 2, 4)
    else if v = (♣, ?, ?, ?, ?) then
        (turn, {1}) // turn back
        (shuffle, {id, (3 4)})
        (perm, (1 2 4 3))
        goto ⋆
```

Protocol 8.2. A five-card finite-runtime AND protocol. It proceeds as in Protocol 8.1 (ignoring card 5) until reaching the line marked as 1, when instead of executing the line, an alternative path is taken using the fifth card. The placeholders Π_i, $\mathcal{F}_i$ are given in (8.3) and (8.4).

Proof. Figure 8.2, is a correct and secure protocol for

$$\Pi_1 \coloneqq \{\mathsf{id}, (3\ 5), (5\ 4\ 3\ 2\ 1)\}, \quad \Pi_2 \coloneqq \{\mathsf{id}, (1\ 3)(2\ 4), (1\ 3\ 2\ 4)\}, \tag{8.4}$$

Π_2 being the same as in (8.2), and $\mathcal{F}_1$ and $\mathcal{F}_2$ being the uniform distributions on Π_1, Π_2, respectively. While these shuffles are only uniform, all other shuffles used in the protocol are uniform closed. □

For an interpretation note that added (3 5) in the shuffle takes $1/3$ of the probability that has previously been assigned to id, but has the same effect as the latter, due to the structure of the state on which the shuffle is applied.

As in the previous section note that the first variant was constructed in [KWH15], and the second (uniform-shuffle) variant as a variation of the first was constructed later in [K18]. This second variant was concurrently and independently discovered in [RI19].

8.3. A $2k$-Card Protocol for any k-ary Boolean Function

This section is taken from [KWH15]. We present a protocol to compute a k-ary Boolean function with $2k$ cards and success probability 2^{-k} in three steps: One shuffle, one turn and one result or restart action. The "hard work" is done in an "irregularly complex" shuffle operation, posing practical problems as discussed in Section 6.7.

Theorem 8.5. *For any Boolean function* $f\colon \{0,1\}^k \to \{0,1\}$ *there is a secure Las Vegas protocol in committed format using* $2k$ *cards. The expected number of* restart *actions in a run is* $2^k - 1$.

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

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

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

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

We can now describe the steps of our protocol:

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

2. $(\mathsf{turn}, \{1, 4, 6, 8, \ldots, 2k\})$, i.e., turn over the first card and all cards with even indices except 2.

3. If the turn revealed ♣ in position 1 and ♡ everywhere else, i.e., the visible sequence is $(\clubsuit, ?, ?, \heartsuit, ?, \heartsuit, \ldots, ?, \heartsuit)$, then perform $(\mathsf{result}, 2, 3)$. Otherwise, $(\mathsf{restart})$.

For a deeper understanding of what is actually going on, we suggest contemplating on Figure 8.3 (which is, admittedly, somewhat intimidating), but correctness and security are surprisingly easy to show directly:

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

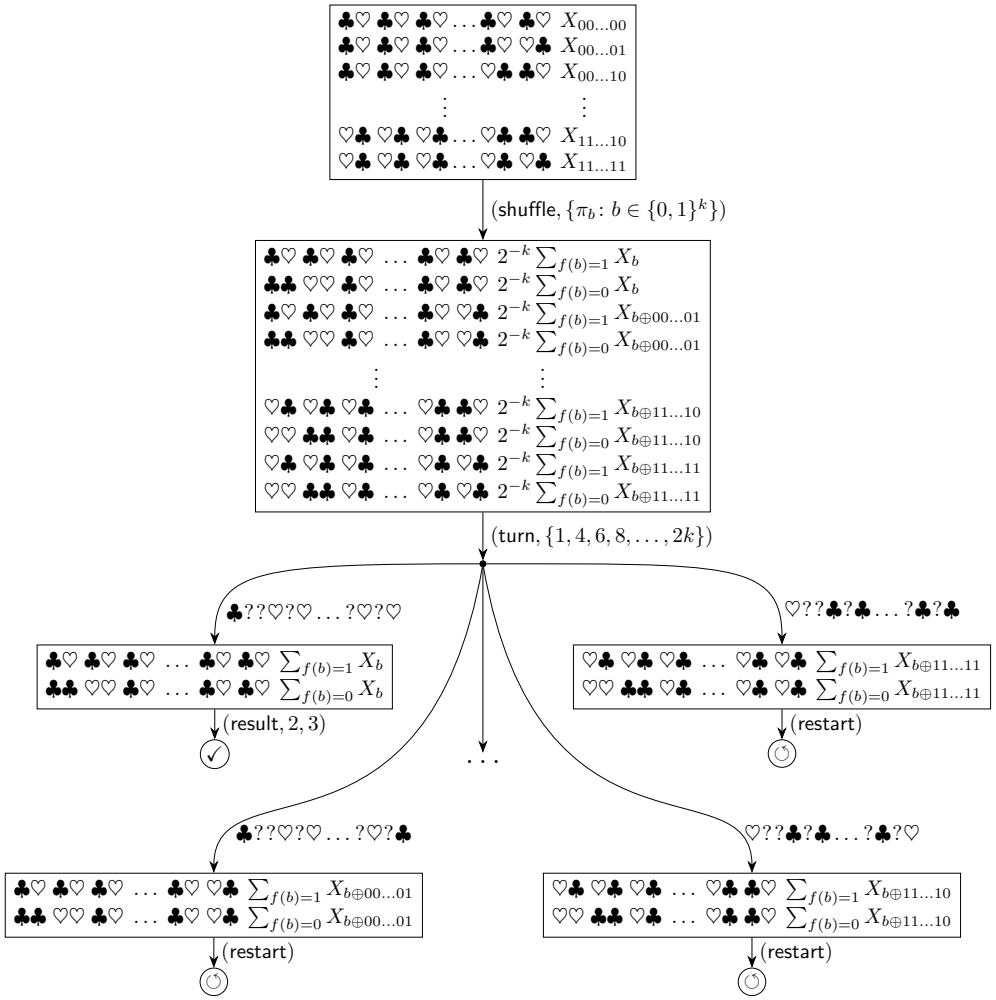


Figure 8.3.: The $2k$-card protocol for an arbitrary Boolean function f of Theorem 8.5. We use the notation $b_1 \oplus b_2$ to denote the bitwise exclusive-or operation.

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

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

8.4. AND Protocols with a Standard Deck

This section is directly taken from [KSK19] with slight modifications, and contains a new AND protocol in the standard deck setting using only four cards. As it is similar in spirit to the five-card protocol by Niemi and Renvall [NR99], we want to start with a description of their protocol, to be later able to discuss the differences. We follow the interpretation of Niemi and Renvall's protocol from [M16b]. The protocol uses five cards with distinguishable symbols, which we denote as $\boxed{1}\,\boxed{2}\,\boxed{3}\,\boxed{4}$ and $\boxed{5}$ for simplicity. Let us quickly recall the standard encoding in this standard deck setting, cf. Section 6.4: $[\![\cdot]\!]_>$ encode by the order of two cards $\boxed{i}\,\boxed{j}$, $i, j \in \{1, \ldots, 5\}$, via

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

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 but with the cards $\boxed{3}\,\boxed{4}$. The protocol needs an additional helping card, $\boxed{5}$, which is put to the right of the players' cards.

We start by swapping Alice's second card with Bob's first card in the card sequence (pile) on the table. This 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 $\boxed{3}$ (to be explained below), one directly obtains the output. To see this, let us look at the different cases:

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

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, unknown to the players. We can then securely turn the first card and if it is $\boxed{2}$ or $\boxed{3}$, remove it. Because of the cut, it is random which card is turned and, hence, it does not give anything away about the inputs. When both are removed, we get to a configuration where $\boxed{5}$ is the first card by the same process with the two remaining cards encoding the AND result. A formal version of this protocol is described in Protocol 8.3 and Figure 8.4.

In this section, we show that one can do away with the helping card $\boxed{5}$, by presenting a protocol using only four cards.

Theorem 8.6. *There is a four-card Las Vegas AND protocol with deck $\mathcal{D} = [1, 2, 3, 4]$ using only random cuts.*

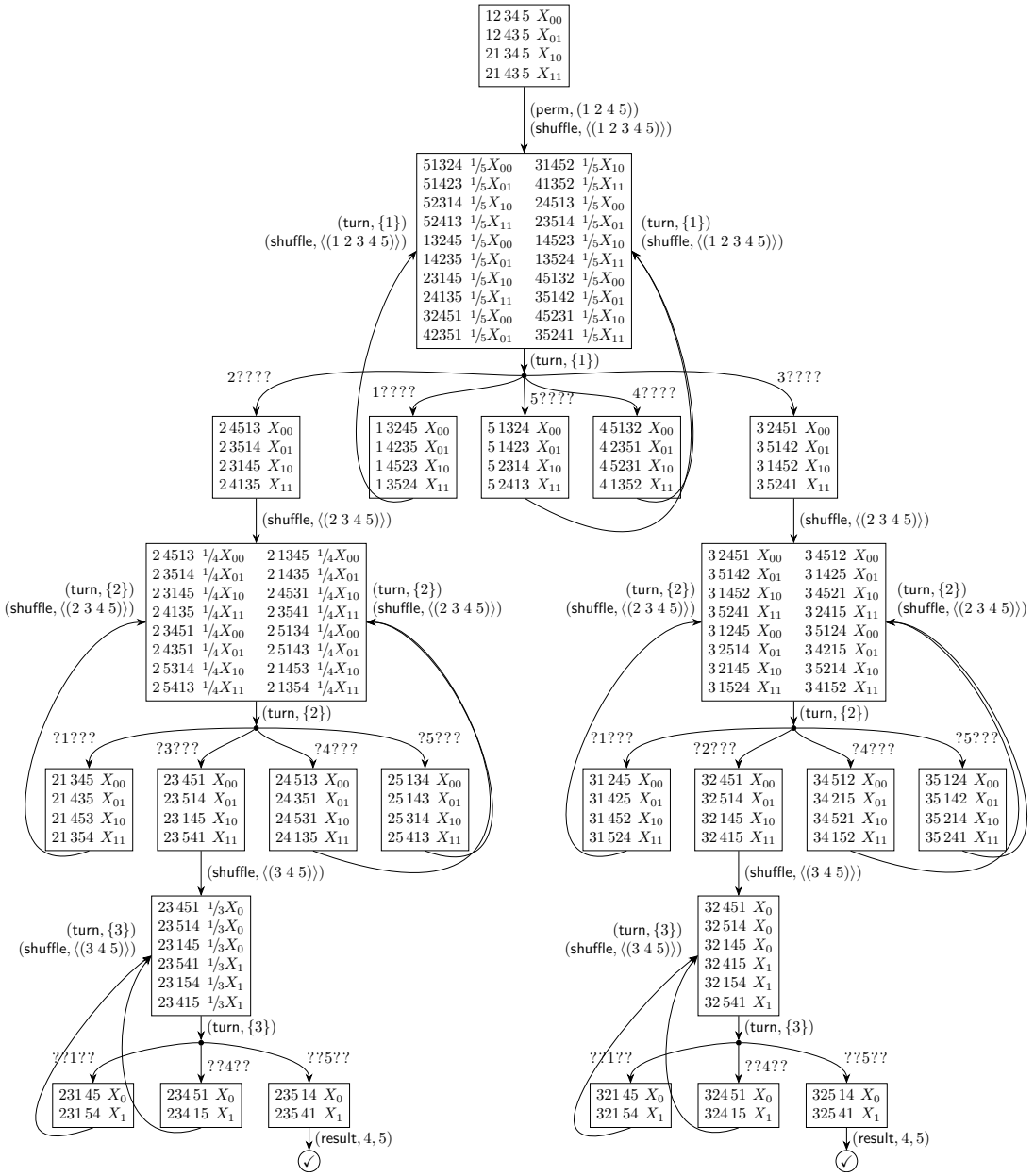


Figure 8.4.: The five-card Las Vegas AND protocol of [NR99] with deck $\mathcal{D} = [1, 2, 3, 4, 5]$ using only random cuts, cf. Protocol 8.3. Note that $X_0 := X_{00}+X_{01}+X_{10}$ and $X_1 := X_{11}$. The output is in basis $\{1, 4\}$.

(perm, (1 2 4 5))
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)

Protocol 8.3. Protocol to compute AND in committed format using five cards by Niemi and Renvall [NR99], with slight modifications. The first bit is given in basis $\{1, 2\}$, the second in basis $\{3, 4\}$. The output basis is $\{1, 4\}$. See also Figure 8.4 for the state diagram of the protocol.

Proof. See Figure 8.5 and Protocol 8.4 and observe that in the leaf states of the state diagram in Figure 8.5 the encoding is unique given the visible sequence trace. □

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 $\boxed{1}$, the other cases are analogous. In this situation, let us look at the different cases, given in Table 8.1.

Table 8.1.: The different states of Protocol 8.4 after a $\boxed{1}$ was revealed in the first turn step. The permutation to be applied in this case is swapping the last two cards. The analysis is similar in all other cases.

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

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 $\boxed{1}$ from the remaining cards, to get the output encoded using the cards $\boxed{2}$ and $\boxed{4}$.

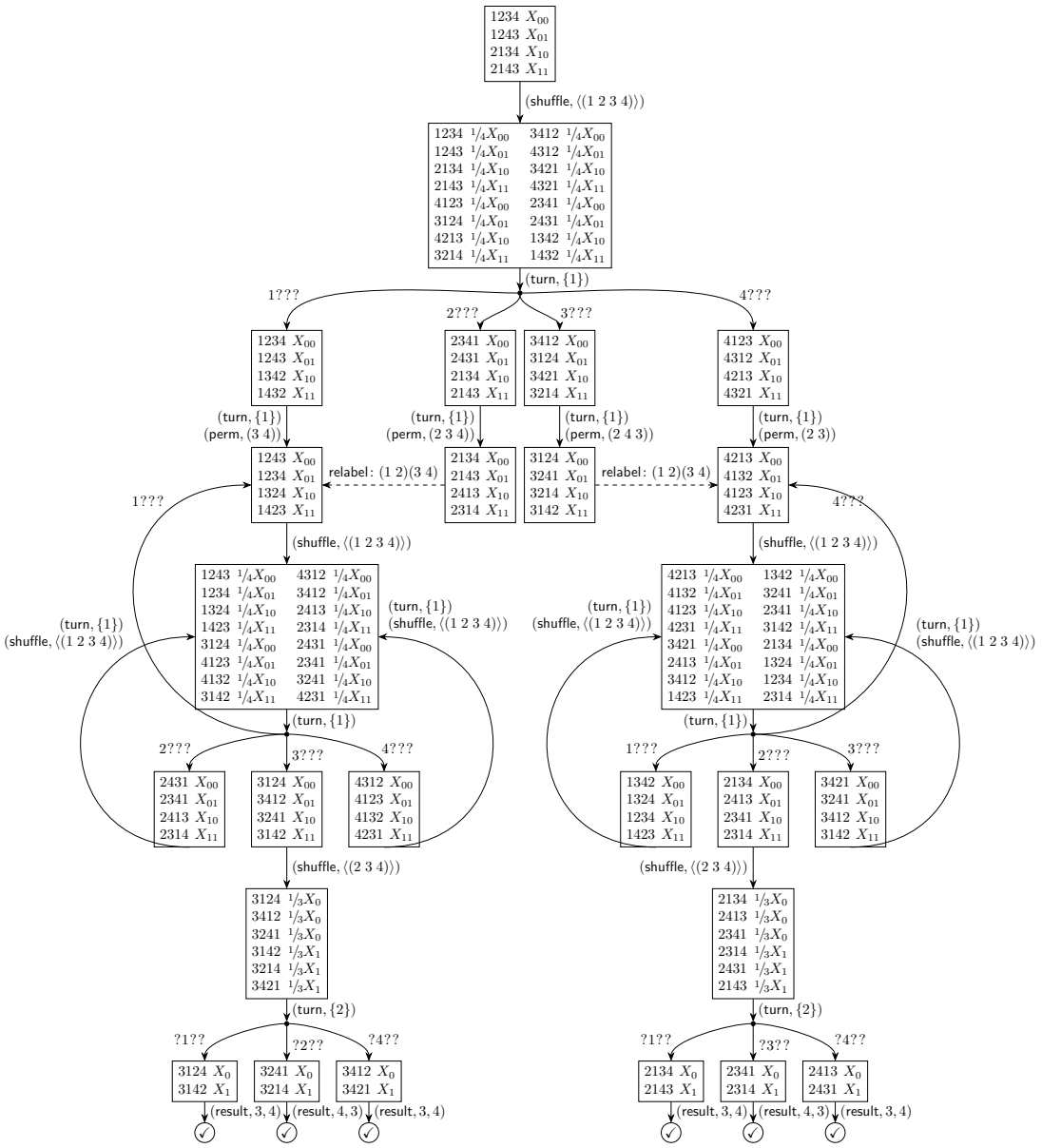


Figure 8.5.: Four-card Las Vegas AND protocol using random cuts, cf. Protocol 8.4. Here, $X_0 \coloneqq X_{00} + X_{01} + X_{10}$ and $X_1 \coloneqq X_{11}$. The relabel operations are not actual actions to be performed but help abbreviate the write-up of the protocol, see Section 7.6.

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 a futile goal for us anyway because in the other cases besides the first turn being 1 it is different anyway). 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 then safely turn any card and, from the resulting symbol, deduce in which order the other cards have to be output to encode the result of the protocol.

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

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

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

Protocol 8.4. Protocol to compute AND in committed format using four cards. The first bit is given in basis $\{1, 2\}$, the second in basis $\{3, 4\}$. The output basis will then be $\{1, 2, 3, 4\} \setminus \{v_2, v_3\}$, where v_2, v_3 are the last two revealed symbols. See also Figure 8.5 for the state diagram of the protocol.

For an analysis of the number of shuffle steps in the protocol, observe that we have perform 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]

As Niemi and Renvall [NR99] state, their running time in the number of shuffle steps is calculated as follows: Their protocol starts with a shuffle and repeats it with probability $3/5$. The second loop contains a shuffle with a repeating probability of $3/4$. The shuffle

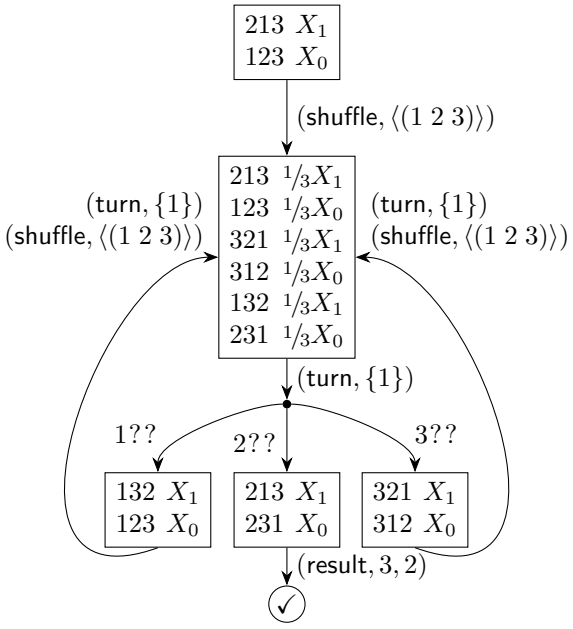


Figure 8.6.: A three-card Las Vegas basis convert protocol for deck $\mathcal{D} = [1, 2, 3]$ with uniform closed shuffles.

in the final loop is repeated with probability $\frac{2}{3}$. In total this gives an expected running time of $3+\sum_{n=1}^{\infty}\left(\frac{3}{5}\right)^n+\sum_{n=1}^{\infty}\left(\frac{3}{4}\right)^n+\sum_{n=1}^{\infty}\left(\frac{2}{3}\right)^n = 3+1.5+3+2 = 9.5$. However, note that 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 (it is not full-sequence secure). In this case, the modified Niemi–Renvall protocol has an expected number of $3 + 1.5 + 3 = 7.5$ shuffle steps. Therefore, our four-card AND protocol needs one card less and outperforms the Niemi–Renvall protocol by an expected number of 1.5 shuffle operations.

8.5. Base Conversion Protocols for Overlapping Bases

In this section, we give two card-minimal 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 not been explicitly discussed 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.

Theorem 8.7. *There is a three-card Las Vegas base conversion protocol for overlapping bases with deck* $\mathcal{D} = [1, 2, 3]$ *and uniform closed shuffles.*

Proof. See Figure 8.6 and Protocol 8.5. □

repeat
  $(\mathsf{shuffle}, \langle(1\ 2\ 3)\rangle)$
  $v := (\mathsf{turn}, \{1\})$
until $v = 2$
$(\mathsf{result}, 3, 2)$

Protocol 8.5. Three-card Las Vegas basis conversion protocol as given in Figure 8.6 with $\mathcal{D} = [1, 2, 3]$, input basis $\{1, 2\}$ and output basis $\{1, 3\}$.

Theorem 8.8. *There is a five-card finite-runtime base 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 applying the base conversion of [M16b] twice, cf. Protocol 8.6. □

$(\mathsf{shuffle}, \langle(1\ 2)(4\ 5)\rangle)$
$v := (\mathsf{turn}, \{1\})$
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 $(\mathsf{result}, 3, 1)$

Protocol 8.6. Five-card finite-runtime base conversion protocol with overlapping bases with $\mathcal{D} = [1, 2, 3, 4, 5]$, input basis $\{1, 2\}$ and output basis $\{1, 3\}$.

8.6. The Coupled Sorting Sub-Protocol

This section is taken from [KW18], with minor modifications. Here, we show that a large subset of the protocols proposed in this thesis and from the literature, can be regarded as a sequence of sub-protocols with basically the same functionality, which we capture under the name "sort protocol". We believe that this observation is of independent interest. We also show that, under weak assumptions, protocols obtained as compositions of sort-protocols are secure. This elegantly re-proves the security of existing protocols and greatly simplifies the security proofs of our own protocols.

We use the term "coupled" to indicate that the same permutation is applied to multiple card subsequences by forming piles (e.g., to be placed in envelopes) and then permuting them, cf. Figure 8.8.

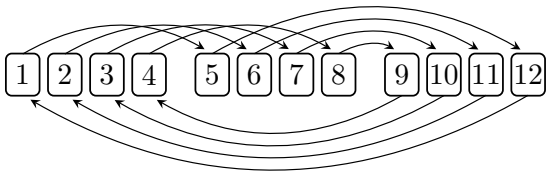


Figure 8.7.: Effect of the permutation $(1\ 2\ 3) \uparrow ((1,2,3,4),(5,6,7,8),(12,11,10,9))$ when applied to a sequence of cards. The idea is to permute the three card sequences in positions $(1,2,3,4)$, $(5,6,7,8)$ and $(12,11,10,9)$ cyclically (as in $(1\ 2\ 3)$), taking the groups of four cards "as a whole", and rearranging them while maintaining the order within the group. We reversed the third sequence to illustrate this possibility.

Notation

Let $\pi \in S_n$, $A = (a_1, \ldots, a_n)$ a sequence of distinct natural numbers and B a sequence of length n. We define the *lift* $\pi \uparrow A$ of π to A via

$$(\pi \uparrow A)(m) \coloneqq \begin{cases} a_{\pi(i)}, & \text{if } m = a_i \text{ for some } i, \\ m, & \text{otherwise.} \end{cases}$$

For instance, the permutation $\pi = (1\ 3)(2\ 4) \in S_4$ lifted to the sequence $A = (5,2,7,8)$ yields the permutation $\pi \uparrow A = (5\ 7)(2\ 8)$. We define the *lift* of a permutation to a *sequence of same-length sequences* $B = ((b_1^1, \ldots, b_1^k), \ldots, (b_n^1, \ldots, b_n^k))$ as:

$$\pi \uparrow B := (\pi \uparrow (b_1^1, \ldots, b_n^1)) \circ \cdots \circ (\pi \uparrow (b_1^k, \ldots, b_n^k)).$$

Note that for each $i \in \{1, \ldots, k\}$, the $(b_1^i, \ldots, b_n^i)$ are again assumed to be distinct. We permit that the b_i^j are sequences again. In this sense this definition is recursive. Figure 8.7 illustrates the simple intuition behind these more complex lifts.

We naturally extend this definition to permutation sets $\Pi \subseteq S_n$ and, for convenience, a lift to two sequences A, B as:

$$\Pi \uparrow A := \{\pi \uparrow A \colon \pi \in \Pi\}, \quad \Pi \uparrow A, B := \{(\pi \uparrow A) \circ (\pi \uparrow B) \colon \pi \in \Pi\}.$$

The Family of Sort (Sub-)Protocols

For each combination of a group of permutations $\Pi \subseteq S_n$, a sequence of (card) positions $A = (a_1, \ldots, a_n)$ and another sequence $B = (b_1, \ldots, b_n)$, we will define a protocol $\mathsf{sort}_\Pi\, A \uparrow B$. Note that Π, A and B are a public part of the protocol specification, not inputs. To describe the intended behavior of the protocol, assume it is executed on a sequence Γ of cards. Let $\mathbb{A} := \Gamma[A] \coloneqq (\Gamma[a_1], \ldots, \Gamma[a_n])$ be the sequence of cards in positions A, and $\mathbb{B} \coloneqq \Gamma[B]$ the sequence of cards in positions B. We assume that these card (symbol) sequences $\mathbb{A}$ and $\mathbb{B}$ are secret.

Let $\pi_{\mathbb{A}} \in \Pi$ be the permutation that *sorts* $\mathbb{A}$, i.e., $\pi_{\mathbb{A}}(\mathbb{A})$ is the lexicographical minimum of $\{\pi(\mathbb{A}) \mid \pi \in \Pi\}$ w.r.t. a given order on the deck symbols[1]. The overall effect of $\mathsf{sort}_\Pi\, A \uparrow B$ should be that $\pi_{\mathbb{A}}$ is applied to both $\mathbb{A}$ and $\mathbb{B}$, yielding a sequence Γ' with $\Gamma'[A] = \pi_{\mathbb{A}}(\mathbb{A})$, $\Gamma'[B] = \pi_{\mathbb{A}}(\mathbb{B})$ and Γ' equal to Γ everywhere else. We permit B, and correspondingly $\mathbb{B}$, to be a sequence of k-element sequences $B = ((b_1^1 \ldots, b_1^k), \ldots, (b_n^1, \ldots, b_n^k))$ for $k \in \mathbb{N}$, in which case applying $\pi_{\mathbb{A}}$ to $\mathbb{B}$ means applying $\pi_{\mathbb{A}}$ to each of the k sequences $\mathbb{B}_1 = \Gamma[(b_1^1, \ldots, b_n^1)]$, ..., $\mathbb{B}_k = \Gamma[(b_1^k, \ldots, b_n^k)]$.

Implementation of Sort Protocols

An example for a practical implementation is given in Figure 8.8 and a formal implementation in Protocol 8.7. The first step applies a randomly chosen permutation $\tau \in \Pi$ to A and B. Then, the cards in positions A are turned over, revealing $\tau(\mathbb{A})$ where $\mathbb{A}$ is the sequence of cards that was previously in positions A.

This allows us to recognize which permutation $\pi_{\tau(\mathbb{A})}$ would sort $\tau(\mathbb{A})$ and apply it to the sequences in positions A and B. Clearly, the overall effect is that $\mathbb{A}$ and $\mathbb{B}$ have both been permuted by the same permutation $\pi_{\tau(\mathbb{A})} \circ \tau$. Moreover this permutation sorted the cards in positions A as desired.

$(\mathsf{shuffle}, \Pi \uparrow A, B)$ `// chooses` $\tau \in \Pi$ `randomly, obliviously applies`
  $\tau \uparrow A, B$
(turn, A) `// reveals` $\tau(\mathbb{A})$
let $\pi_{\tau(\mathbb{A})} \in \Pi$ be the permutation that sorts $\tau(\mathbb{A})$
$(\mathsf{perm}, (\pi_{\tau(\mathbb{A})} \uparrow A) \circ (\pi_{\tau(\mathbb{A})} \uparrow B))$

Protocol 8.7. $\mathsf{sort}_\Pi\, A \uparrow B$.

If we only want to reset the sequence in A to a sorted one, i.e., without applying it to cards at positions B (as in Protocols 11.3 and 11.4), we write $\mathsf{sort}_\Pi\, A$.

Definition 8.1. Let $\mathsf{supp}(A) := \{\Gamma[A] \colon \Gamma \text{ is possible when reaching } \mathsf{sort}_\Pi\, A \uparrow B\}$ be the set of possibilities for $\mathbb{A}$ when the surrounding protocol reaches the occurrence of the sort sub-protocol.

We say an occurrence of a sort sub-protocol $\mathsf{sort}_\Pi\, A \uparrow B$ is *valid* in a surrounding protocol if $\mathsf{supp}(A)$ is contained in an orbit O of the group action of Π on sequences, and $|O| = |\Pi|$.

The rationale behind this definition is that if $\mathsf{supp}(A)$ is subset of O w.r.t. Π, then shuffling $\mathbb{A}$ with Π destroys all information that is held in the sequence $\mathbb{A}$ prior to turning it. Thus, no information is leaked. The condition $|O| = |\Pi|$ ensures that the permutation $\pi_{\mathbb{A}} \in \Pi$ that sorts $\mathbb{A}$ is uniquely defined.[2]

[1] We use the order from $\mathbb{N}$ on cards with natural numbers, and $\clubsuit < \heartsuit$.

[2] We could drop this condition without affecting security. The effect of sort would be that among all permutations that sort $\mathbb{A}$, one is chosen uniformly at random and applied to the cards in A and B.

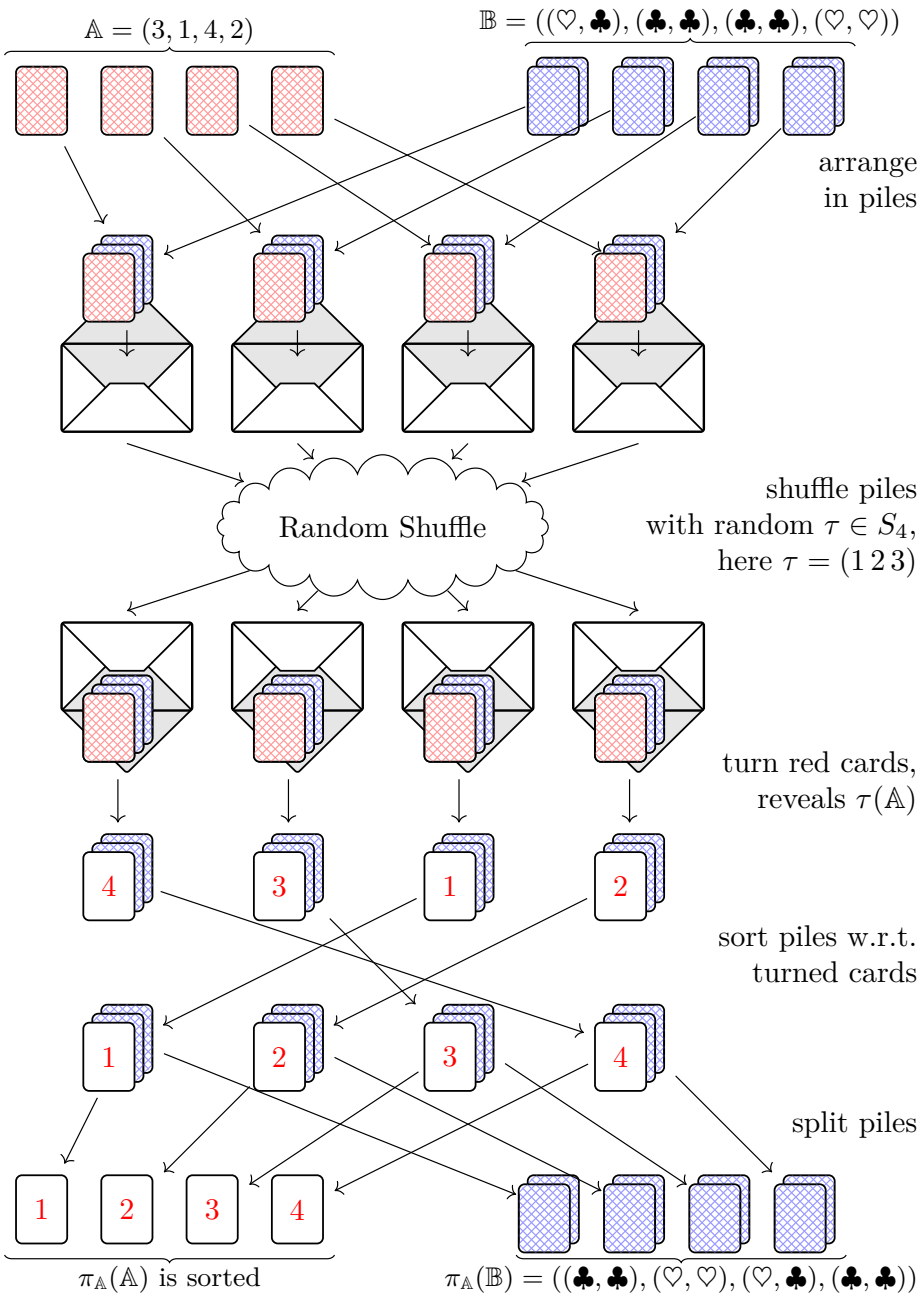


Figure 8.8.: Application of $\mathsf{sort}_{S_4}\, A \uparrow B$ where A denotes the four positions of the red cards and B the four pairs of positions of the blue cards, in canonical ordering. Since the current sequence is $\mathbb{A} = (3, 1, 4, 2)$, the permutation $\pi_{(3,1,4,2)} = \{1 \mapsto 3, 2 \mapsto 1, 3 \mapsto 4, 4 \mapsto 2\}$ is applied to A and B, leaving the red cards sorted and the pairs of blue cards permuted by $\pi_{(3,1,4,2)}$ as shown. Note that the revealed sequence $(4, 3, 1, 2)$ is independent of the input sequences and the output sequence. (The different back colors are for illustration and to avoid errors in handling the cards, but are not necessary in theory.)

Note that this slightly involved criterion is necessary to ensure security in the case that the permutation is chosen at random from a proper subset of S_n. An important example for this is a random cut, which we later use to apply a rotation encoded in a sequence. Assume for instance $\Pi = \langle(1\ 2\ 3)\rangle$ and $\pi \in \Pi$ uniformly random. Moreover, let X be the six-element set of permutations of $(\heartsuit, \clubsuit, \spadesuit)$, and $s \in X$ be arbitrary. Revealing $\pi(s)$ to be, say, $\pi(s) = (\clubsuit, \heartsuit, \spadesuit)$ reveals, e.g., that s is not $(\heartsuit, \clubsuit, \spadesuit)$. The reason is that Π has two orbits when acting on sequences of length 3 with symbols $\heartsuit, \clubsuit, \spadesuit$ and we learn in which orbit we have been, excluding all sequences of the other orbit. That this criterion is also suitable for achieving security is shown by the following lemma.

Lemma 8.1. *If an occurrence of* $\mathsf{sort}_\Pi\, A \uparrow B$ *is valid in a surrounding protocol, then the sequence revealed in the sub-protocol's turn step is independent of the sub-protocol's input- and output-sequences* Γ *and* Γ'*.*

Proof. By definition, $\mathsf{supp}(A)$ is a subset of an orbit O. Whatever the distribution of $\mathbb{A}$ is, if $\pi \in \Pi$ is chosen uniformly at random, then the sequence $\mathbb{A}' = \pi(\mathbb{A})$ revealed in the turn step is uniformly distributed on O. It is thus independent of Γ. Since Γ' is a function of Γ, we conclude that $\mathbb{A}'$ is independent of (Γ, Γ'). □

Corollary 8.1. *If a protocol* $\mathcal{P}$ *contains no turn operations outside of valid instances of sort sub-protocols, then* $\mathcal{P}$ *is secure.*

Encoding Permutations

A sequence $(s_1, \ldots, s_n) \in \{1, \ldots, n\}^n$ of card symbols *encodes a permutation* π if $s_i = \pi(i)$ for $1 \leq i \leq n$. Let us denote $\mathcal{D}_5 := [1, 2, 3, 4, 5]$ and $\mathcal{D}_2 := [\clubsuit, \heartsuit]$, and give a short example.

Example 8.1. The 5-cycle permutation $\pi = (1\ 2\ 3\ 4\ 5)$ is represented via $\mathcal{D}_5$ by $\Gamma_\pi = (2, 3, 4, 5, 1)$. The (self-inverse) transposition $\tau = (1\ 2)$ is represented via $\mathcal{D}_2$ as $\Gamma_\tau = (\heartsuit, \clubsuit)$.

Useful Specializations

Two sub-classes of sort protocols will be particularly useful. The first will be useful, e.g., to apply an encoded permutation to another sequence of cards, the second to rotate a sequence by a specified offset.

Apply a permutation encoded in A to the sequence in B. Assume that in a surrounding protocol $\mathbb{A} = \Gamma[A]$ is known to always be a permutation of a fixed set M of n distinct cards, say of $M = \{1, 2, \ldots, n\}$. Then $\mathsf{sort}_{S_n}\, A \uparrow B$ is valid at this point as $\mathsf{supp}(A)$ is a subset of all permutations of M, which is an orbit w.r.t. $\Pi = S_n$. The effect is that the permutation *encoded* in A is applied to $\Gamma[B]$. Whenever $\Pi = S_n$, we omit Π as an index of $\mathsf{sort}_\Pi\, A \uparrow B$.

Apply a rotation encoded in A to the sequence in B. Assume that in a surrounding protocol $\mathbb{A} = \Gamma[A]$ is known to always be a permutation of a multiset M with $n-1$ copies of one symbol and one copy of another symbol, say $M = [(n-1)\cdot\heartsuit, \clubsuit]$. Let $\clubsuit < \heartsuit$ by convention. Then for $\Pi = \langle(1\,2\,\ldots\,n)\rangle$, an occurrence of $\mathsf{sort}_\Pi\, A \uparrow B$ is clearly valid at this point, as

$$\mathsf{supp}(A) \subseteq \{(\clubsuit, \heartsuit, \ldots, \heartsuit), (\heartsuit, \clubsuit, \heartsuit, \ldots, \heartsuit), \ldots, (\heartsuit, \ldots, \heartsuit, \clubsuit)\}$$

and the latter is an orbit w.r.t. Π. The effect is that the rotation *encoded* in A is applied to $\Gamma[B]$. In this case we also write $\mathsf{rot}\, A \uparrow B$ for $\mathsf{sort}_\Pi\, A \uparrow B$.

Note that for $n = 2$, the two cases are the same.

Non-Destructive Variant sort*

We define a variation sort^* of sort that differs only in so far as it should make no net change to the cards in positions A. For this, a sequence of *helping cards* is assumed to be available in (otherwise unused) positions $H = (h_1, \ldots, h_n)$. We implement sort^* in Protocol 8.8 by two applications of sort, where the latter restores $\mathbb{A}$ from the helping "register".

$\mathsf{sort}_\Pi\,(a_1, \ldots, a_n) \uparrow ((b_1, h_1), \ldots, (b_n, h_n))$
$\mathsf{sort}_\Pi\,(h_1, \ldots, h_n) \uparrow (a_1, \ldots, a_n)$

Protocol 8.8. $\mathsf{sort}^*_\Pi(a_1, \ldots, a_n) \uparrow (b_1, \ldots, b_n)$:

We say an application of sort^* is *valid* whenever an application of sort would be valid and $\mathbb{H} := \Gamma[H] = (1, \ldots, n)$ is guaranteed, i.e., H contains cards with numbers in ascending order.

It is easy to see that under these conditions, if π is applied to the cards in positions A and H in the first sorting step, then π^{-1} is applied to the cards in positions A and H in the second sorting step, as this is the unique permutation that sorts the cards in positions H. Thus, one complete valid application of sort^* makes no net changes to A and H. It is also easy to check that both applications of sort are valid in the original sense, therefore Lemma 8.1 and Corollary 8.1 extend naturally to sort^*. We use rot^* for the variant using cyclic rotations.

Stating Classical Protocols in Terms of sort

The standard AND, OR, XOR and COPY protocols due to Mizuki and Sone [MS09] can all be stated as single application of our sort sub-protocol as shown in Protocols 8.9 to 8.12 in Figure 8.9. We also provide a *permutation application protocol* that takes the encoding of a permutation and a sequence as input and outputs the permuted sequence. This is in essence the permutation division protocol by Hashimoto et al. [HSN+17] (the only change being that we encode the inverse permutation).

Input: bits $(x, y, 0)$ encoded in $((1, 2), (3, 4), (5, 6))$
Output: encoding of $x \wedge y$
$\mathsf{sort}\,(1, 2) \uparrow ((3, 4), (5, 6))$
$(\mathsf{result}, 5, 6)$

Protocol 8.9. AND.

Input: bits $(x, y, 1)$ encoded in $((1, 2), (3, 4), (5, 6))$
Output: encoding of $x \vee y$
$\mathsf{sort}\,(1, 2) \uparrow ((3, 4), (5, 6))$
$(\mathsf{result}, 3, 4)$

Protocol 8.10. OR.

Input: bits (x, y) encoded in $((1, 2), (3, 4))$
Output: encoding of $x \oplus y$
$\mathsf{sort}\,(1, 2) \uparrow (3, 4)$
$(\mathsf{result}, 3, 4)$

Protocol 8.11. XOR.

Input: bits $(x, 0, \ldots, 0)$ encoded in $((1, 2), \ldots, (2n + 1, 2n + 2))$
Output: n card pairs, all encoding x
$\mathsf{sort}\,(1, 2) \uparrow ((3, 5, \ldots, 2n + 1), (4, 6, \ldots, 2n + 2))$
$(\mathsf{result}, 3, 4, \ldots, 2n + 1, 2n + 2)$

Protocol 8.12. n-COPY.

Input: permutation π encoded in $(1, \ldots, n)$ and some sequence $\mathbb{A}$ in $(n + 1, \ldots, 2n)$
Output: $\pi(\mathbb{A})$
$\mathsf{sort}\,(1, \ldots, n) \uparrow (n + 1, \ldots, 2n)$
$(\mathsf{result}, n + 1, \ldots, 2n)$

Protocol 8.13. APPLY.

Figure 8.9.: The classical protocols AND, OR, XOR and COPY as well as a permutation application protocol, all stated as sort protocols.

8.7. Conclusion

All new protocols in this chapter, except the first protocol in Section 8.2 appear in Tables 17.1 to 17.3 which give an overview of card-minimal protocols w.r.t. different requirements, decks and functionalities. Hence, they can be seen as optimal in some precise sense. Constructing these protocols can in part also be seen as an exploration of the strength of the computational model of [MS14a] and shuffle-restrictions thereof. For example, we see the protocol computing k-ary Boolean functions as merely a touch-stone of the computational model, as we feel that the protocol shows that non-closed shuffles together with restarts, are unplausibly powerful.

As an interesting further direction, one might re-consider the finite running time case, both for two-color and standard decks. The two-color deck protocols use the fifth card only in a certain branch of the protocol, and one might be inclined to search for

protocols that employ this card throughout the protocol to, e.g., achieve a simpler or faster version. Moreover, in the standard deck case, the best protocol for AND in finite-runtime uses eight cards. We are keen to know whether there is a protocol using less cards in this setting.

We introduced sort protocols as an interesting building block – also largely simplifying security proofs – which we believe to be of independent interest, as many protocols from the literature can be restated in these terms. It would be interesting to find more protocols which can be restated in this way. In Section 12.10, we will see additional protocols of another type, namely those where the input of a player is not given by a commitment via two face-down cards, but by a permutation that is done to the cards by the player, hence requiring input awareness. (Using sort protocols, these could be transformed to protocols without input awareness, by encoding the permutation to be done for the respective input with cards and then using a sort protocol to apply them. As we will see, this is only "committed format", if we encode a transposition.)

9. Minimal Decks for Secure Computation

In this chapter, we give proofs for several lower bounds on the number of cards for protocols computing AND and COPY. This is an important ingredient for identifying the best protocols for secure multiparty computation with cards.

For this, we consider weaker forms of security, as they are easier to work with. If even for these security notions we can show that no protocol exists, than this holds also for the stronger security notion. As our proofs work by induction on state diagrams, we introduce a reduced variant of state diagrams that fit this weaker security, and reduces the state space to a finite (but large) set.

9.1. Possibilistic Security and Reduced States

This section is from [KKW+17], with some adaptations. When trying to prove that no secure protocol with certain properties exists, the space of possible states can seem vast. We therefore opt to show stronger results, namely the non-existence of protocols with the weakened requirement of possibilistic security, defined next. This allows to consider a projection of the state space which is finite in size.

Definition 9.1. A protocol $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ computing a function $f\colon \{0,1\}^k \to \{0,1\}^m$ has *possibilistic input security (possibilistic output security)* if for uniformly[1] random input $I \in U$ and any visible sequence trace v with $\Pr[v] > 0$ as well as any input $i \in \{0,1\}^k$ (any output $o \in \{0,1\}^m$ in the image of f) we have $\Pr[v \mid \mathsf{val}_\mathsf{i}(I) = i] > 0$ ($\Pr[v \mid f(\mathsf{val}_\mathsf{i}(I)) = o] > 0$).

In other words, from observing a visible sequence trace it is never possible to exclude an input (or output) with certainty. Clearly, security implies possibilistic input and output security.

Reduced State Diagrams

To decide whether a protocol has possibilistic output security, it suffices to consider projections of states in the following sense.

We define reduced states, where states are not annotated by their symbolic probabilities, but by the result that is specified by their inputs. This is to simplify impossibility results, as reducing information allows us to make sense of which types of states are reachable without caring about the exact probabilities, and it actually makes the search space of (reachable) states finite. Any such reduced diagram captures only the weak

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

form of possibilistic security as defined 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. This is the general strategy which we pursue for our impossibility proofs.

Definition 9.2. Let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a protocol computing a Boolean function $f\colon \{0,1\}^k \to \{0,1\}^m$ w.r.t. input valuation val_i and output valuation val_o. If μ is a state in the state diagram, then the *reduced state* $\hat{\mu}$ has the same sequences as μ with simpler annotations. Assume $\mu(s)$ is a polynomial with positive coefficients for the variables $X_{b_1}, \ldots, X_{b_i}$ $(i \geq 1)$. Then we set $\hat{\mu}(s) := o \in \{0,1\}^m$ if $o = f(b_1) = f(b_2) = \ldots = f(b_i)$. If not all b_i evaluate to the same output under f, set $\hat{\mu}(s) = \bot$. Accordingly, sequences in $\hat{\mu}$ are called *o-sequences* or *⊥-sequences*. We also say they are *of type o* or *of type ⊥*, respectively. For COPY protocols, we call 1^m- and 0^m-sequences just 1- and 0-sequences, respectively.

Note that if a state μ has children $\mu_1, \ldots, \mu_i$ in the state diagram, then the reduced states $\hat{\mu}_1, \ldots, \hat{\mu}_i$ can be computed from the reduced state $\hat{\mu}$. In particular, it makes sense to define *reduced state diagrams* as projections of state diagrams and aim to prove the non-existence of certain state diagrams by proving the non-existence of the corresponding reduced state diagrams.

9.2. Important Properties of (Reduced) States

For most of our arguments, reduced states offer a sufficient granularity of detail and the following observations and definitions for reduced states will prove useful. Clearly, they also apply to the richer, non-reduced states which we will require for more subtle arguments in Sections 9.7, 9.10 and 9.11.

Protocols computing a function f that have a ⊥-sequence in a reduced state cannot be restart-free, and hence also not finite-runtime: once the ⊥-sequence is actually on the table, it does not contain sufficient information to deduce the unique correct output, making a restart necessary. The protocol might then repeat the unfortunate path of execution an arbitrary number of times.

We call two (reduced) states *similar*, if one is just a permuted version of the other, i.e., interpreting a state as a matrix of symbols with annotated rows, there is a permutation on the columns mapping one state to the other.

In restart-free committed-format protocols with two outputs (say 1 and 0) any reduced state $\hat{\mu}$ is composed of some number i of 0-sequences and some number j of 1-sequences with $|\hat{\mu}| = i + j$. We call μ and $\hat{\mu}$ an *i/j-state*. They are *final* if they admit a correct output action $(\mathsf{result}, p_1, q_1 \ldots, p_m, q_m)$, i.e., they do not contain a ⊥-sequence and for all outputs $r \in \{0,1\}^m$ in the image of f, there is at least one r-sequence and for all r-sequences s it holds that $\mathsf{val}_>(s[p_i], s[q_i]) = r[i]$ for $1 \leq i \leq m$, i.e., the cards at the respective indices encode the value of the respective bit. For two-color decks, this translates to having a card with symbol ♡ at position p_i if $r[i] = 1$, or with symbol ♣ otherwise, and the respective other symbol at position q_i.

Turnable Positions. Recall the definition of a turnable position $T = \{i\}$ for a state μ from (7.6). For a reduced state $\hat{\mu}$ this simplifies to: For each symbol $c \in \Sigma$ occurring in column i, among the sequences s with $\mathsf{symb}(s[i]) = c$, there is an r-sequence for each $r \in \{0,1\}^m$ in the image of the function computed by the protocol, and/or a $\perp$-sequence. This essentially means that all outputs are still possible, hence captures what we mean by possibilistic (output) security. If some position in a (reduced) state is turnable, we call the state turnable, otherwise unturnable.

Some General Properties. We bundle a few simple properties about i/j-states (taken from [KWH15]) in the following lemma:

Lemma 9.1. *Given a secure protocol computing a non-constant Boolean function with deck $\mathcal{D}$, consisting of n ♡s and m ♣s where $n, m \geq 1$, the following holds.*

1. *In a state of type i/j, we have $i, j \geq 1$, otherwise players could derive the the result, contradicting the committed format property.*
2. *If a turn in a state μ of type i/j can result in two different successor states μ_1 and μ_2 of type i_1/j_1 and i_2/j_2, respectively, then $i = i_1 + i_2$ and $j = j_1 + j_2$. In particular, $i \geq 2$ and $j \geq 2$.*
3. *In a state of type i/j resulting from a turn that revealed a ♡ or ♣ we have $i + j \leq \binom{n+m-1}{n-1}$ or $i + j \leq \binom{n+m-1}{m-1}$, respectively.*
4. *Let μ be a state of type i/j and μ' a state of type i'/j' resulting from μ via a shuffle operation. Then we have $i' \geq i$, $j' \geq j$.*
5. *If μ is a final state of type i/j, then $i, j \leq \binom{n+m-2}{n-1}$.*
6. *Two sequences differ in an even number of positions, i.e., have even* distance.
7. *Given an sequence $s \in \mathsf{Seq}^{\mathcal{D}}$, there are*

$$\binom{n}{\frac{d}{2}}\binom{m}{\frac{d}{2}}$$

   sequences of (even) distance d to s.
8. *Any two sequences have distance at most $\min\{2m, 2n\}$.*
9. *After a single-card turn revealing ♡ or ♣, any two sequences of the state have distance at most $2n - 2$ or $2m - 2$, respectively.*

9.3. Finite-Runtime AND Requires Five Cards

This section is from [KWH15], with some adaptations. There are secure protocols with four cards computing AND in committed format using either the restart operation

(see Section 8.3) or running in cycles for a number of iterations that is finite only in expectation (see Section 8.1). However, it would be nice to have a protocol that is finite-runtime, i.e., is guaranteed to terminate after a finite number of steps. In the following we show that this is impossible.

To this end, we distinguish several different types of (reduced) states and later analyze which state transitions are possible. With our observations from the previous section we are ready to directly state our result in the following theorem:

Theorem 9.1. *There is no secure finite-runtime four-card AND protocol in committed format with deck* $\mathcal{D} = [\heartsuit, \heartsuit, \clubsuit, \clubsuit]$.

Proof. Let $\mathcal{P}$ be a secure protocol computing AND with four cards in committed format.

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

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

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

We first observe which state types i/j can occur with our deck: Since there are $6 = \binom{4}{2}$ sequences in total, we need $i + j \leq 6$. By Lemma 9.1, item 1, states with $i = 0$ or $j = 0$ cannot occur.

Final States are Good. From item 5 in Lemma 9.1 we know that final states fulfill $i, j \leq 2$ so the only candidate for final states are 1/1, 2/2, 1/2 and 2/1. We need to show that they are good which is true by definition for 1/1 and 2/2. Consider a final 1/2-state (the argument for the 2/1-state is symmetric). Its 0-sequence differs from both 1-sequences in the two positions used for the output. Since the two 1-sequences are distinct, at least one of them must differ from the 0-sequence in another position, meaning they must have distance at least 3 and therefore distance 4 (item 6 in Lemma 9.1).

Therefore, all final states are good, but the start state, which is a 3/1-state, is non-good. Consider an action $\mathsf{act} \in \mathsf{Action}_4$ that acts on a non-good state. We show that act has a non-good successor state by considering all cases for the type of act:

Non-trivial Single-card Turns. Let μ be a non-good state of type i/j, and $\mu_\heartsuit$ and $\mu_\clubsuit$ the two possible states after a turn of a single card. From item 2 in Lemma 9.1, we know that μ has to be of type i/j, with $i, j \geq 2$, excluding the case of 2/2, as μ is non-good. This leaves the following possible types for μ: 2/3, 3/3, 2/4 where we assume

without loss of generality that $i \leq j$. The turn partitions the sequences onto the two branches in one of the following ways:

2/3		3/3		3/3		2/4		2/4	
/	\	/	\	/	\	/	\	/	\
1/1	1/2	1/1	2/2	1/2	2/1	1/2	1/2	1/3	1/1

From item 3 in Lemma 9.1, we know that a state resulting directly from a turn contains at most 3 sequences, thereby ruling out turn-transitions that lead to a 2/2- or 1/3-state. Moreover, any 2/1- or 1/2-state occurring after a turn has the property that all sequences have pairwise distance 2 by item 9 in Lemma 9.1. By definition, such 2/1-states are non-good. Note that a turn action on a 2/3-state – while producing a good and even final 1/1-state – produces a non-good 1/2-state on the other branch.[2]

Non-branching Shuffles. Now consider a shuffle that produces a unique subsequent state μ' of type i'/j'. We want to show that μ' is non-good. Using item 4 in Lemma 9.1 and the fact that a good μ' would require $i', j' \leq 2$, we only need to consider the case that μ is a non-good state with $i, j \leq 2$, i.e., μ is of type 1/2 or 2/1 with pairwise distance 2 – without loss of generality of type 1/2 and with a 0-sequence s_0 and two 1-sequences s_1 and s_1'. We argue that without loss of generality μ is of the form

s_0:	♡♡♣♣
s_1:	♡♣♡♣
s_1':	♡♣♣♡

This is because

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

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

[2] Moreover, this is the only way to produce a good state from a non-good state via a turn action. We make use of such a turn in our four-card protocol in Section 8.1, which did not require finite-runtime. (In contrast to our protocol in Section 8.3 this allows us to avoid restart actions.)

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

Restart. This action is not allowed in our finite-runtime setting.

Result. Since non-good states are non-final this action cannot be applied.

Permutation. This is just a special case of a non-branching shuffle.

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

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

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

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

This concludes the proof. □

9.4. Protocols for n-COPY Require $2n$ Cards

In this section, we prove that any protocol that produces n copies of a commitment needs at least $2n + 1$ cards, showing that we cannot improve w.r.t. to the number of cards on the protocol in [NNH+18], cf. Figure 7.10. The results in this section are mainly due to co-authors Miyahara, Hayashi, Mizuki and Sone of [KKW+17], with some made changes and simplifications to the proof and exposition. We start with a lemma.

Lemma 9.2. *Let $\mathcal{P}$ be a secure protocol computing a function f. Assume that a reduced state μ of $\mathcal{P}$ is transformed to a* non-similar *state μ' by an action. Then,*

1. *the action cannot be of type* perm.
2. *if the action is a turn, μ' has a sibling state.*
3. *if it is a shuffle, either $|\mu| = |\mu'|$ and μ' has a $\perp$-sequence,* or $|\mu| < |\mu'|$.

Proof. 1. A permutation can only produce similar states by definition.

2. As μ' and μ are not similar, μ' contains a proper subset of the sequences of μ. The sequences which were removed from μ' are not compatible with the visible sequence annotation to μ' and hence it needs to have a sibling state.

3. Clearly, a $(\mathsf{shuffle}, \Pi, \mathcal{F})$ action cannot reduce the number of sequences. Assume that $|\mu| = |\mu'|$ and μ' contains no $\perp$-sequences. Let $s_1, s_2, \ldots, s_i$ be the sequences in μ. For any $\pi \in \Pi$, μ' includes all sequences $\pi(s_1), \pi(s_2), \ldots, \pi(s_i)$. Because $|\mu| = |\mu'|$, μ' cannot have any other sequences. Thus, μ and μ' are similar via π, a contradiction. □

Our impossibility proof first assumes the existence of a $2n$-card COPY protocol which is minimal in the sense that it admits the shortest run, i.e., there is no protocol with the same functionality where there is a leaf state that is less deep.[3] We call it *run-minimal*. We then derive a contradiction by showing that the leaf state of the shortest run cannot be reached by one of the admissible actions.

Theorem 9.2. *There is no (possibilistically) secure $2n$-card n-COPY protocol.*

Proof. Suppose for a contradiction that there are $2n$-card n-COPY protocols, and let $\mathcal{P} = (\mathcal{D}, H, U, Q, A)$ be a run-minimal one. Let μ' be a leaf state of the state diagram of $\mathcal{P}$ of minimum depth. As μ' is final, it is similar to the state

$$\begin{array}{ll} (\clubsuit\heartsuit)^n & 0 \\ (\heartsuit\clubsuit)^n & 1, \end{array}$$

meaning μ' contains exactly two sequences and the distance between them is $2n$. Let μ be the parent state of μ'. Note that μ and μ' are not similar, as otherwise the shortest run would obviously not be minimal, as we could remove μ' from the diagram. Hence, by Lemma 9.2 we only need to consider the following actions:

- (turn, T): Sequences in μ' coincide at positions in the (non-empty) turn set T. But then μ' would not contain two sequences of distance $2n$, a contradiction.

- $(\mathsf{shuffle}, \Pi, \mathcal{F})$: The only way for μ to have only one sequence is if it is a $\perp$-sequence. Note that from that situation a shuffle cannot produce any 0- or 1-sequences and hence cannot end in μ'. Therefore, assume $|\mu| \geq 2$. Then, by item 3 of Lemma 9.2 there are two possibilities. If $|\mu'| = |\mu|$ we would have introduced a $\perp$-sequence, which is not present in μ'. Therefore, we have $|\mu'| > |\mu| \geq 2$, which is a contradiction to $|\mu'| = 2$.

- $(\mathsf{restart})$: μ' would be the start state, where there are exactly two sequences of distance 2, contrary to the distance of $2n$, with $n > 1$. □

By Theorem 9.2 and the existing $(2n+1)$-card COPY protocol [NNH+18], $2n+1$ is the necessary and sufficient number of cards for making n copied commitments. In the next section, we restrict our attention to finite-runtime protocols.

[3] This is to avoid intricacies with definitions such as: "we cannot delete a leaf while retaining functionality" (used in Section 9.5) for *infinite trees*.

9.5. Finite-Runtime n-COPY Requires $2n+2$ Cards

In Section 9.4, we showed that $2n+1$ cards are minimal for COPY computations. Note that the existing $(2n+1)$-card COPY protocol [NNH+18] has a running time that is finite only in expectation. In this section, we show that if we are restricted to a finite running time, there cannot be a protocol using only $2n+1$ cards. For this, we look at the finite state tree of an assumed minimal protocol and consider a leaf state with a position of largest depth in the tree (i.e., there is no leaf state which is even deeper). By contradiction, we show that no such leaf state can exist, as it would have a sibling which is not yet a leaf state. The results in this section are mainly due to co-authors Miyahara, Hayashi, Mizuki and Sone of [KKW+17], with some made changes and simplifications to the proof and exposition.

Theorem 9.3. *There is no (possibilistically) secure* $(2n+1)$*-card finite-runtime* n*-COPY protocol.*

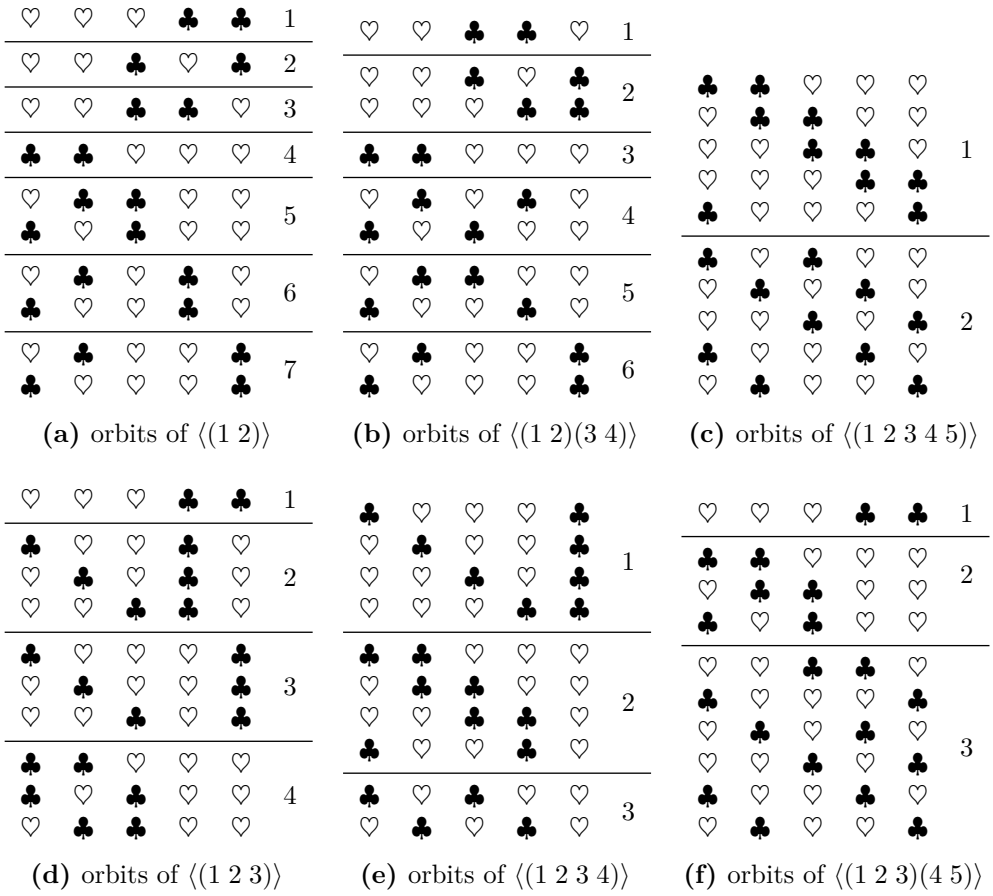
Proof. Suppose for a contradiction that there exist $(2n+1)$-card finite-runtime n-COPY protocols and let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a minimal one, in the sense that we cannot remove a leaf while retaining functionality. The deck $\mathcal{D}$ includes at least n ♣s and n ♡s, and let remaining card be of symbol ♣ or of third color, say ♢. Because of the finite running time, the height of the state tree of $\mathcal{P}$ is finite, and hence, there must be a deepest leaf state, i.e., a leaf state with no other leaf state being on a level below it. We call it μ'. As it is a final state, it needs to look like this, up to reordering:

$$\begin{array}{llcll} ♣(♣♡)^n & 0 & & ♢(♣♡)^n & 0 \\ ♣(♡♣)^n & 1, & \text{or} & ♢(♡♣)^n & 1. \end{array}$$

The distance between the two sequences is $2n$, and one column is constant. Let μ be the parent state of μ'. Similarly to the proof of Theorem 9.2, perm and shuffle cannot have transformed μ to μ'. Hence, from the form of μ' the action must be $(\mathsf{turn}, \{1\})$. By the minimality of $\mathcal{P}$, μ and μ' are not similar. Therefore, item 2 of Lemma 9.2 implies that there is a sibling state μ'' of μ'. This sibling state has a constant ♡-column (or possibly a ♣-column, if we have a ♢) in the first position, as it is the only way for the visible sequences of the turn to differ. In this case, we cannot construct two sequences whose distance is $2n$ with the remaining $n-1$ ♡s and $n+1$ ♣s (or n ♣s and a ♢) in μ''. Therefore, μ'' cannot be a leaf state of $\mathcal{P}$, and there would be a deeper leaf than μ', contrary to our assumption. □

9.6. Prerequisites for Restricted Shuffle Lower Bounds

This section is mainly taken from [KKW+17; K18] and contains important insights for lower bound proofs when considering protocols that use only uniform or closed or uniform closed shuffles.



(a) orbits of $\langle(1\ 2)\rangle$ **(b)** orbits of $\langle(1\ 2)(3\ 4)\rangle$ **(c)** orbits of $\langle(1\ 2\ 3\ 4\ 5)\rangle$

(d) orbits of $\langle(1\ 2\ 3)\rangle$ **(e)** orbits of $\langle(1\ 2\ 3\ 4)\rangle$ **(f)** orbits of $\langle(1\ 2\ 3)(4\ 5)\rangle$

Figure 9.1.: Orbits of different closed (cyclic) shuffle operations on the card sequences.

Properties of (Uniform) Closed Shuffles

For our proof we will need the orbit partitions of closed shuffles on sequences of five cards with three ♡ and two ♣. A display of all relevant orbit partitions can be seen in Figure 9.1. Before we proceed with the main theorem in Section 9.7, it is beneficial to state some observations about orbit partitions and closed shuffles.

Lemma 9.3. *Assume we shuffle a state μ into a state μ' using a closed permutation set Π. If there is $\pi \in \Pi$ with $\pi(i) = j$, then in μ' the columns i and j contain the same multiset of symbols.*

Proof. Assume column i has k ♣s in μ'. Because Π is closed, shuffling again with Π yields μ' again. In particular if s' is contained in μ', then so is $\pi(s')$. Since π is a bijection on the set of all sequences, the sequences with a ♣ in position i are mapped to distinct sequences with a ♣ in position j. In particular column j contains at least as many ♣s as column i in μ'. Since the same argument works for all other symbols and the total number of sequences did not increase, the numbers of ♣ (and other symbols) in column i and j coincides. Since ♣ was arbitrary, the claim follows. □

Observe that Lemma 9.3 is false, if Π is not closed. Take for instance an action (perm, π) for $\pi \neq \mathsf{id}$. It is a non-closed shuffle clearly lacking this property.

Lemma 9.4 (Shuffling and Orbit Partitions)**.** *Let μ be transformed into μ' via some shuffle with closed permutation set Π. Let O be an orbit of Π.*

1. *If $\mu \cap O = \emptyset$, then*[4] *$\mu' \cap O = \emptyset$*
2. *If $\mu \cap O$ contains a sequence of type r and no other sequences of type $r' \neq r$, then $\mu' \cap O = O$ contains only r-sequences.*
3. *Otherwise, $\mu' \cap O = O$ contains only $\perp$-sequences.*

Proof. Note that for any pair s_1, s_2 of sequences, there is some $\pi \in \Pi$ with $\pi(s_1) = s_2$ precisely if they are in the same orbit of Π. Thus when shuffling with Π, the type $\mu(s_1)$ directly "infects" precisely the entire orbit of s_1. With $\perp$ indicating several types jumbled together, the three cases are easily checked. □

For the more restricted case of uniform closed shuffles, let us state the following simple but interesting observation: all sequences in an orbit have the same symbolic probability after the shuffle.

Lemma 9.5. *Let $\mathsf{act} = (\mathsf{shuffle}, \Pi)$ be a uniform closed shuffle and μ a state. Let μ' be the state arising from μ through act and s'_1 and s'_2 two sequences of symbols in μ'. If $s'_1 = \pi(s'_2)$ for $\pi \in \Pi$, then $\mu'(s'_1) = \mu'(s'_2)$, i.e., the sequences have the same probability.*

For an analysis of a "backwards shuffle" (from [K18], to be introduced in Section 9.9), we make use of the state partitions as defined next. A *partition* $P(\mu)$ of a state μ arises by putting all sequences of the state into the same set, which evaluate to the same symbolic probability. (More formally, it arises from the equivalence relation of having the same $\mu(\cdot)$ value). Moreover, a *type partition* $T(\mu)$ of μ is defined similarly, putting all sequences of the same type (i.e., the associated output value or a $\perp$-symbol) into a set.

To state our criterion in a formal way, we define the (natural) ordering on partitions as follows: Given two partitions P, P' on some set M, we say that P *is finer* than P', or equivalently, that P' *is coarser* than P, if every set in P is subset of a set in P'. It holds that for any state μ, its partition $P(\mu)$ is finer than its type partition $T(\mu)$. This can, e.g., be seen in Example 9.1, as $\{\heartsuit\heartsuit\clubsuit\clubsuit, \heartsuit\clubsuit\heartsuit\clubsuit\}$ is contained in both sets, and both $\{\clubsuit\heartsuit\heartsuit\clubsuit, \heartsuit\clubsuit\clubsuit\heartsuit\}$ and $\{\clubsuit\heartsuit\clubsuit\heartsuit, \clubsuit\clubsuit\heartsuit\heartsuit\}$ are subsets of the larger four-element set of $T(\mu)$. Using these, let us give a rephrasing of relevant results from just before, using our new vocabulary.

Lemma 9.6. *Let μ be transformed into μ' via some shuffle with permutation group Π. Then the* orbit partition *of Π is (non-strictly) finer than the* type partition *$T(\mu')$ of μ'. Moreover, if the shuffle is uniform, the orbit partition of Π is (non-strictly) finer than the* partition *$P(\mu')$ of μ'.*

[4] We slightly abuse notation here, using μ for the set of sequences contained in μ.

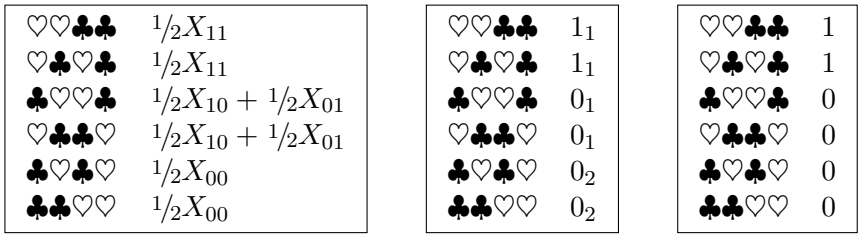


Figure 9.2.: A state from Figure 8.1 given in three forms. *Left:* the full (non-reduced) form, *middle:* its semi-reduced form, where the four 0-sequences are divided into two parts because they are assigned distinct symbolic probabilities, and *right:* its reduced form, which no longer carries the information about these distinct probabilities. The state is turnable at the second position.

Proof. Let μ be transformed into μ' via some shuffle with permutation group Π. Assume to the contrary, that there are two sequences in the same orbit of Π, but with distinct type, where type includes $\emptyset$ (if the sequence is not contained in the state) and $\perp$. This is a contradiction to Lemma 9.4. In the same way, if the shuffle is uniform, there cannot be two sequences in the same orbit of Π with distinct probabilities due to Lemma 9.5. □

The case that it is *strictly* finer occurs, if the sequences of two distinct orbits of the shuffle subgroup happen to have the same symbolic probability or type, respectively, after the shuffle.

Semi-reduced States

In this section from [K18], we introduce a new representation, which is an intermediate between states and reduced states, but captures the *distinctness* of sequence probabilities that is not contained in (fully) reduced states. We make use of this additional information in impossibility proofs for protocols which are restricted to uniform closed shuffles. As an intermediary step, we use partitions of a state as just defined. We refer the reader to Figure 9.2 for an example and illustration of what our definitions in this section aim at.

Example 9.1. The state of Figure 9.2 (left), let us call it μ, has a partition

$$P(\mu) = \{\{♡♡♣♣, ♡♣♡♣\}, \{♣♡♡♣, ♡♣♣♡\}, \{♣♡♣♡, ♣♣♡♡\}\}$$

(which is also encoded in its semi-reduced representation as exactly the sequences with the same annotation end up in a common set of the partition), and a type partition

$$T(\mu) = \{\{♡♡♣♣, ♡♣♡♣\}, \{♣♡♡♣, ♡♣♣♡, ♣♡♣♡, ♣♣♡♡\}\}.$$

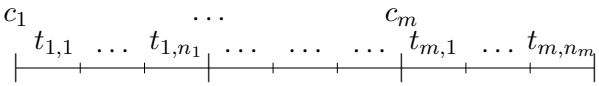
We aim to represent the partition $P(\mu)$ of a state in its semi-reduced form. For this, we introduce as many copies of types as there are partitions with sequences of this type, and add a subscript from $\mathbb{N}$ to distinguish these, cf. Figure 9.2 (middle). (If there is only one partition of a type, we nevertheless give it the subscript 1). The order of these subscripts will be irrelevant for our purposes.

This *semi-reduced form of a state* can be generated by appropriately projecting the sequence probabilities to these *indexed type* symbols.

Turn-Split Representation of a (Semi-)Reduced State

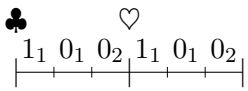
For turnable states, we would like to introduce an additional, compressed method to depict the state, where we are only interested in the partitioning of the state and how it behaves relative to the split of the state due to the turn. For this, let us regard empty sequences (sequences not in the state, i.e., with probability 0) as a *sequence of type* $\emptyset$. We sort the state according to a column i (usually a turnable column) and assume that the deck is chosen over an alphabet $c_1, \ldots, c_m$, which carries some ordering $c_1 \leq \cdots \leq c_m$. The ordering inside the blobs for each c_k is irrelevant, and we may only assume an ordering by type.

For this, we will use the following *turn-split representation* w.r.t. a position k of a state:



where $t_{1,1}, \ldots, t_{1,n_1}, \ldots, t_{m,1}, \ldots, t_{m,n_m}$ are (indexed) types of corresponding sequences. The indicated ordering means: if the state is turnable at position k, $t_{i,j}$ belongs to the c_i-branch of a turn at position k, for all $1 \leq j \leq n_i$.

As an example, let us depict the state from Figure 9.2 (left) via this split-state representation w.r.t. the turnable column 2:



We will make more heavy use of this representation in the next subsection and in Section 9.10.

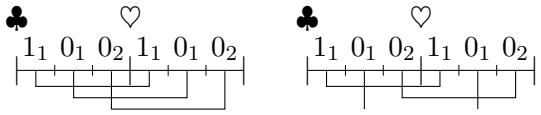
Enriching Turn-Split Representations with Orbit Partitions

As we saw in Lemma 9.6, a necessary criterion to show that we can reach a state μ' via a uniform closed shuffle, is to show that it is compatible with an orbit partition, i.e., that there is a Π with an orbit partition that is finer than the partition of μ'. Hence, we would like to depict a possible orbit partition in a simple way that makes it is obvious that it is finer than the partition of the state.

We do this via connecting horizontal brackets between the sequences representing the orbit decomposition of an admissible shuffle. Here, all connected sequences are meant to be in the same orbit. A single line drawn downward will mean that the sequence is in an orbit of size 1.

We use our previous example of Figure 9.2 in a turn-split representation w.r.t. the turnable position 2. We depict two possible orbit decompositions (on the left with three sets of size two, on the right with one orbit split into two, which are represented by single

lines pointing downwards). It shows that the state is generateable by a shuffle with such an orbit decomposition, as it is not coarser than the partition of the state (specified by the indices).[5] (It would then be left to show that such an orbit decomposition is in fact possible. For example, we will see later in Lemma 9.9, that the left decomposition is impossible.)



The example abstracts from many details and from the form of Π, but if one can already show (and we do so in Section 9.10) that there is *no* orbit partition as fine as the state partition, then direct reachability of such a state via a closed uniform shuffle is impossible.

9.7. Restart-Free Uniform Closed AND Requires Five Cards

As discussed on Chapter 5, if we are willing to discard the finite running time requirement, there is an intermediate property, namely restart-freeness. This section, taken from [KKW+17], focuses on restart-free AND protocols. Our proof is similar to the setting of Theorem 9.1 in that we start from a (slightly enlarged) set of good states, but instead of showing that there is an infinite path of non-good states, we show the stronger property that no path contains a good state.

Note that for protocols with only closed, but possibly non-uniform shuffles, the four-card AND protocol of Section 8.1 already provides a card-minimal solution. For protocols which are additionally restricted to uniform and closed shuffles, we show in the following that five cards are necessary to compute AND in committed format without restarts.

The following argument speaks about non-reduced states. We still use the terminology for reduced states, but the precise notion of turnability from Eq. (7.6).

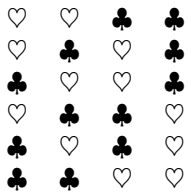
Theorem 9.4. *There is no secure restart-free Las Vegas four-card AND protocol with deck* $\mathcal{D} = [\heartsuit, \heartsuit, \clubsuit, \clubsuit]$ *in committed format if shuffling is restricted to uniform closed shuffles.*

Proof. The proof is similar to the proof of Theorem 9.1. Let $\mathcal{P}$ be a secure protocol computing AND with four cards using only uniform closed shuffles and no restart actions. We define a set $\mathcal{G}$ of good states that contains all final states but not the start state. We then show that we cannot get into the set of good states from a non-good state with a turn or a uniform closed shuffle that does not create $\bot$-sequences. A state is *good* if it is

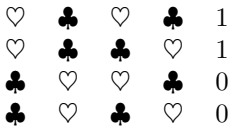
- a state of type 1/1 or 2/2,
- a state of type 1/2 or 2/1 without a constant column,

[5] Note that there are also orbit compositions of sizes $1, 1, 4$, sizes $3, 3$, sizes $2, 4$ and size 6, which are however incompatible as they are coarser than the partition of the state and hence are not displayed

Figure 9.3.:



(a) All sequences with four cards.



(b) A sequence-maximal final state with four cards w.r.t. $(\mathsf{result}, 1, 2)$.

- a turnable state of type 2/3 or 3/2.

All other states are *bad*. The main difference to Theorem 9.1 is that we need to consider states where turning yields at least one good state instead of only the ones that yield only good states. In particular, the additional good states are the turnable 2/3- and 3/2-states.

Note that as our deck is $[\heartsuit, \heartsuit, \clubsuit, \clubsuit]$, we can form at most 6 sequences shown in Figure 9.4a. The start state is bad because it is of type 3/1. The final states are among the good states because they are of type i/j with $i, j \leq 2$. If they are of type 2/1 or 1/2 they do not have a constant column as one can easily see when looking at the sequence-maximal final state for four cards (Figure 9.4b).

Turns. Only states of type i/j with $i, j \geq 2$ are turnable. The only bad states that fit this criterion are of type 3/3 or 4/2 (2/4). The maximum state with a constant column has three sequences, therefore these states can only be turned into two states both of which are of types 1/2 or 2/1 with a constant column.

Shuffles. We cannot shuffle a bad state μ into a good state μ' of type i/j where both $i, j \leq 2$. Assume we could. This would require μ to be a bad 1/2-state, thus μ has a constant column and three sequences. Any shuffle that adds sequences involves the constant column. Any other column of μ contains the symbol of the constant column only once, hence, by Lemma 9.3 it must contain it at least three times after the column has been shuffled with the constant column. This adds at least two sequences, contradicting our assumption on the type of μ'. Therefore we only need to look at shuffles that yield a turnable 2/3 or 3/2-state.

Assume that an action $(\mathsf{shuffle}, \Pi)$ transforms a bad μ into a turnable 3/2 state μ'. Without loss of generality, we assume position 1 is turnable in μ' and the sequence not contained in μ' is $s = \heartsuit\heartsuit\clubsuit\clubsuit$.[6] By Lemma 9.4, $\{s\}$ is an orbit of Π of size 1, i.e., s is invariant under Π. In particular, Π is a non-trivial subgroup of $\{\mathsf{id}, (1\ 2), (3\ 4), (1\ 2)(3\ 4)\} \subseteq S_4$.

If we had $(3\,4) \in \Pi$, then the only two sequences of μ' with $\heartsuit$ in position 1, namely $\heartsuit\clubsuit\heartsuit\clubsuit$ and $\heartsuit\clubsuit\clubsuit\heartsuit$, would share an orbit of Π, and therefore have the same type in μ',

[6] If this is not the case we can apply a permutation such that the constant sequence either is $\heartsuit\heartsuit\clubsuit\clubsuit$ or $\clubsuit\clubsuit\heartsuit\heartsuit$ and use the symmetry of exchanging (relabeling) $\heartsuit$ and $\clubsuit$.

contradicting turnability of the first column of μ'. Thus, either (1 2) or (1 2)(3 4) is in Π (but not both). Assume (1 2) $\in \Pi$, the other case is analogous. By Lemma 9.5, it holds that

$$\mu'(\heartsuit\clubsuit\heartsuit\clubsuit) = \mu'(\clubsuit\heartsuit\heartsuit\clubsuit) \text{ and } \mu'(\heartsuit\clubsuit\clubsuit\heartsuit) = \mu'(\clubsuit\heartsuit\clubsuit\heartsuit), \qquad (*)$$

because these each share an orbit. As position 1 of μ' is turnable, we know that

$$\mu'(\heartsuit\clubsuit\heartsuit\clubsuit) + \mu'(\heartsuit\clubsuit\clubsuit\heartsuit) = p \in (0,1), \text{ and}$$
$$\mu'(\clubsuit\clubsuit\heartsuit\heartsuit) + \mu'(\clubsuit\heartsuit\heartsuit\clubsuit) + \mu'(\clubsuit\heartsuit\clubsuit\heartsuit) = q \in (0,1),$$

i.e., constant. Using this and $(*)$, we obtain $\mu'(\clubsuit\clubsuit\heartsuit\heartsuit) = q - p$, which is constant. It is non-zero, so each of the monomials $X_{00}, X_{01}, X_{10}, X_{11}$ occurs with a positive coefficient, meaning $\clubsuit\clubsuit\heartsuit\heartsuit$ is a $\perp$-sequence of μ' – a contradiction. □

9.8. Finite-Runtime Closed AND Requires Six Cards

In this section, we prove that AND protocols which are restricted to closed shuffle operations, require six cards. The section is from [KKW+17]. The proof uses a similar technique as in Section 9.3. While in the latter there was a set of good states including the final states which you never enter with all branches of a branching point in the protocol, here we found that it was easier to start the other way round, namely to define a set of bad states, about which we prove that starting from these there is always a path in the tree which will enter a bad state again and this path does not contain any final states. Here the situation is more complex as there are many more possible states and we needed to derive new tools (orbit partitions) to make use of the structure of the restricted permutation sets. For this we will make heavy use of Lemmas 9.3 and 9.4, because they enable us to exploit the rich structure of closed shuffles. Let us begin by stating our theorem.

Theorem 9.5. *Let $\mathcal{P}$ be a (possibilistically) secure protocol computing AND in committed format using only closed shuffles with five cards of two symbols.*[7] *Then $\mathcal{P}$ is not finite-runtime.*

Proof Outline. We define a set of *bad states*, such that the start state is one of them. We then show that in any protocol from each bad state there is a path into another bad state. In particular, none of the bad states and none of the states on the paths between them are final. This implies that there is an infinite path starting from the start state precluding a finite running time.

Without loss of generality, the protocols we consider have the following properties, since each protocol that does not have some of these properties can be transformed into an equivalent protocol that does.

[7] Whether a five-card protocol is possible using a deck of three colors, i.e., $\mathcal{D} = [\heartsuit, \heartsuit, \clubsuit, \clubsuit, \diamondsuit]$, is an interesting open question.

- $\mathcal{P}$ does not use operations that transform a (reduced) state into any similar state. These operations are clearly not necessary, when arguing about possibilistic security, which is sufficient in our case.
- $\mathcal{P}$ does not use shuffle operations while cards are lying face up. These are unnecessary by Corollary 6.1 and Lemma 6.2.
- each shuffle set Π is a cyclic subgroup of S_5. This is because each subgroup Π of S_5 can be written as the product $\Pi = \prod_{\pi\in\Pi}\langle\pi\rangle$, implying that doing the cyclic shuffles $\langle\pi\rangle$ one after the other gives the same set of permutations that can happen in total as in Π itself.
- The deck is $\mathcal{D} = [\heartsuit, \heartsuit, \heartsuit, \clubsuit, \clubsuit]$. We need two $\clubsuit$ and two $\heartsuit$ for the inputs. Our arguments work regardless of whether the fifth card is $\clubsuit$ or $\heartsuit$.

Definition of Bad States. Any state μ with one of the following properties is bad:

$\mathcal{B}_{4*}$: μ has four sequences of the same type. In particular, this includes all states with seven or more sequences,

$\mathcal{B}_{3\clubsuit}$: μ has a constant $\clubsuit$-column and three or more sequences,

$\mathcal{B}_{5\heartsuit}$: μ has a constant $\heartsuit$-column and five or more sequences,

$\mathcal{B}_{3\heartsuit\heartsuit}$: μ has two constant $\heartsuit$-columns and three sequences,

$\mathcal{B}_{\heartsuit 3/1}$: μ has a constant $\heartsuit$-column and is of type 3/1 or 1/3.

To see that states of these types are all non-final, first note that any final state is, up to reordering, a (not necessarily proper) subset of the sequences of the following state including at least one 0-sequence and at least one 1-sequence. Hence, we will call it the *maximal final state*:

$\heartsuit\clubsuit\clubsuit\heartsuit\heartsuit$	1
$\heartsuit\clubsuit\heartsuit\clubsuit\heartsuit$	1
$\heartsuit\clubsuit\heartsuit\heartsuit\clubsuit$	1
$\clubsuit\heartsuit\clubsuit\heartsuit\heartsuit$	0
$\clubsuit\heartsuit\heartsuit\clubsuit\heartsuit$	0
$\clubsuit\heartsuit\heartsuit\heartsuit\clubsuit$	0

The bad states do not "fit" into this state, since

$\mathcal{B}_{4*}$: the maximal final state has only three sequences of each type.

$\mathcal{B}_{3\clubsuit}$: the only two columns in the maximal final state with three $\clubsuit$ are the first two. But a restriction to the sequences with $\clubsuit$ in the first (or second) position contains 0-sequences (1-sequences) only.

$\mathcal{B}_{5\heartsuit}$: all columns of the maximal final state contain at most four $\heartsuit$.

$\mathcal{B}_{3\heartsuit\heartsuit}$: no two columns in the maximal final state have three $\heartsuit$ in the same positions.

$\mathcal{B}_{\heartsuit 3/1}$: any admissible restriction of the maximal final state with four sequences and a constant $\heartsuit$-column is of type 2/2.

The start state is bad as required as it falls into category $\mathcal{B}_{\heartsuit 3/1}$. We begin by making a few observations that greatly simplify the proof.

Claim 9.1 (Restriction to Simple Shuffles)**.** *We need to only consider the three shuffle sets* $\langle(1\,2)\rangle, \langle(1\,2\,3)\rangle$ *and* $\langle(1\,2)(3\,4)\rangle$ *in the main proof.*

Proof. As discussed before, we only consider cyclic shuffles, i.e., shuffles of the form $\Pi = \langle\pi\rangle$ for $\pi \in S_5$. Unless $\pi = \mathsf{id}$, the cycle decomposition of π can have either one cycle of length 2, 3, 4 or 5 or two cycles of length 2 or one cycle of length 2 and one cycle of length 3. We treat states that are equal up to similarity (reordering) as equivalent; it therefore suffices to consider one shuffle of each type. Among them, $\langle(1\ 2)(3\ 4\ 5)\rangle$ can be decomposed[8] as $\langle(1\ 2)\rangle \circ \langle(3\ 4\ 5)\rangle$ and we handle those factors anyway. In the cases $\langle(1\ 2\ 3\ 4\ 5)\rangle$ and $\langle(1\ 2\ 3\ 4)\rangle$ there is no choice of two orbits such that both contain less than four sequences, so any shuffle that does not produce $\bot$-sequences will produce at least four sequences of the same type, yielding a bad state of type $\mathcal{B}_{4*}$. □

Claim 9.2 (Criteria for Dead Columns)**.** *The following criteria for columns ensure that if the next turn in the protocol occurs in this column, then we are either already bad or this turn entails a bad successor state. We say the column is* dead.[9] *In particular, if all columns are dead, we know that after the next turn, we get a bad state.*

$\mathcal{D}_{3\clubsuit}$*: The column contains* 3 $\clubsuit$*.*

$\mathcal{D}_{5\heartsuit}$*: The column contains* 5 $\heartsuit$*.*

$\mathcal{D}_{2*\clubsuit}$*: The column contains* 2 $\clubsuit$ *belonging to sequences of the same type.*

$\mathcal{D}_{3*\heartsuit}$*: The column contains* 3 $\heartsuit$ *belonging to sequences of the same type.*

Proof.
- If a column contains three or more $\clubsuit$, turning this column yields a bad state with a constant $\clubsuit$-column and three or more sequences.
- If a column contains five or more $\heartsuit$, turning this column yields a bad state with a constant $\heartsuit$-column and five or more sequences.
- if a column contains two $\clubsuit$ belonging to sequences of the same type, an additional sequence of the other type with $\clubsuit$ must be added in this position to make it turnable. This leads to a column with three $\clubsuit$, and turning there yields a $\mathcal{B}_{3\clubsuit}$.

[8] The cycles have co-prime length, as opposed to the case $\langle(1\ 2)(3\ 4)\rangle$, which we handle explicitly in the proof of the theorem.

[9] If the column is unturnable then any method (not involving a turn) to make it turnable first before turning it in this column, will retain/ensure this deadness property.

- If three sequences of the same type all have ♡ in a column, there needs to be an additional sequence of this type with ♣ in this column to make it turnable. Adding such a sequence yields a bad state of type $\mathcal{B}_{4*}$. □

Claim 9.3 (Death is Contagious)**.** *Consider a dead column with index i in a state μ and a closed shuffle* act *with permutation set Π such that $\pi(i) = j$ for a $\pi \in \Pi$. Then the column with index j is dead as well after applying* act *to μ.*

Proof.
- By Lemma 9.3, the number of ♣ must be the same in columns i and j after the shuffle. Therefore $\mathcal{D}_{3\clubsuit}$ spreads.
- This case is completely analogous to the previous, with $\mathcal{D}_{5\heartsuit}$ instead of $\mathcal{D}_{3\clubsuit}$.
- By Lemma 9.3 column j must have at least two ♣ in the same type of sequences after the shuffle.
- For any shuffle that does not create $\perp$-sequences, by Lemma 9.3 there must be three sequences of the same type that have ♡ in column j. Therefore j is dead after the shuffle. □

Proof of Theorem 9.5. We show that from each bad state there is a path into another bad state by case analysis.

States with four sequences of the same type. Any non-trivial shuffle not producing $\perp$-sequences retains the four sequences of the same type.

Assume w.l.o.g. that there are four 0-sequences and consider turn operations. This requires at least two 1-sequences. If there are two sequences of type 1, we have six sequences in total. Turning either yields two states of at least three sequences, in particular one with constant ♣, a $\mathcal{B}_{3\clubsuit}$, or a 3/1 state with constant ♡, a $\mathcal{B}_{\heartsuit 3/1}$. If there are more that two 1-sequences, there are at least seven sequences in total. A turn yields two successor states – one with a constant ♡-column and k sequences, and one with a constant ♣-column with m sequences and $k + m \geq 7$. So we have $k \geq 5$ ($\mathcal{B}_{5\heartsuit}$) or $m \geq 3$ ($\mathcal{B}_{3\clubsuit}$).

States with a constant ♣-column and four sequences. Up to reordering, the state looks like

♣	♣	♡	♡	♡
♣	♡	♣	♡	♡
♣	♡	♡	♣	♡
♣	♡	♡	♡	♣

This state admits no non-trivial turn operation. Any shuffle operation that does not involve the first column produces a similar state. Since any column other than the first contains three ♡, any other shuffle produces three additional ♡ in the first column by Lemma 9.3, resulting in a state of type $\mathcal{B}_{4*}$.

States with a constant ♣-column and three sequences. Without loss of generality, these states are of type 2/1. They are non-turnable. Shuffles that do not involve the constant column will retain the property of being constant ♣ in that column and three or more sequences. Hence, in the following we consider shuffles involving the constant column.

To keep the proof simple, an important tool are the orbit partitions of each of the three equivalence classes of shuffles, as in Figure 9.1. We try to place the sequences into the orbits, such that completing these does not yield a bad state. W.l.o.g. we choose sequences, such that the constant ♣-column is the first column.

Case 1: $(1\,2)$. The orbit partition looks as in Figure 9.1a. The first three orbits contain no ♣ there and are out. No orbits contain two ♣ in the first column, so both 0-sequences must be placed in distinct orbits. If both orbits are of size 2, shuffling yields four 0-sequences giving a $\mathcal{B}_{4*}$-state. Otherwise, one of the 0-sequences is ♣♣♡♡♡. The other 0-sequence and the 1-sequence must be placed into orbits of size 2. All of them have ♣ only in one column out of the columns $3, 4, 5$, and ♣♣♡♡♡ has ♣ in none of them, so for all choices we end up in a $\mathcal{B}_{5\heartsuit}$-state.

Case 2: $(1\,2)(3\,4)$. Similarly to before, we need to choose one 0-sequence as ♣♣♡♡♡ and need to place the remaining two sequences into the last three orbits of Figure 9.1b. Choosing orbit 4 and 5 yields a $\mathcal{B}_{5\heartsuit}$-state.

If we choose orbits 4 (or 5) and 6 then the first two columns are dead ($\mathcal{D}_{3\clubsuit}$) and the fifth column is dead as well ($\mathcal{D}_{2*\clubsuit}$). If we choose the second 0-sequence within orbit 6, then columns 3 and 4 are dead ($\mathcal{D}_{3*\heartsuit}$).

If we choose the 0-sequence within orbit 4 (or 5) and the 1-sequence within orbit 6, there are two living non-turnable columns. Any shuffle that contains only columns 3 and 4 in one cycle does not produce a 1-sequence with ♣ in those columns, so they remain non-turnable. Any shuffle that makes column 3 or 4 turnable by shuffling them with at least one of the dead columns kills the column in the process.

Case 3: $(1\,2\,3)$. If we place the three sequences in the pairwise distinct orbits of Figure 9.1d, we end up with nine sequences after the shuffle. Otherwise, the two 0-sequences share the bottommost orbit and the 1-sequence must be in the second or third orbit, and we get a $\mathcal{B}_{5\heartsuit}$-state.

States with a constant ♡-column and five or more sequences. W.l.o.g. the first column is constant ♡. Consider a turn operation on column $i \neq 1$. Column i has either three ♡ and turning therefore leads to a $\mathcal{B}_{3\heartsuit\heartsuit}$-state, or column i has three ♣ and turning leads to a $\mathcal{B}_{3\clubsuit}$-state.

Any shuffle not involving the first column keeps it constant ♡, and therefore bad. Consider any shuffle involving the first column, say π is a possible permutation in the permutation set of the shuffle with $\pi(i) = 1$. Column i contains at least

two ♣ and by Lemma 9.3 shuffling yields two new sequences with ♣ in position 1, giving seven or more sequences – a $\mathcal{B}_{4*}$-state.

States with two constant ♡-columns and three sequences. No turn operation is possible as the state is of type 1/2 or 2/1. Shuffles involving none of the constant columns keep them constant. Shuffles involving only one of them, produce a state with two ♣ in that column, so five sequences in total, where the other ♡-column stays constant. This is a $\mathcal{B}_{5\heartsuit}$-state.

For the interesting case of shuffles involving both constant ♡-columns, we again try to place the sequences into the orbits of the different types of shuffles such that completing these orbits does not yield two or more additional 0-sequences, as this would lead to a 4/2 state or to seven or more sequences.

Case 1: $(1\,2)$. As both constant columns have to be involved in the shuffle, they have to be in positions 1 and 2. This leaves the state constant.

Case 2: $(1\,2)(3\,4)$. If the constant ♡-columns are both in the same cycle, we do not get additional sequences. Otherwise, say they are in positions 2 and 3, the only three sequences are ♡♡♡♣♣ from orbit 2, ♣♡♡♣♡ from orbit 5, and ♣♡♡♡♣ from orbit 6, and shuffling results in a $\mathcal{B}_{4*}$-state.

Case 3: $(1\,2\,3)$. W.l.o.g. the constant columns are 1 and 2. The sequences with ♡ in those positions are all in distinct orbits with a combined size of 7, which is a $\mathcal{B}_{4*}$-state.

3/1-*states with a constant ♡-column.* This state is not turnable as it has only one 1-sequence. A non-trivial shuffle not involving the constant column yields a state with five or more sequences and a constant ♡-column. For shuffles involving the constant column we try to place the sequences into the orbits of the different classes of shuffles such that completing these orbits does not yield any additional 0-sequences.

Case 1: $(1\,2)$. We have to involve the constant ♡-column, which is w.l.o.g. in position 1. The only orbits with constant ♡ are 1, 2 and 3. To not produce additional 0-sequences, those need to be the ones occupied by the 0-sequences. We need to place the 1-sequence in orbits 5, 6 or 7, and choose w.l.o.g. 5. Then, the first two columns are dead via $\mathcal{D}_{3*\heartsuit}$. The third column is dead via $\mathcal{D}_{3\clubsuit}$. Columns 4 and 5 are dead via $\mathcal{D}_{2*\clubsuit}$.

Case 2: $(1\,2)(3\,4)$. Regardless, the first two columns are dead via $\mathcal{D}_{3*\heartsuit}$ and the other columns are dead via $\mathcal{D}_{2*\clubsuit}$

Case 3: $(1\,2\,3)$. The constant ♡-column is w.l.o.g. the first. No orbit contains three ♡ in column 1, so the 0-sequences are spread over at least two orbits. Any choice of two orbits contains four or more sequences, so we have four sequences of the same type after the shuffle.

This concludes the proof. □

9.9. A Backwards Calculus for Card-based Protocols

In this section, taken from [K18], we want to develop our proof technique further, by defining a systematic way to determine all states that can reach the final states (or a specified state set) of a protocol by an allowed set of actions. This happens by an iterative process, which, upon termination, lets you check for impossibility, by showing that the start state is not contained in the set. In this case we know that the final state is not reachable from the start state, showing impossibility of a protocol for the allowed actions.

We define the following set of operations

- $\mathsf{shuf}_{*}^{-1}(\mathcal{G})$ for a set of states $\mathcal{G}$ and $* \in \{\emptyset, \mathrm{u}, \mathrm{c}, \mathrm{uc}\}$ which we omit if $* = \emptyset$. This is the *set of states that are transformed into a state in $\mathcal{G}$ by a shuffle* that is *i)* general if $* = \emptyset$, *ii)* closed if $* = \mathrm{c}$, *iii)* uniform if $* = \mathrm{u}$ and *iv)* uniform closed if $* = \mathrm{uc}$. The trivial shuffle is allowed, i.e., $\mathcal{G}$ is a subset of this set.
- $\mathsf{turn}_{\star}^{-1}(\mathcal{G})$ for a set of states $\mathcal{G}$ and $\star \in \{\emptyset, \mathrm{r}, \mathrm{f}\}$ which we omit if $\star = \emptyset$. This is the set of states from $\mathcal{G}$ or states that have a turnable position i such that
  - *if $\star = \emptyset$:* a state from $\mathcal{G}$ is on one of the branches of a turn at i (as immediate child).
  - *if $\star = r$:* a state from $\mathcal{G}$ is on one of the branches of a turn at i (as immediate child) and all other branches are $\perp$-free, i.e., the other children states do not contain a sequence of type $\perp$.
  - *if $\star = f$:* all immediate successor states from a turn at i are in $\mathcal{G}$.

Using this, we can follow a high-level strategy for proving lower bounds on the number of cards for general functionalities. For this, let $\mathcal{G}$ be the set of final states of the respective protocol and note that they do not include any states that contain a $\perp$-sequence. Let $* \in \{\emptyset, \mathrm{u}, \mathrm{c}, \mathrm{uc}\}$ and $\star \in \{\emptyset, \mathrm{r}, \mathrm{f}\}$. Define by $\mathsf{cl}_{\star,*}(\mathcal{G})$ *the closure* of $\mathsf{turn}_{\star}^{-1}(\cdot)$ and $\mathsf{shuf}_{*}^{-1}(\cdot)$ operations on $\mathcal{G}$. If the start state is not contained in $\mathsf{cl}_{\star,*}(\mathcal{G})$, then no protocol exists for the running time/shuffle restrictions $\star$ and $*$, as specified above.

Note that we can define this backwards calculus for both detail levels of state trees, namely for (semi-)reduced states and for normal states. However, in this work we *only* need it for *(semi-)reduced states*, and assume this to be the case throughout the document. If the start state is found in the set derived from the calculus, one may take a closer look and see at which state and by which order of operations it has been added, possibly leading to a protocol, if one switches to the non-reduced version of the calculus. If the process constitutes a search for finite-runtime protocols, the protocol can be directly derived from the steps. Otherwise, we can at least guarantee a restarting Las Vegas protocol, as we can recover at least one protocol path leading to a final state. All branches which leave this path could be replaced with a restart operation, giving a complete protocol.

Remark 9.1. The closures are well-defined, i.e., they do not depend on the order of turn and shuffle backwards steps. Moreover, $\mathsf{shuf}_{*}^{-1}(\cdot)$, $\mathsf{turn}_{\star}^{-1}(\cdot)$ and $\mathsf{cl}_{\star,*}(\cdot)$ are inclusion-monotone functions.

In impossibility proofs we can use the monotonicity to deliberately enlarge the sets of states, as long as the impossibility still holds. This helps to simplify the proofs.

In the following we derive two lemmas which make calculating $\mathsf{turn}_\star^{-1}(\cdot)$ and $\mathsf{shuf}_*^{-1}(\cdot)$ easier. They are based on the simple fact that states that are not reachable by a turn or shuffle operation need not be considered when determining the respective backwards calculus state sets.

Lemma 9.7. *Let $\mathcal{G}$ be a set of states, and let $\mathsf{cc}(\mathcal{G})$ be the subset of $\mathcal{G}$ with states that have a constant column. Then, $\mathsf{turn}_\star^{-1}(\mathcal{G}) = \mathcal{G} \cup \mathsf{turn}_\star^{-1}(\mathsf{cc}(\mathcal{G}))$ for $\star \in \{\emptyset, \mathrm{r}, \mathrm{f}\}$.*

Proof. Monotony of $\mathsf{turn}_\star^{-1}(\cdot)$, $\mathsf{cc}(\mathcal{G}) \subseteq \mathcal{G}$, and $\mathcal{G} \subseteq \mathsf{turn}_\star^{-1}(\mathcal{G})$ directly implies $\mathcal{G} \cup \mathsf{turn}_\star^{-1}(\mathsf{cc}(\mathcal{G})) \subseteq \mathsf{turn}_\star^{-1}(\mathcal{G})$. For the other direction, observe that any state μ in $\mathsf{turn}_\star^{-1}(\mathcal{G})$ (not already in $\mathcal{G}$) arises from a state $\mu' \in \mathcal{G}$ which is at one of the branches when turning a position in μ, by definition. For this we need to have $\mu' \in \mathsf{cc}(\mathcal{G})$. □

We derive a similar statement for reverse shuffle steps, which will be useful in the proof of Theorem 9.6. In the following, a state is *generateable via uniform closed shuffles*, if it has a partition that is non-strictly coarser than that of a permissible orbit partition of a subgroup Π, and *generateable via closed shuffles*, if it has a *type* partition that is non-strictly coarser than that of a permissible orbit partition of a subgroup Π.

Lemma 9.8. *Let $\mathcal{G}$ be a set of states, and let $\mathsf{gen}_{\mathrm{c}}(\mathcal{G})$ and $\mathsf{gen}_{\mathrm{uc}}(\mathcal{G})$ be the subset of $\mathcal{G}$ that contains all states that are generateable via a closed and uniform closed shuffles, respectively. Then,*

$$\mathsf{shuf}_*^{-1}(\mathcal{G}) = \mathcal{G} \cup \mathsf{shuf}_*^{-1}(\mathsf{gen}_*(\mathcal{G})), \text{ for } * \in \{\mathrm{c}, \mathrm{uc}\}.$$

Proof. For $\mathcal{G} \cup \mathsf{shuf}_*^{-1}(\mathsf{gen}_*(\mathcal{G})) \subseteq \mathsf{shuf}_*^{-1}(\mathcal{G})$ we use that $\mathsf{shuf}_*^{-1}(\cdot)$ is (inclusion-)monotone, $\mathsf{gen}_*(\mathcal{G}) \subseteq \mathcal{G}$, and $\mathcal{G} \in \mathsf{shuf}_*^{-1}(\mathcal{G})$. The other direction follows, as any state μ in $\mathsf{shuf}_*^{-1}(\mathcal{G})$ must either be in $\mathcal{G}$, or it arises from a state $\mu' \in \mathcal{G}$ such that there is a shuffle of type $*$ that produces μ' from μ. Let Π be the permutation group of such a shuffle. We obtain generateablility of μ' via a shuffle of type $*$ with group Π directly from Lemma 9.6. □

9.10. General Uniform Closed AND Requires Five Cards

In this section, due to [K18], we generalize the impossibility result regarding four-card AND protocols restricted to uniform closed shuffles of Theorem 9.4 to restarting protocols. This, together with the impossibility result of Section 9.11 and the four-card restart-free AND protocol using only uniform (non-closed) shuffles, shows that *restarts are unnecessary* for realizing AND and COPY with a minimal number of cards. This shows that allowing restarts is relatively powerless, except possibly for protocols that directly compute more complex Boolean functions.

The proof demonstrates the developed backwards calculus and how to deal with $\perp$-sequences in impossibility arguments. Before we start, let us note some general facts about orbit partitions on the four-card deck:

Lemma 9.9. *Let $\mathcal{D} = [\heartsuit, \heartsuit, \clubsuit, \clubsuit]$ be a deck of cards, X the set of all (symbol) sequences on $\mathcal{D}$ and Π a non-trivial subgroup of S_4. Then,*

1. *the size of the stabilizer on an $s \in X$ is at most 4.*
2. *if there is an orbit of size 1, then there are exactly two orbits of size 1. In this case, the corresponding sequences have distance 4.*
3. *any orbit partition of X via Π has set sizes i) $(1,1,2,2)$, ii) $(1,1,4)$, iii) $(3,3)$, iv) $(4,2)$ or v) (6). Note that i) corresponds to a partition of maximal fineness. (In total, this upper-bounds the number of orbits to 4.)*

Proof. 1. Let us first show that $|\mathsf{Stab}_s(S_4)| = 4$, the statement then follows from $\mathsf{Stab}_s(\Pi) \subseteq \mathsf{Stab}_s(S_4)$. For this, note that for any sequence on $\mathcal{D}$ there are exactly four permutations that leave the sequence fixed, namely id, the swap of the two $\heartsuit$, the swap of the two $\clubsuit$, and the combination of both.

2. For this, assume there is an orbit of size 1, inhabited by a sequence $s \in X$. This implies that Π is a subset of the stabilizer $\mathsf{Stab}_s(S_4)$ of s. Determine the unique sequence $\overline{s}$ of maximum distance 4 by exchanging $\heartsuit$ and $\clubsuit$, and note that it has the same stabilizer, i.e., $\mathsf{Stab}_s(S_4) = \mathsf{Stab}_{\overline{s}}(S_4)$. We can infer that $\overline{s}$ also has an orbit of size 1.

   By this argument, we can deduce that the number of orbits of size 1 is even. Assume there are two distinct sequences s_1, s_2, such that $s_2 \neq \overline{s_1}$. Hence, $s_1, s_2, \overline{s_1}, \overline{s_2}$ are four distinct sequences and s_1, s_2 have distance 2. Now observe that the stabilizers of s_1 and s_2 have trivial intersection, i.e., $\mathsf{Stab}_{s_1}(S_4) \cap \mathsf{Stab}_{s_2}(S_4) = \{\mathsf{id}\}$. As above, note that Π is a subset of *both* stabilizers, leading to $\Pi \subseteq \{\mathsf{id}\}$, which contradicts the assumption that Π was non-trivial. This restricts the number of orbits of size 1 to 2.

3. For the third statement, observe that because of item 2, it remains to exclude the case $(2,2,2)$ and to show that all five orbit set sizes are attainable by giving an example. Let us assume for a contradiction that the orbit partition has set sizes $(2,2,2)$. Then, it holds for all $s \in X$ and all $\pi \in \Pi$ that $\pi^2(s) \in \{s, \pi(s)\}$ and hence $\pi^2(s) = s$. This means that in the cycle decomposition of any $\pi \in \Pi$, there cannot be cycles of length 3 or larger, and hence any $\pi \in \Pi$ has at most order 2. Then, Π is a subset of $\langle (1\ 2), (3\ 4) \rangle$ (up to joint conjugation of the generators) of size 2 or 4. Thus, $\{\heartsuit\heartsuit\clubsuit\clubsuit\}$ is an orbit of size 1, a contradiction.

   To see that the other orbit decompositions are in fact possible, consider the following examples (any joint conjugation of the generators will do as well):

   i) $\langle (1\ 2) \rangle$ and $\langle (1\ 2)(3\ 4) \rangle$ has set sizes $(1,1,2,2)$.

   ii) $\langle (1\ 2), (3\ 4) \rangle$ has set sizes $(1,1,4)$.

   iii) $\langle (1\ 2\ 3) \rangle$ has set sizes $(3,3)$.

   iv) $\langle (1\ 2\ 3\ 4) \rangle$ has set sizes $(2,4)$.

v) S_4 has set sizes (6). □

Using this, and the techniques from Section 9.9 we can now prove the following:

Theorem 9.6. *There is no secure (restarting) four-card AND protocol with deck* $\mathcal{D} = [\clubsuit, \clubsuit, \heartsuit, \heartsuit]$ *in committed format if shuffling is restricted to uniform closed shuffles.*

Proof. As before, we start by iteratively expanding the set of good states, starting from the final states, and in each step adding all the states that are able to reach the good-states-so-far by a shuffle or turn, to the set. We show that this process terminates, and the resulting set does not include the start state, showing that no final state is reachable at all.

Note that, contrary to the proof of Theorem 9.4, we again need to take $\perp$-sequences into account, which complicates the proof quite a bit. Moreover, it is important to make some use of both the uniformity and the closedness of the shuffles, hence to look at the more concrete probabilities of the states, as there are four-card protocols in the closed-shuffle setting (Theorem 8.1), and in the uniform-shuffle setting (Theorem 8.2). We want to avoid working with very concrete probabilities, for reasons of simplicity, hence we make only use of the type-annotated partition of the states, which already carries a lot of information, as introduced in the form the semi-reduced variants of states in Section 9.6.

Moreover, for $\perp$-sequences we may distinguish two versions, namely those, which are assigned a constant probability, by $\perp^{\mathrm{c}}$, and those which are assigned a non-constant (but result-mixed) probability, by $\perp^{\mathrm{nc}}$. For example, $\frac{1}{3}(X_{00} + X_{01} + X_{10} + X_{11})$ is of type $\perp^{\mathrm{c}}$, where as $\frac{1}{3}X_{00} + \frac{2}{3}X_{11}$ would be of type $\perp^{\mathrm{nc}}$. We write $\perp$ if we do not differentiate the types.

As before, our deck is $[\heartsuit, \heartsuit, \clubsuit, \clubsuit]$, and we can form at most six sequences shown in Figure 9.6a.

Figure 9.5.:

♡	♡	♣	♣
♡	♣	♡	♣
♣	♡	♡	♣
♡	♣	♣	♡
♣	♡	♣	♡
♣	♣	♡	♡

(a) All sequences with four cards.

♡	♣	♡	♣	1
♡	♣	♣	♡	1
♣	♡	♡	♣	0
♣	♡	♣	♡	0

(b) A reduced, sequence-maximal final state with four cards, where the first two positions encode the result.

Final and Prefinal States

The state of Figure 9.6b is (up to similarity) the largest final state. Up to similarity, we obtain all final states by choosing a subset of the sequences of this state with the restriction of having at least one 1- and one 0-sequence. The set of final states is denoted by $\mathcal{G}_0$. It is easy to see that the start state is not already final.

As $\mathcal{G}_0$ does not contain any states with a $\bot$-sequence, but each state has at least one 1- and one 0-sequence, any state which shuffles into $\mathcal{G}_0$ contains a subset of the sequences of a state in $\mathcal{G}_0$, with the same restriction of having at least one 1- and one 0-sequence, as we cannot create 0-/1-sequences by a shuffle out of nothing. Hence, $\mathsf{shuf}^{-1}(\mathcal{G}_0) = \mathcal{G}_0$.

Therefore, the prefinal states are the non-final states that reach the set of final states by a turn. W.l.o.g. they look as follows:

♡	♣	♣	♡	1
♣	♡	♣	♡	0
♣	♡	♡	♣	?
♡	♣	♡	♣	?
♣	♣	♡	♡	?

where the sequences marked by ? can be any of $1, 0, \bot$ or empty, as long as there is at least one of the sequences present (otherwise, it would already be a final state). These need to be turnable at the third or fourth column, of course. This general form of the state is as described because the only final states with a constant column are the states with exactly one 1-sequence and one 0-sequence (above the rule), and they can be combined with any state with a constant position (of the other symbol) at the same index, leading to at least one and at most three additional sequences (below the rule).

Note that the start state is not a prefinal state. For this you would need to choose two of the sequences below the rule to be 0 and the other absent. But then the state is no longer turnable, because for this, there would need to be at least one additional 1- or $\bot$-sequence below the rule. For notation, set $\mathcal{G}_1 \coloneqq \mathsf{turn}^{-1}(\mathcal{G}_0)$.

Second Enlargement, by reverse-turn

We enlarge the set of good states ($\mathcal{G}_1$) by the states that can reach them with a turn operation. Denote these by $\mathcal{G}_2 \coloneqq \mathsf{turn}^{-1}(\mathcal{G}_1)$. These look as follows:

♡	♣	♣	♡	1
♣	♡	♣	♡	0
♣	♣	♡	♡	$\bot^{\mathrm{c}}$
♡	♣	♡	♣	?
♣	♡	♡	♣	?
♡	♡	♣	♣	?

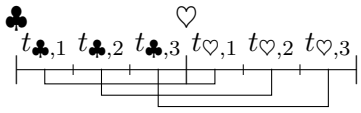
Here again the sequences marked by ? can be any of 1, 0, $\bot$ or empty, as long as there is at least one of the sequences present (otherwise, it would already be a prefinal state). These need to be turnable at the fourth column, of course. (The first three sequences constitute a state with a constant column from $\mathcal{G}_1 \setminus \mathcal{G}_0$, turnable at the third position, if looked at in isolation.) As the newly-added states have a $\bot$-sequence, they cannot be the start state. Moreover, note that an additional reverse-turn on $\mathcal{G}_2$ is futile, as states from $\mathcal{G}_2 \setminus \mathcal{G}_1$ all have at least four sequences, and hence no constant column.

Third Enlargement, by reverse-shuffle

The difficult part is to consider the states which will lead to any of the currently-good states ($\mathcal{G}_2$) via a uniform closed shuffle. Denote them by $\mathcal{G}_3 := \mathsf{shuf}_{\mathrm{uc}}^{-1}(\mathcal{G}_2)$. As id is present in any shuffle, these states are subsets of their resulting states, with the additional possibility of splitting two (or more) $\perp$-sequences into a 1- and a 0-sequence (and possibly more 1-, 0-, or $\perp$-sequences) beforehand. (As in the forward-process two sequences of different types can mix into a $\perp$-sequence.)

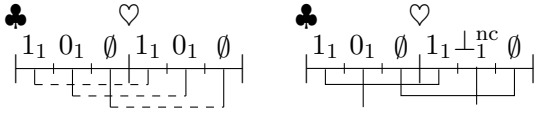
As we saw before, the final states cannot be reached by a shuffle. Hence, by Lemma 9.8, we only need to consider the generateable states from $\mathcal{G}_2 \setminus \mathcal{G}_0$. By Lemma 9.9, the orbit sizes of a finest partition that can be generated are $1, 1, 2, 2$ and there are at most 2 orbits of size 1. Moreover, note that generateable states from $\mathcal{G}_2 \setminus \mathcal{G}_0$ have at least four sequences.[10]

We will use the turn-split representation of a state, as introduced in Section 9.6:



where $t_{\clubsuit,1}, \ldots, t_{\clubsuit,3}$, and $t_{\heartsuit,1}, \ldots, t_{\heartsuit,3}$ are types of sequences denoted by $s_{\clubsuit,1}, \ldots, s_{\clubsuit,3}$, and $s_{\heartsuit,1}, \ldots, s_{\heartsuit,3}$, the first three belonging to the ♣-branch of a possible turn (the state was assumed to be turnable), and the last three at the ♡-branch of a turn. The connecting brackets between the sequences (from which we abstract by this drawing) represent a possible orbit decomposition of a shuffle, namely connected sequences are in the same orbit. A single line drawn downward will mean that the sequence is in an orbit of size 1. By Lemma 9.9, we know that we cannot have both sequences in an orbit of size 1 *on the same side of the turn*, as they would not have distance 4 then.

We think it is instructive to take a closer look at the possible generateable states from $\mathcal{G}_2 \setminus \mathcal{G}_0$. Note that all of these are turnable. Let us do a case distinction on the number of sequences in a state and start with the minimum of *four sequences*. Using our abstract notation observe that we are in one of the following situations:

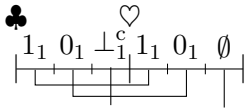


(The dashed lines on the left indicate that we can arbitrarily choose one of the orbits into two, as by Lemma 9.9 having exactly three orbits of size 2 is impossible in our setting). This is because turnability requires $\mu(s_{\clubsuit,1}) + \mu(s_{\clubsuit,2}) = p$ and $\mu(s_{\heartsuit,1}) + \mu(s_{\heartsuit,2}) = 1 - p$, for $p \in [0, 1]$, where $s_{\clubsuit,1}, s_{\clubsuit,2}$ are sequences in one part of the turn branch, and $s_{\heartsuit,1}, s_{\heartsuit,2}$ in the other. Moreover, we can only choose one of the two orbit combinations (where on the right the 0 was w.l.o.g. be chosen to be in the orbit with only one element). Hence,

[10] If it would have only three sequences, then states that reach such a state via a non-trivial shuffle need to have at least one sequence less. But if it has exactly two sequences (one 1-sequence, one 0-sequence), it is already final.

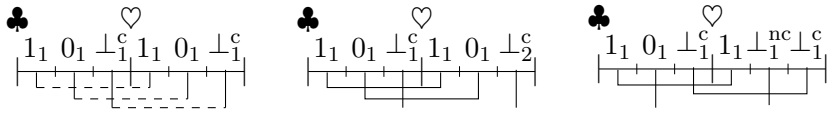
the orbit partition implies that $\mu(s_{\clubsuit,1}) = \mu(s_{\heartsuit,1})$, *or* $\mu(s_{\clubsuit,2}) = \mu(s_{\heartsuit,2})$, let us assume w.l.o.g. the first. Then we can deduce that if $p = 1/2$, we are in the situation to the left, and if $p < 1/2$ in the situation to the right.

In the case of *five sequences*, the only turnable and generateable states look as follows:



This is, similarly to the proof in Section 9.7, because we have to choose the single empty sequence to be in an orbit of size one, forcing the pairs of 1- and 0-sequences each into an orbit of size two. By turnability we have that $\mu(s_{\clubsuit,1}) + \mu(s_{\clubsuit,2}) + \mu(s_{\clubsuit,3}) = p$ and $\mu(s_{\heartsuit,1}) + \mu(s_{\heartsuit,2}) = 1 - p$ for some $p \in [0,1]$. The orbit decomposition implies $\mu(s_{\clubsuit,1}) = \mu(s_{\heartsuit,1})$ and $\mu(s_{\clubsuit,2}) = \mu(s_{\heartsuit,2})$ which, if plugged into the second equation implies that $\mu(s_{\clubsuit,3})$ is a constant, because $\mu(s_{\clubsuit,1}) + \mu(s_{\clubsuit,2}) = 1 - p$ and $\mu(s_{\clubsuit,1}) + \mu(s_{\clubsuit,2}) + \mu(s_{\clubsuit,3}) = 1 - p + \mu(s_{\clubsuit,3}) = p \Leftrightarrow \mu(s_{\clubsuit,3}) = 2p - 1$.

For states with *six sequences*, we are in one of the following configurations:



Note that a resulting state needs a $\perp^{\mathrm{c}}$-sequence, and if a state has two or three 0-sequences and only one 1-sequence, it cannot result in a state with sequences of the types $\perp$, 1 and 0 by a uniform closed shuffle. This is because for $\perp$ you need at least one 1- and one 0-sequence which are mixed together, and one additional 1- and 0-sequence in different orbits which are not mixed together, so that you need at least two 1-sequences.

By the claim, it follows that the start state – which has three 0-sequences and one 1-sequence – would only be added to the set, if it shuffles into a good state *without a $\perp$-sequence*. For this to happen, the start state has to be a proper subset of the resulting state (cf. Lemma 9.2). Given that we are in one of the scenarios as argued above, this cannot happen, as none of the states has three 0-sequences.

A Fourth Enlargement is Futile

Now let us observe that a reverse-turn does not lead to any new states. For this, note that we need to only consider states that have been added in the last enlargement. For a reverse-turn, we can only reach those which have a constant column, and hence have at most three sequences. Note that by the same argument as before, states with two 0-sequences and one 1-sequence or the other way round, *with a constant column* have not been added in the last step, as the only $\perp$-free configuration of the above list cannot have a constant column. (We did not add any new states with a constant column to $\mathcal{G}_2$.) Hence $\mathcal{G}_3 = \mathsf{turn}^{-1}(\mathcal{G}_3)$.

We want to show $\mathcal{G}_3 = \mathsf{shuf}_{\mathrm{uc}}^{-1}(\mathcal{G}_3)$. For this note that no arising state has more than two sequences of the same type (except for the three $\perp$-sequences that may arise in

states with 6 sequences). The additional power of successive $\mathsf{shuf}_{\mathrm{uc}}^{-1}(\cdot)$-steps comes only into play, if there are more than two sequences of the same type, as we are only possibly restricted by the shuffle when taking a subset of the sequences. □

9.11. Closed n-COPY Requires $2n+2$ Cards

This section is taken from [K18]. We apply the techniques from Section 9.9 to show a strong impossibility result for closed-shuffle COPY protocols with $2n+1$ cards, where $n \geq 2$. This implies that in this setting, the protocol of [MS09] using $2n+2$ cards is optimal.

Theorem 9.7. *There is no (possibilistically) secure $(2n+1)$-card n-COPY protocol using only closed shuffles and two colors.*

Proof. We proceed by using the backwards calculus technique as developed in Section 9.9, i.e., we would like to show that the start state is not contained in the closure $\mathsf{cl}_{\mathrm{c}}(\mathcal{G}_0)$, where $\mathcal{G}_0$ is the set of final states in $(2n+1)$-card n-COPY protocols. For this, we apply $\mathsf{turn}^{-1}(\cdot)$ and $\mathsf{shuf}_{\mathrm{c}}^{-1}(\cdot)$ operations to the growing set of states, starting from $\mathcal{G}_0$, until this process becomes stationary. Our analysis will show that this already happens after one reverse turn and one reverse shuffle step. We assume w.l.o.g. that the helping card is ♡, yielding the deck $\mathcal{D} = [(n+1) \cdot \heartsuit, n \cdot \clubsuit]$. Let $\mathcal{G}_0$ be the set of *final states for n-COPY* and note that these look in reduced form (up to similarity of states) like this:

$$\begin{array}{ll} \heartsuit(\clubsuit\heartsuit)^n & 0 \\ \heartsuit(\heartsuit\clubsuit)^n & 1. \end{array}$$

The start state of an n-COPY protocol on deck $\mathcal{D}$ looks w.l.o.g. (up to similarity of states) as follows:

$$\begin{array}{ll} \clubsuit\heartsuit(\heartsuit\clubsuit)^{n-1}\heartsuit & 0 \\ \heartsuit\clubsuit(\heartsuit\clubsuit)^{n-1}\heartsuit & 1 \end{array}$$

As in Theorem 9.2, we know that final states are not reachable by a shuffle, because there is only one 0-sequence and only one 1-sequence, and we cannot take any proper subset of these. Hence, let us look at the set $\mathcal{G}_1 \coloneqq \mathsf{turn}^{-1}(\mathcal{G}_0)$, i.e., the states that are turnable and have a state from $\mathcal{G}_0$ at one of its branches.

The newly added states, i.e., the states from $\mathcal{G}_1 \setminus \mathcal{G}_0$, look (up to similarity) as follows:

$$\begin{array}{ll} f_0' \colon \heartsuit(\clubsuit\heartsuit)^n & 0 \\ f_1' \colon \heartsuit(\heartsuit\clubsuit)^n & 1 \\ \clubsuit\ldots\ldots & ? \\ \vdots & ? \\ \clubsuit\ldots\ldots & ?, \end{array}$$

where ? can be of type 0, 1, $\perp$, and we have at least one sequence starting with ♣ (if it is a $\perp$-sequence; otherwise we have at least one 0- and one 1-sequence, due to turnability). Let us call them *prefinal*. As they have at least three sequences, the start

state is not among the prefinal states. Because they do not have a constant column, a new reverse turn does not yield any additional states, i.e., $\mathcal{G}_1 = \mathsf{turn}^{-1}(\mathcal{G}_1)$. Let us denote the 0-sequence in the first row by f_0', and the 1-sequence in the second row by f_1'.

For the main step of the proof, let $\mathcal{G}_2 \coloneqq \mathsf{shuf}_{\mathrm{c}}^{-1}(\mathcal{G}_1) = \mathsf{shuf}_{\mathrm{c}}^{-1}(\mathcal{G}_1 \setminus \mathcal{G}_0)$. These are the states that reach a prefinal state via a closed shuffle. Note that if f_0' and f_1' is the only 0- and 1-sequence, respectively (in a state $\mu' \in \mathcal{G}_1$), all other sequences need to be of type $\bot$ and any subset of the state (i.e., a $\mu \in \mathcal{G}_2$, as in a reverse shuffle) needs to contain at least f_0', f_1'. These states are again prefinal or final.

Hence, we assume w.l.o.g. that there are at least two 0-sequences (i.e., the type partition of type 0 has more than one sequence), and assume that the shuffle step creates one of these. More formally, let Π be the shuffle subgroup from a state $\mu \in \mathcal{G}_2$ to a prefinal state $\mu' \in \mathcal{G}_1 \setminus \mathcal{G}_0$ as specified above, such that μ' contains at least one additional 0-sequence, which we can also w.l.o.g. assume to be f_0'. Let $f_0 \in \mu$, such that there is a $\tilde{\pi} \in \Pi$ with $\tilde{\pi}(f_0) = f_0'$, and let us consider the state $\tilde{\mu} \coloneqq \tilde{\pi}(\mu)$ that is similar to μ but contains f_0'. Due to the closedness of the shuffle, it is retained that $\tilde{\mu}$ shuffles to μ' via Π.

We can deduce that $\tilde{\mu}$ looks as follows:

$$\begin{array}{ll} \heartsuit(\clubsuit\heartsuit)^n & 0 \\ \clubsuit\ldots\ldots & ? \\ \vdots & ? \\ \clubsuit\ldots\ldots & ?, \end{array}$$

where none of the other rows contains a $\heartsuit$ in the first position. Assume the contrary, namely that there is a sequence $s \neq f_0'$, with a $\heartsuit$ in the first position. If it would be of type 0 or $\bot$, then this does not shuffle into μ', as $\mathsf{id} \in \Pi$ and there is no other 0-sequence in μ'. Otherwise, an s of type 1 would mean that either $\tilde{\mu}$ is identical to μ' (in the case that $s = f_1'$), or similarly to before this does not shuffle into μ' via Π. Hence, the form is as specified above.

Note that another reverse shuffle does not yield any new states. This is because the states we described did not assume anything about the concrete orbit partition, and taking another subset of the sequences does not help.

Hence, it remains to show that we cannot reach the states in $\mathcal{G}_2 \setminus \mathcal{G}_1$ via a turn, i.e., that $\mathcal{G}_2 = \mathsf{turn}^{-1}(\mathcal{G}_2)$. We proceed by showing that the newly added states from $\mathcal{G}_2 \setminus \mathcal{G}_1$ do not have a constant column. For this, let us assume the contrary, namely there is a constant column, at position $p \neq 1$. Let us distinguish two cases by the parity of p:

Case 1: $p = 2m + 2$ is even. In this case we get a constant $\clubsuit$ column, and the state $\tilde{\mu}$ looks like this (with at least two sequences in total):

$$\begin{array}{ll} f_0' \colon \heartsuit(\clubsuit\heartsuit)^m\clubsuit\heartsuit(\clubsuit\heartsuit)^{n-m-1} & 0 \\ f_1 \colon \clubsuit\ldots\ldots\ \clubsuit\ldots\ldots\ldots\ldots & 1 \\ \vdots & ? \\ \clubsuit\ldots\ldots\ \clubsuit\ldots\ldots\ldots\ldots & ? \end{array}$$

Let $f_1 \in \tilde{\mu}$ be a sequence that is mapped to $f_1' = \heartsuit(\heartsuit\clubsuit)^n \in \mu'$ by some $\pi \in \Pi$. As indicated, because of the constant column, f_1 has at least two $\clubsuit$s which are in positions, where there is a $\heartsuit$ in f_1', namely positions 1 and p. Let us look at π^{-1}, which is also in Π due to the closedness of the shuffle, and which maps f_1' to f_1.

For the general proof idea, we show that π^{-1} needs to have certain features which imply that it maps *the* 0*-sequence* f_0' to a sequence of type 0 with a $\heartsuit$ at positions 1 *and* p, leading to the contradiction, as the only 0-sequence with a $\heartsuit$ in the first position has to be f_0', which has a $\clubsuit$ at even positions.

For this, note that to get $\clubsuit$s into positions 1 and p via π^{-1}, there need to be odd positions $u, v \neq 1$, (which contain $\clubsuit$ in $f_1' = \heartsuit(\heartsuit\clubsuit)^n$,) such that $\pi^{-1}(u) = p$, and $\pi^{-1}(v) = 1$. Let us look at the sequence $s' = \pi^{-1}(f_0')$. f_0' has a $\heartsuit$ at all the odd positions, hence, s' does have a $\heartsuit$ in positions 1 and p, leading to the contradiction as discussed above.

Case 2: $p = 2m + 3$ is odd. In this case we get a constant $\heartsuit$-column, and the state looks like this:

$$\begin{array}{ll} f_0' \colon \heartsuit(\clubsuit\heartsuit)^m\clubsuit\heartsuit(\clubsuit\heartsuit)^{n-m-1} & 0 \\ f_1 \colon \clubsuit\ldots\ldots\ldots\ \heartsuit\ldots\ldots\ldots\ldots & 1 \\ \quad\vdots & ? \\ \quad\clubsuit\ldots\ldots\ldots\ \heartsuit\ldots\ldots\ldots\ldots & ? \end{array}$$

Our argumentation is similar to above, but slightly more involved, as f_0' has a $\heartsuit$ also in the first position. As before let $f_1 \in \tilde{\mu}$ be a sequence that is mapped to $f_1' = \heartsuit(\heartsuit\clubsuit)^n \in \mu'$ by some $\pi \in \Pi$. Here, f_1 has a $\clubsuit$ at the first position which needs to be a $\heartsuit$ in f_1', and a $\heartsuit$ at the odd position p, which should become a $\clubsuit$ through π. Again, $\pi^{-1} \in \Pi$ and maps f_1' to f_1.

To fulfill this, we need a position u which is even or 1, such that $\pi^{-1}(u) = p$, and an odd position $v \neq 1$, such that $\pi^{-1}(v) = 1$. Here again, let us analyze $s' = \pi^{-1}(f_0')$. s' is a 0-sequence with a $\heartsuit$ in the first position. If u would be even, then s' has a $\clubsuit$ in position p, already leading to the same contradiction as above. Hence, $u = 1$. Then, for prefinality of μ', $s' = f_0'$ (otherwise, we would have two 0-sequences, starting with $\heartsuit$), and $\pi, \pi^{-1} \in \mathsf{Stab}_{f_0'}(S_{2n+1})$.

In this case, we can deduce that $f_1 = \clubsuit(\heartsuit\clubsuit)^m\heartsuit\heartsuit(\heartsuit\clubsuit)^{n-m-1}$, i.e., that the state $\tilde{\mu}$ looks more closely as follows:

$$\begin{array}{ll} f_0' \colon \heartsuit(\clubsuit\heartsuit)^m\clubsuit\heartsuit(\clubsuit\heartsuit)^{n-m-1} & 0 \\ f_1 \colon \clubsuit(\heartsuit\clubsuit)^m\heartsuit\heartsuit(\heartsuit\clubsuit)^{n-m-1} & 1 \\ \quad\clubsuit\ldots\ldots\ldots\ \heartsuit\ldots\ldots\ldots\ldots & ? \\ \quad\vdots & ? \\ \quad\clubsuit\ldots\ldots\ldots\ \heartsuit\ldots\ldots\ldots\ldots & ? \end{array}$$

where we have at least three sequences, as otherwise we would have a final state.

By Lemma 9.3, because $\pi^{-1}(1) = p$ as described above, after shuffling with Π, we obtain the same number of $\heartsuit$, and $\clubsuit$ in columns 1 and p in the resulting prefinal state

μ'. Because the prefinal state has exactly 2 ♡s in the first column, the same holds for column p in the prefinal states reachable from $\tilde{\mu}$ via Π. The same has to hold for $\tilde{\mu}$, i.e., in column p only two ♡ are allowed. (To see this, note that Π contains id and, hence, the sequences in $\tilde{\mu}$ form a subset of the sequences of μ'). But by assumption this is the constant column which may only contain ♡s. Hence, there would be at most two sequences in $\tilde{\mu}$. As discussed before, this would then be already final.

As the start state has a constant column, contrary to the states in $\mathcal{G}_2 \setminus \mathcal{G}_0$, it is not contained in the closure $\mathsf{cl}_\mathrm{c}(\mathcal{G}_0)$. Hence, no final state is reachable from the start state. □

9.12. Lower Bounds for Protocols using a Standard Deck

In this section we give our impossibility results concerning standard decks. This section is taken from [KSK19].

Theorem 9.8. *There is no four-card finite-runtime base conversion protocol for overlapping bases with deck* $\mathcal{D} = [1, 2, 3, 4]$.

Proof. We proceed by using the backwards calculus technique from Section 9.9. That is, we show that if we start with the set of (highly-structured) final states $\mathcal{G}_0$ of base conversion protocols and enlarge this set iteratively by states which reach the given states by a shuffle or a turn operation, 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. Then, it remains to show that the start state is not contained in the closure.

We assume w.l.o.g. that the input basis is $\{1, 2\}$ with helping cards 3 and 4, and that the output basis is $\{o_1 < o_2\}$ such that $|\{1, 2\} \cap \{o_1, o_2\}| = 1$. For simplicity, we want the output basis to be $\{1, 3\}$ and argue later why this choice was inconsequential for 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

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

We have that $\mathsf{shuf}^{-1}(\mathcal{G}_0) = \mathcal{G}_0$, i.e., shuffling steps do not help in the last step of an 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 Lemma 9.7, 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. These states look as follows:

13 24	0
31 24	1

13 42	0
31 42	1

However, we aim to enlarge this set (which we can do because this only makes our claim stronger due to monotonicity of the backwards operations) by the following two 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 [M16b, 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 $\mathcal{G}_3 := \mathsf{turn}_\mathrm{f}^{-1}(\mathcal{G}_2)$. Note that 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 contained in $\mathcal{G}_2$ finishes the proof.

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

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

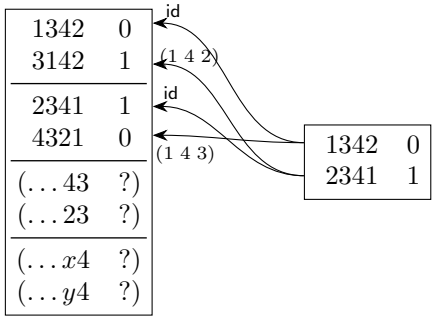
$\ldots da$	0
$\ldots da$	1
$\ldots db$	?
$(\ldots ab$	?)
$\ldots dc$	?
$(\ldots ac$	?)
$(\ldots xd$	?)
$(\ldots yd$	?)

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 which would interfere with our wish to have a constant column in this position (put in parenthesis in the state 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 the state $\mu \in \mathsf{cc}(\mathcal{G}_2)$.

Let us consider only the first case, the other cases are analogous. Let μ contain *two sequences*. If they would be both from the first block, the state would trivially be in $\mathsf{cc}(\mathcal{G}_0)$. This leaves us with two choices, either to include a sequence ending with da or to exclude it. For concreteness, we choose w.l.o.g. $a = 2$, $d = 4$, $b = 1$ and $c = 3$.

In this case it looks like this:



Reaching this state on the left by a shuffle should contain at least $\{\mathsf{id}, (1\ 4\ 3), (1\ 4\ 2)\}$. But applying (1 4 2) to the first sequence 1342 gives sequence 3241, which is not contained on the left side (it has a 1 in the end, but is not in the second block), leading to a contradiction. The other cases are similar. □

Theorem 9.9. *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 9.8, because if there would be such an AND protocol, by fixing the second

bit to 1 one could easily generate a base 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 base convert, as the bit remains unchanged. In this case, a close analysis of the proof of Theorem 9.8 above yields that the theorem also holds in this case. We omit the details. □

9.13. Conclusion

Let us conclude, using a merge of conclusions from [KWH15; KKW$^+$17; KSK19; K18]. To summarize our results, we have extensively considered the question on tight lower bounds on the number of cards for AND protocols, which has been open for several years. Moreover, we extended the analysis on the necessary and sufficient number of cards in card-based protocols computing an AND in committed format to certain plausible restrictions on the operations that can be performed during a protocol run. These are restrictions to certain forms of shuffling, namely closed and/or uniform shuffles and whether running in loops or restarting is allowed. This focus allows to get a clearer view on how many cards are necessary in protocols with *favorable* properties, such as finite running time or easy-to-do/actively secure shuffling. It is for example useful to now be aware that a search for five-card finite-runtime protocols using only closed shuffles will be fruitless – and thereby identifying the six-card protocol by [MS09] as optimal w.r.t. closed-shuffle protocols. In the process, we highlight interesting properties from which the orbit partitions are the most useful. Furthermore, we extended the four-card impossibility result of [KWH15] to the case of uniform closed shuffles for restart-free Las Vegas protocols.

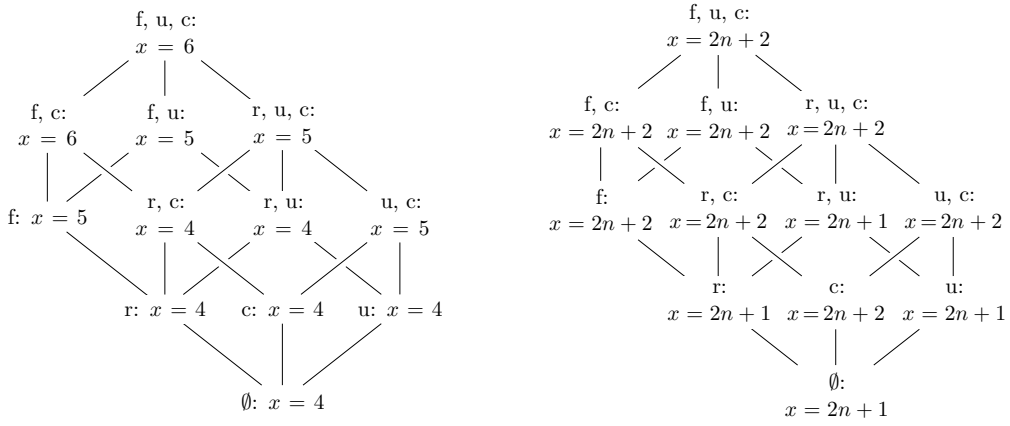
For bit copy, we proved that the $(2n + 1)$-card COPY protocol of [NNH$^+$18] is card-minimal, and the $(2n + 2)$-card COPY protocol of [MS09] is card-minimal w.r.t. finite-runtime protocols or closed-shuffle protocols. Figure 9.7 summarizes our results and surveys current bounds on the number of cards for all combinations of restrictions.

Hence, for the two central building blocks in composite card-based protocols, AND and COPY, we complete the picture on the necessary number of cards in committed format protocols with respect to all combinations of practicality requirements, in the two-color deck setting. With this work, all bounds are tight, as shown in Figure 9.7 and Table 17.1. Hence, this work provides a reference, showing the best protocol for a used setting, and one can compare whether to trade fewer cards with different characteristics of the running time or shuffles.

For the four-card standard deck setting, we showed that there is no finite-runtime AND protocol, if the output basis is fixed in advance, regardless of the shuffle operations used. Finally, we showed tight lower bounds on base conversions for single bits.

Open Problems

An interesting remaining open problem is whether a helping card of an additional color, say $\diamondsuit$, may circumvent some of the impossibility results. More specifically, we would be keen to know whether there is a five-card AND protocol with (uniform) closed



(a) AND protocols

(b) COPY protocols

Figure 9.7.: The number of cards necessary and sufficient for committed format protocols for AND and COPY, given as a Hasse diagram. The nodes/corners specify the different requirement combinations: f is finite-runtime, r is restart-free LV, c and u are restrictions to closed and uniform shuffles, respectively. A line between two nodes in the lattice specifies that the configuration given by the node above has more restrictions (requiring at least as many cards).

shuffles and finite running time on deck $\mathcal{D} = [\clubsuit, \clubsuit, \heartsuit, \heartsuit, \diamondsuit]$, and whether there is a $(2n + 1)$-card n-COPY protocol with a helping card of a third color using only closed shuffles. This is not clear, as the number of possible sequences in a state (and hence the number of possible states) grows immensely with an additional color, but we are also more restricted in turning as we learn more information by observing a symbol. (Note that in contrast to the standard deck setting, where we have more colors on principle, here, we want that the output- and input bases still use only $\clubsuit, \heartsuit$.)

10. Formal Verification of Run-Minimality

This chapter is taken from [KSK19] on applying formal verification to the area of card-based cryptography.

Prior to this thesis, the literature offers only three protocols and no proofs for non-trivial lower bounds on the number of cards in the case of a *standard* deck. 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 chapter, we employ the technique of software bounded model checking (SBMC), which reduces the problem to a bounded state space. This state space is then automatically searched exhaustively using a SAT solver as a backend.

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 from Protocol 8.4 to have a shortest run for protocols using this number of cards.

Related Work on Formal Methods. To the best of our knowledge, this is the first work which applies formal methods to the field of card-based cryptography. However, a wide range of research has been done using formal methods in the more general field of secure two-party and multiparty computations. In general, 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 structures 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 [APS14], 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]. This language is then used to implement MPC programs with enabled formal verification provided by the semantics of the language.

10.1. 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. Let us start by describing the general technique of using bounded model checking 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 in 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 such an entry.

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.

10.2. Automatic Formal Verification for Card-based Protocols

Our approach employs a standardized program representation of the state trees introduced in Chapter 7. This allows a general programmatic encoding of both shuffle and turn operations, as well as of 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) helping 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[1] 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 the 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 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 state 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 state 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[2] 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.

10.3. An Illustration of Our Verification Methodology

In the following, we exemplify our translation of card-based cryptographic protocols using standard decks to the specific bounded model checker CBMC, which takes programs in the C language, and consider the case of computing a secure AND function. For our

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

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

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

Listing 1: C struct holding the state trees.

experiments, we used CBMC 5.11 [CKL04] with the built-in solver based on the SAT-solver MiniSat 2.2.0 [ES04]. 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 state 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. We subsequently repeatedly (after each turn/shuffle operation) check whether all possible resulting (non-deterministic) states correctly and securely compute the specified function, e.g., in our case a secure AND.

An example shuffle operation is shown in Listing 2 for the case of output-possibilistic security. Herein, 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 special 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

```
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.

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, on the above-mentioned platform. The constructed formulas contain between 150 and 180 million SAT clauses.

10.4. Conclusion

In this chapter, 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.

Open Problems

We want to point out some open problems that could be approached based on the findings in this work:

1. For finite-runtime protocols, there exist no proven *tight* lower bounds on the required number of cards (between five and eight cards) in the standard deck setting. 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. It would certainly be interesting to identify and exploit additional symmetries in the state space. For example, it would be interesting whether there is a *normal form* that is unique for every state and into which any state can be brought via perm and relabel operations in polynomial time.

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 running time setting). For such a mixed setting, [SM19] provide nice results to build upon in further research.

11. Private Function Evaluation with Cards

This chapter is directly based on [KW18], with some adaptations. Recall that Secure Multiparty Computation (MPC) allows multiple players to jointly compute a function, without giving away anything about their inputs, except what can be deduced from the output. An important special case is when the function to be evaluated constitutes an input itself and should remain hidden, called *Private Function Evaluation* (PFE). This has been considered in the standard cryptographic setting e.g., using universal circuits [V76] in [MS13; LMS16; BBKL17; GKS17]. The aim of this chapter is to obtain conceptually simple protocols for PFE, and also obfuscation with a deck of cards.

Motivation

Card-based protocols are often used in educational and recreational settings. For an illustration of PFE, we stretch the usual motivation for card-based AND protocols a bit, namely the dating problem where players want to find out whether there is mutual love.

We assume a predefined set of binary attributes A such as $A = \{\texttt{LikesCats}, \texttt{HasPhD}, \texttt{IsGeeky}, \dots\}$. Alice implicitly specifies (by providing a circuit or program) which combinations $P \subseteq 2^A$ of attributes she likes and Bob specifies which attributes $B \subseteq A$ he has. The task is to determine whether Bob's secret attributes satisfy Alice's secret preferences, i.e., whether $B \in P$. Here, we want to ensure that both Alice's and Bob's input remains hidden, i.e., nothing about the input is revealed, except what can be deduced from the output of the protocol.

In the same vein, PFE is useful for the game *Skipjack* [D15][1], where a game master invents a rule and the other players take turns querying whether a chosen code words satisfies the rule or not – in order to deduce/guess the rule in this process. Applying our PFE protocol would allow to prevent the game master from cheating by changing the rule mid-game, or even to play the game in absence of a game master, assuming an encoding of a rule is available or can be obtained at random. (Moreover, as PFE even hides the code words that the player is testing, we can derive a competitive multi-player mode where questions of other players do not help the others.) Besides these, there is also a purely theoretical motivation of constructing protocols for PFE and obfuscation.

Look and Feel of Our Protocols

Imagine a room with a table, where Alice puts an encoding of a function f in a sequence on the table, each bit of the description as two face-down cards encoding 0 via ♣♡ and

[1] a follow-up on a game by Abbott [A] from 1956. Skipjack was given as a present to all participants of ASIACRYPT 2015

1 via ♡♣. Next to Alice's cards, Bob will put his input x as a bit string using the same encoding. We then proceed according to a protocol that may prescribe to shuffle or turn over cards (the observed symbols may affect the future course of the protocol), as before. The protocol terminates with output $f(x)$ encoded as face-down cards. The output can then be revealed to the players or used obliviously in further computations.

On Interaction in Card-based Protocols

We point out that card-based cryptography can be assumed secure in a rather *non-interactive* physical model: it suffices to have one protocol executer, who is under surveillance by the other players. For example, when the protocol description specifies that a certain shuffle is to be performed, this step can be implemented by this one player, the executer, who uses envelopes (or helping cards) and completely random shuffles or uniform random cuts in a manner that ensures that not even he himself can keep track of the concrete permutation done to the cards. We could also use shuffling machines, such as the turntable in [V14].

Note that in this *surveillance model* where players watch that the protocol is done correctly, many protocols can be argued secure with almost no interaction. For example, [GNPR07, Protocol 3] is a nice physical zero-knowledge proof system for proving that there is a solution to a Sudoku puzzle, where the verifier chooses one of three cards in each cells of the Sudoku to be assigned to piles for rows, columns and subgrids to be able to later verify that all numbers are present. In our model, we can plausibly argue that the randomness chosen by the verifier can also be directly generated by the prover himself on an additional deck of helping cards. If he is watched to perform the shuffle in a way that generates high entropy not under his control, he can use this generated randomness to assign the cards to the piles. This is actually a general observation regarding protocols using public coins, where this shuffling produces an output that can be interpreted to be like the Random Oracle output in the Fiat–Shamir heuristic. The possibility of secure shuffling in this way is a common assumption that people make when playing card games with others. This is similar to *public observation* protocols due to Fisch, Freund, and Naor [FFN14].

Using the PFE protocols introduced in this chapter, this immediately leads to a direct way to obtain cryptographic *obfuscation* in this card-based surveillance model: Assuming that the encoded protocol is lying on the table using cards, the executer can add cards encoding the inputs and then execute a universal protocol, such as the ones proposed in this chapter, with *the only interaction* being guards that watch out for publicly observable deviations from the protocol. However, note that because of the very different setting, there are no implications for the usual non-physical cryptographic world, where general (virtual black-box) obfuscation is impossible, cf. [BGI+01].

Universal Protocols and Their Qualities

We implement four different universal card-based protocols with varying degrees of abstraction, based on branching programs, circuits, Turing machines and RAM machines.

Our primary focus is on simplicity and elegance of the protocols, but we also consider efficiency in terms of running time and required cards.

The benefit of providing several solutions is that, depending on the nature of the task, a certain computational model may be particularly suitable. For example, in the generalized dating game described above, using universal circuits is a natural option, while a rule in Skipjack might most naturally be described as a program using loops and thus benefit from the possibilities available in Turing machines and RAM machines. For didactic settings, all options are interesting in itself, as they demonstrate the computational models and the implemented privacy properties in a palpable way.

Contribution

In this chapter, we

- show how to encode and execute circuits, Turing machines, RAM machines and branching programs with cards and specify protocols for executing these on hidden inputs so that nothing about the machine description (except the length, etc.) or the inputs is leaked. We achieve this by using envelopes and only very natural shuffle operations, namely random cuts and S_n-shuffles (i.e., ordinary shuffling, where all card reorderings are equally likely).

- we thereby obtain what may be called cryptographic obfuscation in a card-based setting, given the weakly interactive nature of card-based cryptography in the "surveillance model" (see above).

Related Work

Regarding our branching program construction, let us mention that there are several card-based protocols to randomly generate a permutation with specific, prescribed properties. For example, the secret santa game asks for random permutations on the player indices (encoding who gives a present to whom) that are fixed-point free to ensure that nobody receives their own present, and has been implemented with cards in [CK94; ICM15]. Moreover, they also give protocols for generating permutations with cycles of a certain minimal length. Moreover, Hashimoto et al. [HSN+17] give a protocol for generating permutations with a prespecified cycle structure, and show how to obliviously execute the inverse of a permutation encoded with cards on another card sequence, which is a special case of our sorting operations.

Note that cryptographic obfuscation has been performed in other models. For example, Goyal et al. [GIS+10] make use of tamper-proof hardware tokens (such as smart cards) introduced by Katz [K07]. Moreover, [MN10a] allows to execute many cryptographic primitives (albeit not obfuscation) using scratch-off cards. They have a slightly weaker setting, as they do not gather players around a table, but use sealed (tamper-evident) envelopes that are sent between the players via mail.

Crépeau and Kilian [CK94] also discuss playing games against a card-encoded (probabilistic) circuit opponent. However, they do not aim to hide this circuit to the player as it is given by the player himself.

Preliminaries: Boolean Circuits

A *Boolean circuit* with k input variables $v_1, \ldots, v_k$ is a directed acyclic graph $C = (V, E)$. The nodes are called gates and are labeled with $\vee$, $\wedge$, $\neg$, an input variable, or one of the constants 1 or 0. In the cases of $\vee$, $\wedge$, $\neg$, the in-degree must be 2, 2 or 1, respectively, otherwise it is 0. The *output node* is the unique node with out-degree 0. The *depth* of C is the maximum number of $\wedge$ and $\vee$ gates on a path in C.

The value $C(\vec{v}) \in \{0,1\}$ that a circuit outputs on input $\vec{v} = (v_1, \ldots, v_k) \in \{0,1\}^k$ is defined in the natural way. Here, it is convenient to transform all $\vee$-gates into $\wedge$-gates using de Morgan's rule $(x \vee y) = \neg(\neg x \wedge \neg y)$. Note that this transformation does not affect the depth of the circuit.

11.1. Securely Evaluating a Universal Circuit

Let us start with the most direct case, namely implementing PFE using universal circuits, first constructed by Valiant [V76]. We do not want to go into the details of the construction and just import facts about the general structure of the circuit and how it is used. In our examples, Alice provides her private function, here as a circuit C, and Bob his private input to the function, and it should hold that neither party learns anything about the other's respective secrets. The universal circuit U_n for circuits of size n takes as input an encoding $\langle C \rangle$ of C, where C has size n, and an input $I \in \{0,1\}^k$ of length k. We assume C to have fan-out and fan-in at most 2, i.e., each gate has at most two inputs and at most two outputs.

In the constructions by Valiant, U_n is described via an directed acyclic graph with $O(n \log n)$ vertices, where each vertex represents a logic gate taking values on its incoming edges as well as certain "configuration" (or programming) bits as input and computes outputs emitted to its outgoing edges. More concretely, U_n contains the following types of nodes:

- n *universal gates* with in- and out-degree exactly two and four configuration bits $c_1, \ldots, c_4$ that compute

  $$\mathsf{ug}(c_1, c_2, c_3, c_4, x, y) = c_1\bar{x}\bar{y} + c_2\bar{x}y + c_3x\bar{y} + c_4xy$$

  where $c_1, \ldots, c_4$ determine the Boolean operation performed at this gate, e.g., AND corresponds to $(c_1, \ldots, c_4) = (0, 0, 0, 1)$.

- $O(n \log n)$ *X-switches* with a configuration bit c and in- and out-degree two, that compute

  $$\mathsf{x}(c, a_0, a_1) = (a_c, a_{1-c}),$$

  where a_c is forwarded on one outgoing edge and a_{1-c} on the other.

- $O(n)$ *Y-switches* computing

$$\mathsf{y}(c, a_0, a_1) = a_c,$$

  where Alice's configuration bit c decides which of the two inputs is forwarded as the output.

- $O(n)$ *forks* (or "λ-switches") where the signal on one wire is forwarded to both outgoing wires, i.e., $\lambda(a) = (a, a)$.

- k input nodes with out-degree 1 and in-degree 0, and one output node with in-degree 1 and out-degree 0 with their natural interpretation.

The universal gates correspond to the gates of Alice's circuit with the configuration bits determining what kind of gate it is, and the configuration of X and Y-switches ensures that the intermediate results are routed correctly to the relevant gates. For us, it suffices that there is an (efficient) way to obtain $\langle C \rangle$ from C, which Alice applies beforehand. Valiant [V76] describes such a general mapping from circuits C to a string of $O(n \log n)$ configuration bits for U_n, such that U_n configured with $\langle C \rangle$ (in canonical order) implements C.

We describe in Protocol 11.1 and Theorem 11.1 how, given U_n, encodings of $\langle C \rangle$ and Bob's input I in sequences of cards, we can compute $C(I)$ securely.

Theorem 11.1. *For any $k, n \geq 0$ there exists a secure card-based protocol $\mathcal{P}$ with the following properties:*

1. *The input sequences are all sequences (V, P) where*
   - *V encodes the values of k Boolean variables $(v_1, \ldots, v_k) \in \{0,1\}^k$ using the deck $k \cdot [\clubsuit, \heartsuit]$.*
   - *P encodes a circuit C of size n, via $m = O(n \log n)$ programming bits, i.e., via deck $m \cdot [\clubsuit, \heartsuit]$.*
2. *The output is two cards encoding $C(v_1, \ldots, v_k)$.*
3. *In addition to the input cards, we use the helping deck $(h+1) \cdot [\clubsuit, \heartsuit]$, where $h = O(n)$ is the number of forks in U_n. (The additional pair is used for the* sort^* *command.)*

Proof. $\mathcal{P}$ is given as Protocol 11.1. All nodes of U_n are considered in some topological order $s_1, \ldots, s_N$, allowing us to compute the bits "flowing" along each edge of U_n in a systematic way. The message at an edge e is stored in positions $V_e = (V_e[0], V_e[1])$. Note that the bit on each edge is only used in one subsequent computation: After processing s_i, only the bits on the edges crossing the cut $(\{s_1, \ldots, s_i\}, \{s_{i+1}, \ldots, s_N\})$ are needed in future computations. When processing s_{i+1} we may therefore, when storing the bits for the outgoing edges of s_{i+1}, reuse the now freed up cards that stored the bits on the incoming edges of s_{i+1}. In Protocol 11.1 this is reflected by identifying V_e and $V_{e'}$ for some pairs (e, e') of edges. We only need a new pair of cards in the case of a fork.

To verify correctness, let us interpret the main sort commands in the protocol.

1. In the X-switch case, $\mathsf{sort}\, C_v \uparrow (V_e, V_f)$ swaps the positions encoding the incoming input values at edges e and f, if the configuration bit of the X-switch equals 1 and leaves them unchanged, if it equals 0. This is exactly what we wanted.

2. In the Y-switch case, the command is exactly the same, with the difference that afterwards only the output bit that ends up in the first position (V_e) is used afterwards.

3. In the fork case, we (non-destructively, i.e., with restoring) copy the bit to another position, used as an additional output wire value.

4. The universal gate case is the most interesting. Recall that we want to evaluate $\mathsf{ug}(c_1, c_2, c_3, c_4, x, y) = c_1\bar{x}\bar{y} + c_2\bar{x}y + c_3x\bar{y} + c_4xy$. For this, first observe that exactly one of the terms $\bar{x}\bar{y}$, $\bar{x}y$, $x\bar{y}$, xy equals one. Essentially, the values of x and y select which configuration bit constitutes the output. If $x = 0$ then only c_1 and c_2 are relevant. If $x = 1$ only c_3 and c_4 are. So in the first sorting step we obliviously swap (C_1, C_2) for (C_3, C_4) if $x = 1$ and leave things as is, if $x = 0$. The interesting two configuration bits end up in positions C_1, C_2, without us knowing which they are.

   Now we do the same with C_1, C_2, based on the value of y, so that the only relevant configuration bit is now in C_1. In the last step we write this value in both V_g and V_h (recall the fan-out two requirement).

To see that $\mathcal{P}$ is secure, we use Corollary 8.1 and the fact that no turn operations are performed outside of sorting steps. □

Dependent on the topological ordering used in Protocol 11.1, the helping deck we use to implement forks is not fully required. Instead of using a "fresh" pair of cards to store a copy of the incoming value whenever a fork is encountered, we can reuse cards that have already served their function and will not be used in the remainder of the protocol. This includes, for instance, the cards that encoded configuration bits of universal circuits or X-switches that have already been executed.

Remark 11.1 (Reusability of the Circuit)**.** If we would like to be able to execute the circuit multiple times, we want that the programming bits of Alice's program are not destroyed during the execution. Here, we have to take a little care to ensure that the relevant bits are written back and that conditionally swapped cards are "unswapped" again. For this variant of our algorithm, we replace all sort operations in Protocol 11.1 by their starred variants. In the case of v being a universal gate, we need to take extra care: In the penultimate line of the case, instead of reusing V_e and V_f (which are now in temporary use to swap back the relative positions of the cards containing the configuration bits), we set V_g and V_h as the positions of two new cards, containing ♣♡ as in the fork case. To undo the swaps, we perform $\mathsf{sort}\, V_f \uparrow (C_1, C_2)$ and then $\mathsf{sort}\, V_e \uparrow ((C_1, C_2), (C_3, C_4))$ at the very end of the procedure in the universal gate case. Afterwards, the cards in V_e and V_f may be reused again.

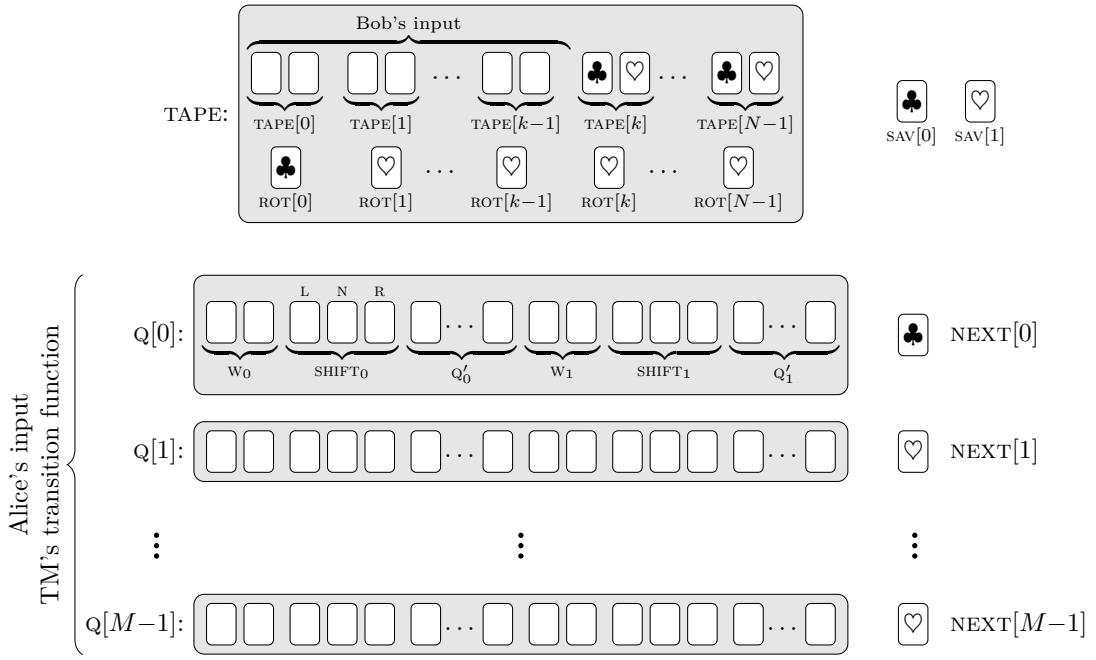


Figure 11.1.: Overview of a run of the universal TM.

11.2. Securely Simulating a Turing Machine

Assume we wish to execute a Turing machine (TM) with a secret encoding provided by one player, Alice, on a secret input provided by another player, Bob. As any secure card protocol uses a fixed number of cards and has a running time which is independent of the input, there must be known bounds on certain parameters of the Turing machine. Let M be a bound on the number of states, N a bound on the number of accessed tape cells and t a bound on the execution time. For simplicity, assume Alice's TM has precisely M states (it can be padded with dummy states), runs t steps ("halting" can be achieved by staying in one state, writing the current tape symbol and not moving) and think of the tape as a cycle of length N (which makes no difference for a TM only ever accessing N memory cells).

All cards (and names for them occurring in the following description) used for our protocol, with the exception of a few helping cards used for sort^* and rot^* operations, are given in Figure 11.1. The encoding of a Turing machine consists of the encoding of its M states. The encoding of each state $q \in \{0, \ldots, M-1\}$ consists of the encoding of two transitions, one for each of the two tape symbols ♡♣ and ♣♡. Take for instance the positions $\mathrm{W}_0, \mathrm{SHIFT}_0 = (\mathrm{L}, \mathrm{N}, \mathrm{R})$ and Q'_0 encoding the transition from state $q = 0$ if the tape symbol is ♣♡. The two cards in positions W_0 contain the tape symbol to be written. The three cards in positions SHIFT_0 specify the movement of the Turing machine head, ♣♡♡ for "left", ♡♡♣ for "right", ♡♣♡ for "no movement" / "halt". Lastly, the M cards in positions Q'_0 contain a unary encoding of $q - q'$ (mod M) where

$q' \in \{0, \ldots, M-1\}$ is the index of the state to be entered next ($\clubsuit\heartsuit\ldots\heartsuit$ encodes 0, $\heartsuit\clubsuit\heartsuit\ldots\heartsuit$ encodes 1, etc.).

The input to the TM, provided by Bob, is encoded in the first k bits of the tape. When executing the Turing machine, the current tape cell will always be in position TAPE[0] and the current state in position Q[0]. Instead of having an explicit moving head we simply rotate the entire tape. Moreover, instead of having an explicit value encoding the current state, we rotate the sequence of states. This is also the reason we encode state index differences in the state transitions instead of absolute indices. The protocol is given as Protocol 11.2 and consists of a loop that does t times the following:

- "read" the tape symbol in position TAPE[0] by conditionally swapping the two transitions in state Q[0] such that the transition that should be done is available in the positions W_0, SHIFT_0 and Q'_0. To undo this operation later, the value of TAPE[0] is also stored temporarily in (SAV[0], SAV[1]).
- the content of TAPE[0], which was reset to 0 in the previous step, is now overwritten with the symbol in position W_0.
- The cards in positions (L, N, R) are used to rotate the $\clubsuit$ of $\mathsf{rot}[0]$ into the positions $\mathsf{rot}[0]$, $\mathsf{rot}[1]$ or $\mathsf{rot}[N-1]$ depending on whether the $\clubsuit$-card among SHIFT_0 is in position N, R or L, respectively. Then the TAPE and ROT cards are rotated together such that the tape cell whose corresponding ROT card is $\clubsuit$ comes to rest in position TAPE[0] (and such that one does not learn which rotation has been performed.)
- The same idea is used to first copy the information about the next state into NEXT$[0 \ldots M-1]$ and then rotate the sequence of all states accordingly. Note that we need to undo the conditional swap of the two transitions in $Q[0]$ before the rotation of the states (using a coupled sorting with (SAV[0], SAV[1])).

Using this protocol idea, we obtain the following theorem.

Theorem 11.2. *For any $k, N, M, t \geq 0$ there exists a secure card-based protocol $\mathcal{P}$ with the following properties:*

1. *The input sequences are all sequences (V, P) where*
   - *V encodes the values of k Boolean variables $(v_1, \ldots, v_k) \in \{0,1\}^k$ using the deck $k \cdot [\clubsuit, \heartsuit]$.*
   - *P encodes a Turing machine T with a state set of size M, using the deck $2M \cdot [3 \cdot \clubsuit, (M+2) \cdot \heartsuit]$.*
2. *The output is a sequence of cards encoding the output of T after running t steps on a cyclic tape of length N initially containing the input $(v_1, \ldots, v_k)$.*
3. *In addition to the cards encoding the inputs, the helping deck $[(N-k+3) \cdot \clubsuit, (M+2N-k-1) \cdot \heartsuit] \cup [\clubsuit, \min\{2, M-1\} \cdot \heartsuit]$ is used. (The latter part is implicit in the use of the starred rot^* commands and not shown in Figure 11.1.)*

foreach *node v of U_n in (some) topological order* **do**
  if *v is an input node* **then**
    let e be the outgoing edge, I_e the positions of the corresponding input bit
    set $V_e \coloneqq I_e$ `// regard the cards at` I_e `to belong to` e
  else if *v is an X-switch* **then**
    let C_v be the position pair of the configuration bit for v
    let e, f be the two incoming edges and g, h the two outgoing edges
    sort $C_v \uparrow (V_e, V_f)$
    set $V_g \coloneqq V_e$ and $V_h \coloneqq V_f$
  else if *v is a Y-switch* **then**
    let C_v be the position pair of the configuration bit for v
    let e, f be the two incoming edges and g the outgoing edge
    sort $C_v \uparrow (V_e, V_f)$
    set $V_g \coloneqq V_e$
  else if *v is a fork* **then**
    let e be the incoming edge and g, h the two outgoing edges
    set $V_g \coloneqq V_e$
    set V_h as the positions of two new cards, containing ♣♡
    sort* $V_e \uparrow V_h$
  else if *v is a universal gate* **then**
    let $C_1, \ldots, C_4$ be the position pairs containing the configuration bits of v
    let e, f be the two incoming edges and g, h the two outgoing edges of v
    sort $V_e \uparrow ((C_1, C_2), (C_3, C_4))$
    sort $V_f \uparrow (C_1, C_2)$
    set $V_g \coloneqq V_e$ and $V_h \coloneqq V_f$
    sort $C_1 \uparrow ((V_g[0], V_h[0]), (V_g[1], V_h[1]))$
  else if *v is an output node* **then**
    let e be the only incoming wire at the output node
    (result, V_e)

Protocol 11.1. $\mathsf{UC}(\langle C \rangle, I)$: executing C on input I

repeat t **times**
  sort $\textsc{tape}[0] \uparrow ((\textsc{w}_0, \textsc{shift}_0, \textsc{q}'_0, \textsc{sav}[0]), (\textsc{w}_1, \textsc{shift}_1, \textsc{q}'_1, \textsc{sav}[1]))$
  sort* $\textsc{w}_0 \uparrow \textsc{tape}[0]$
  rot*$(\textsc{n}, \textsc{l}, \textsc{r}) \uparrow (\textsc{rot}[0], \textsc{rot}[N-1], \textsc{rot}[1])$
  rot $\textsc{rot} \uparrow \textsc{tape}$
  rot* $\textsc{q}'_0 \uparrow \textsc{next}$
  sort $\textsc{sav} \uparrow ((\textsc{w}_0, \textsc{shift}_0, \textsc{q}'_0), (\textsc{w}_1, \textsc{shift}_1, \textsc{q}'_1))$
  rot $\textsc{next} \uparrow \textsc{q}$
result TAPE `// or parts of it`

Protocol 11.2. executeTM()

Proof. The protocol is given in Protocol 11.2 and Figure 11.1. For security, observe that the protocol consists only of sort sub-protocols; we can thus use Corollary 8.1.

For the cards needed, we just count the number of cards depicted in Figure 11.1. In a bit more detail, for the helping cards needed, note that we need $N-k$ pairs of ♣♡ for the empty tape cells, which are placed next to Bob's input string. We have one ♣ for each of the registers ROT, SAV and NEXT, and $N-1$, 1 and $M-1$ ♡s respectively. The second part of the union scales with the size of the largest register to be used in starred commands, which is either SHIFT_0 or Q'_0. □

Remark 11.2 (Variants to the Implementation)**.** Using techniques presented in Section 11.3, we could use a binary instead of a unary encoding of state indices in the encoding of transitions. This would reduce the number of required cards from $O(N+M^2)$ to $O(N+M\log(M))$. However, given that the charm of Turing machines is their simplicity rather than their efficiency, we felt that we should reserve this trick for later.

For simplicity, we also chose to describe how to implement TMs with band alphabet $\{0,1\}$, excluding the special blank symbol ␣. While one can generically map this to the standard case by using an encoding $1 \mathrel{\widehat{=}} 11$, $0 \mathrel{\widehat{=}} 10$, and ␣ $\mathrel{\widehat{=}} 00$, let us briefly discuss how one can easily upgrade our implementation with a TM supporting an additional blank symbol. For this, we encode tape cells with three cards via ♣♡♣ $\mathrel{\widehat{=}} 0$, ♡♣♣ $\mathrel{\widehat{=}} 1$ and ♣♣♡ $\mathrel{\widehat{=}}$ ␣. In this way, the first two cards encode the value as previously, unless they are ♣♣, which would be a blank. We then need to add W_2, SHIFT_2 and Q'_2 to each of the Qs, specifying the operation in the case that a blank symbol is used (Note that the W_i contain the symbol to be written in reversed order, to ensure the right action is done to the tape cards). This approach has the advantage of allowing us to learn the length of the output after the computation (if it is not to be protected), by just turning over the third card in each of the tape cells and outputting (the first two cards of) those cells which do not show a ♡, i.e., which are not blank.

Remark 11.3 (Reusability of the TM)**.** First note that we never destroy any of the state description entries of the TMs code as in normal execution it is always possible to enter the state again. Hence, to be able to run a TM multiple times, we only need to ensure that after the execution the first state is again in Q[0]. As we cannot trust Alice to provide a program that guarantees this behavior, we can introduce an additional register START$[0 \dots M-1]$ which is a copy of NEXT and is rotated together with Q. It can then be used to rotate Q back into its initial configuration by executing rot START $\uparrow$ Q after the loop in Protocol 11.2.

11.3. Securely Simulating a Random Access Machine (RAM)

We now describe a simple bounded Random Access Machine model. The goal is to execute a RAM machine with a secret encoding of the machine specified by one player, Alice, on a secret input provided by another player, Bob.

A Simple RAM Model

We assume fixed constants $N = 2^n$ (memory words), $M = 2^m$ (instruction groups), $k \leq N$ (input size) and $t < \infty$ (time limit). The machine has access to N binary words $\text{RAM}[0], \ldots, \text{RAM}[N-1]$ of length n each, the first k of which contain the input and the remaining $N - k$ contain zero. The following types of instructions are available, where x, y are n-bit words and p is an m-bit word:

Load a Constant.	$\text{RAM}[x] \leftarrow y$
Copy.	$\text{RAM}[x] \leftarrow \text{RAM}[y]$
Indirect Read.	$\text{RAM}[x] \leftarrow \text{RAM}[\text{RAM}[y]]$
Indirect Write.	$\text{RAM}[\text{RAM}[x]] \leftarrow \text{RAM}[y]$
Addition.	$\text{RAM}[x] \leftarrow \text{RAM}[x] + \text{RAM}[y]$
Subtraction.	$\text{RAM}[x] \leftarrow \text{RAM}[x] - \text{RAM}[y]$
Conditional Jump.	$\mathsf{jnz}\,\text{RAM}[x]\ p$

To simplify the implementation step later, we assume that a program is a sequence $\text{I}[0], \ldots, \text{I}[M]$ of *groups* of instructions. Each group of instructions contains precisely one instruction of each of the above types, in canonical order. Note that this fixed instruction order does not affect the strength of the model. Indeed, if we assume that without loss of generality the cell $\text{RAM}[0]$ is never used in any "real" instruction, we may choose $x = y = 0$ to turn any instruction into a dummy instruction that has no effect. By turning all but one desired instruction in each instruction group into such a dummy instruction, we can implement programs without having to worry about the fixed instruction order at the expense of increasing the number of instructions by a constant factor.

Here, the $\mathsf{jnz}\,\text{RAM}[x]\ p$ ("jump if not zero") instruction means that if $\text{RAM}[x]$ contains zero, the execution should continue with the next instruction group. Otherwise, p is to be interpreted as the relative offset to the next instruction group that should be executed, i.e., if the current instruction group has index j, then the instruction group with index $(j + p) \bmod M$ should be executed next.

Implementation with Cards

Assume we want a secure implementation of the RAM model with parameters $N = 2^n$, $M = 2^m$, k, t using playing cards. We may imagine that one player, Alice, provides the sequence of instructions, and the other player, Bob, provides the input in $\text{RAM}[0 \ldots k-1]$ of $k \cdot n$ bits. As usual, each bit is encoded with a pair of cards and a word of n or m bits is a sequence of n or m such pairs. In addition to the inputs, we have an encoding of $\text{RAM}[k \ldots N-1]$ (initially zero) and two additional n-bit "accumulators" A and A' (initially zero). Finally, there are ♡-cards in ("instruction pointer") positions labeled $\text{IP}[1], \ldots, \text{IP}[M-1], \text{IP}^*$ and one ♣-card in the position labeled $\text{IP}[0]$, which will be used for the conditional jumps. An overview is given in Figure 11.2.

We say a few words about the implementation of the instructions, starting with a general description of how words can be loaded from and stored to arbitrary addresses.

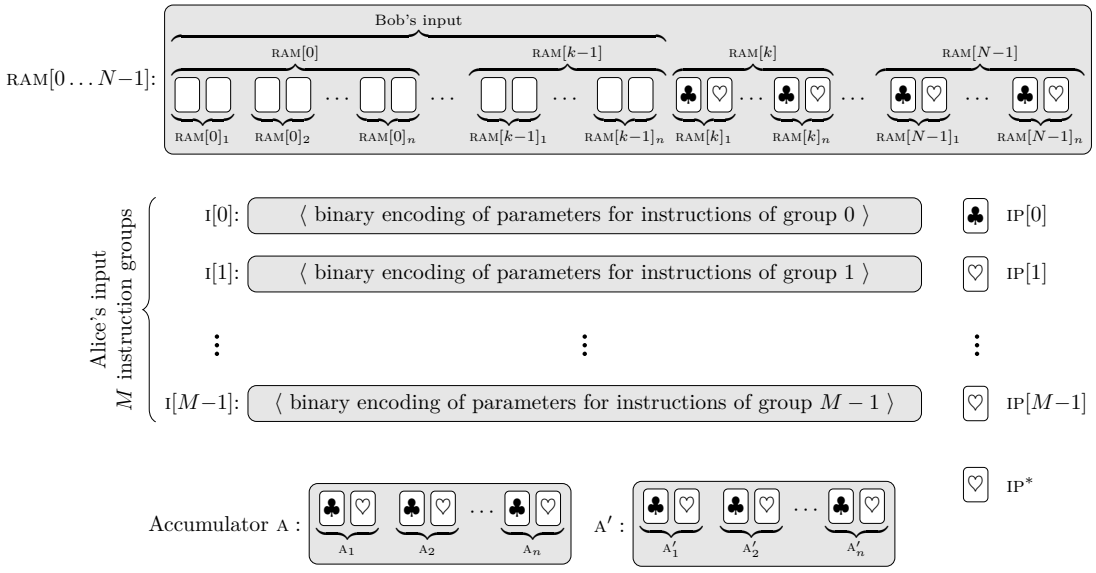


Figure 11.2.: Overview of our RAM machine construction, cf. Protocol 11.5.

Loading a Word. Assume that an address is available as an n-bit word $x = (x_1, \ldots, x_n)$, each bit x_i encoded as a pair of face-down cards in positions $X_i = (X_i[0], X_i[1])$ and that the word $\text{RAM}[x]$ should be loaded into the accumulator. We give an implementation as Protocol 11.3. The first loop uses n conditional swaps of RAM ranges to transport the content of $\text{RAM}[x]$ into $\text{RAM}[0]$. The invariant is that after the i-th loop, the content of $\text{RAM}[x]$ has been transported to $\text{RAM}[x]\&(2^{n-i} - 1)$ where & denotes the bitwise AND. For instance, if $n = 4$ and $x = 10 = (1010)_2$, then in the rounds $i = 1$ the left half $\text{RAM}[0 \ldots 7]$ and right half $\text{RAM}[8 \ldots 16]$ of the memory would be swapped and in round $i = 3$ the ranges $\text{RAM}[0, 1]$ and $\text{RAM}[2, 3]$ would be swapped, in total transporting $\text{RAM}[10]$ via $\text{RAM}[2]$ to $\text{RAM}[0]$.

The second for-loop copies the content of $\text{RAM}[0]$ to the accumulator. Since the copy protocol can copy information only onto card pairs that are in a known state, we must securely reset the accumulator bits before each copy operation. The third for-loop undoes all swaps of the first loop, in reverse order.

Storing a Word. Storing is very similar to loading, we give an implementation in Protocol 11.4. Here, instead of copying the RAM content to the accumulator in the second line of the second for loop, we copy the value of the accumulator into the RAM.

Move Operations. The operations previously dubbed **copy**, **indirect read** and **indirect write** are easy to implement using the load and store algorithms. For temporary storage, the accumulator A' is used. For instance, the indirect store operation $\text{RAM}[\text{RAM}[x]] \leftarrow \text{RAM}[y]$ with the words x and y encoded in positions

for $i = 1$ **to** n **do**
  $\mathsf{sort}^* X_i \uparrow (\textsc{ram}[0 \dots 2^{n-i}-1], \textsc{ram}[2^{n-i} \dots 2^{n-i+1}-1])$
for $i = 1$ **to** n **do**
  $\mathsf{sort}\, A_i$ `// securely reset` i`-th bit of accumulator`
  $\mathsf{sort}^* \textsc{ram}[0]_i \uparrow A_i$ `// copy` i`-th bit`
for $i = n$ **down to** 1 **do**
  $\mathsf{sort}^* X_i \uparrow (\textsc{ram}[0 \dots 2^{n-i}-1], \textsc{ram}[2^{n-i} \dots 2^{n-i+1}-1])$

Protocol 11.3. $\mathsf{load}(X)$, where $X = (X_1, \dots, X_n)$ is a sequence of n card-pairs encoding an n-bit address $x = (x_1, \dots, x_n)$.

for $i = 1$ **to** n **do**
  $\mathsf{sort}^* X_i \uparrow (\textsc{ram}[0 \dots 2^{n-i}-1], \textsc{ram}[2^{n-i} \dots 2^{n-i+1}-1])$
for $i = 1$ **to** n **do**
  $\mathsf{sort}\, \textsc{ram}[0]_i$ `// securely destroy content`
  $\mathsf{sort}^* A_i \uparrow \textsc{ram}[0]_i$ `// copy` i`-th bit of accumulator`
for $i = n$ **down to** 1 **do**
  $\mathsf{sort}^* X_i \uparrow (\textsc{ram}[0 \dots 2^{n-i}-1], \textsc{ram}[2^{n-i} \dots 2^{n-i+1}-1])$

Protocol 11.4. $\mathsf{store}(X)$, where $X = (X_1, \dots, X_n)$ is a sequence of n card-pairs encoding an n-bit address $x = (x_1, \dots, x_n)$.

X and Y can be implemented using $\mathsf{load}(Y)$, $\mathsf{swap}(A, A')$, $\mathsf{load}(X)$, $\mathsf{swap}(A, A')$, $\mathsf{store}(A')$, where swap just swaps the two card sequences.

Loading Constants. Copying a value given directly in the instruction is simply done by copying each of the n bits one by one.

Addition and Subtraction. Secure half and full adders have been described by [MAS13]. If $n \geq 2$, the accumulator A' is sufficient to store carry-bits temporarily. We omit the details.

Conditional Jump. While it would be possible to have an instruction pointer that is affected by jump operations, we opt for an approach that seems slightly more elegant. We always execute instruction group $\textsc{i}[0]$, and when executing the last instruction $\mathsf{jnz}\, \textsc{ram}[x]\ p$ of that group, we rotate the sequence of all instructions such that *either* $\textsc{ip}[1]$ *or* $\textsc{ip}[p]$ becomes $\textsc{ip}[0]$, depending on the value of $\textsc{ram}[a]$. See below for the exact description.

The overall execution of the RAM program is given in Protocol 11.5. We assume the addresses x and p are available in positions X and P, respectively. To carry out the conditional jump, first load x into the accumulator and form the Boolean OR of all its bits. Assuming $\textsc{ram}[0]$ is not zero, then the bit a_1 is set to true by this OR operation and the single ♡-card is swapped into $\textsc{ip}^*$ before the for-loop and is put into position

IP[1] afterwards. If, however, RAM[0] is zero, then a_1 is set to false in which case the for-loop transports the ♣-card into position IP[p] (the loop invariant is that the ♣-card is in position IP[$p \,\&(2^{m-i}-1)$]). The rot operation in the last step rotates the sequence of instructions as desired.

repeat t **times**
  ⟨execute all instructions in group I[0], except the jump⟩
  `// Now execute jnz` RAM[x] p`:`
  load(X)
  $A_1 \leftarrow A_1$ OR A_2 OR ... OR A_n
  $A_1 \leftarrow \neg A_1$ `// swap` A_1`'s cards`
  sort* $A_1 \uparrow$ (IP[0], IP*)
  for $i = m$ **down to** 1 **do**
    sort* $P_i \uparrow$ (IP[$0 \dots 2^{m-i}-1$], IP[$2^{m-i} \dots 2^{m-i+1}-1$])
  sort $A_1 \uparrow$ (IP*, IP[1])
  rot IP[$0 \dots M-1$] $\uparrow$ I[$0 \dots M-1$]
result RAM `// or parts of it`

Protocol 11.5. executeRAM()

Theorem 11.3. *For any $N = 2^n, M = 2^m, k < N, t \geq 0$ there exists a secure card-based protocol $\mathcal{P}$ with the following properties:*

1. *The input sequences are all sequences (V, P) where*
   - *V encodes k n-bit words $(v_1, \dots, v_k) \in \{0,1\}^{nk}$ using the deck $nk \cdot [\clubsuit, \heartsuit]$.*
   - *P encodes an n-bit-word RAM machine R with M instruction groups using the deck $kM \cdot [\clubsuit, \heartsuit]$, where $k = O(n+m)$ is the length of the encoding of one instruction group.*
2. *The output is a sequence of cards encoding the output of R on input $(v_1, \dots, v_k)$ after t steps.*
3. *In addition to the cards encoding the inputs, we need the helping deck $(N-k+2)n \cdot [\clubsuit, \heartsuit] \cup [\clubsuit, M \cdot \heartsuit]$. (Additional cards for the starred sort variants can borrow from A'.)*

Proof. For the correctness, we refer to the above explanation of all the relevant commands. For security we again use Corollary 8.1 and the fact that we do not turn over any cards outside sort or rot operations. For this, note that the OR operation in line 5 of Protocol 11.5 can be framed as a sort operation, cf. Protocol 8.10. □

Remark 11.4 (Reusability of the Program)**.** Similarly to Remark 11.3 for the TM case, we can ensure that we end in the original configuration (with the first instruction in IP[0]) by introducing an additional register START[$0 \dots M-1$] which is rotated together

with the instruction groups and IP. At the end of the execution we use it to rotate everything back into place and additionally reset the accumulators.

11.4. Securely Evaluating a Branching Program

Branching Programs [B89] are commonly used for constructing program obfuscation, e.g., in [GGH+13; GKW17; WZ17], which inspired this section.

Branching Programs

A *branching program* B of length N and width w for k variables is a sequence

$$((j^{(i)}, \pi_0^{(i)}, \pi_1^{(i)}))_{1 \le i \le N} \in (\{1, \dots, k\} \times S_w \times S_w)^N$$

of *instructions*. The permutation belonging to a sequence $\vec{v} = (v_1, \dots, v_k) \in \{0,1\}^k$ of inputs is

$$B(\vec{v}) = \prod_{1 \le i \le N} \pi^{(i)}_{v_{j^{(i)}}}.$$

In other words, in the i-th step, the value of the $j^{(i)}$-th variable determines which of the two permutations of the i-th instruction is used.

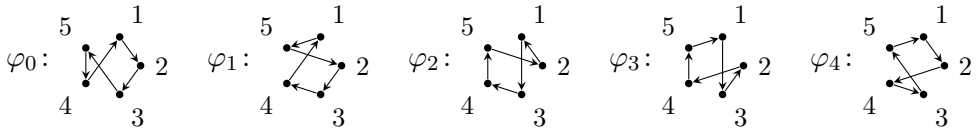
For $\sigma \in S_w$ we say B *σ-computes a Boolean circuit* C, if for any $\vec{v} \in \{0,1\}^k$

$$B(\vec{v}) = \begin{cases} \sigma, & \text{if } C(\vec{v}) = 1, \\ \mathsf{id}, & \text{if } C(\vec{v}) = 0. \end{cases}$$

Now let State be a set of states on which S_w acts via some group action $*$ and executing B on $\vec{v}$ starting from some start state $q_0 \in \mathsf{State}$ means computing states $(q_i)_{1 \le i \le N}$ iteratively as $q_{i+1} = \pi_{v_{j^{(i)}}} * q_i$. Of course, we end with $q_N = \pi_{v_{j^{(N)}}} * \cdots * \pi_{v_{j^{(1)}}} * q_0 = B(\vec{v}) * q_0$. In this document, State is a set of card sequences of length w and $\pi * q$ yields the card sequence q permuted by π.

A Peculiar Subset of S_5

Barrington's Theorem makes heavy use of the fact that S_5 is not a solvable group. In particular, there are permutations $\pi, \tau \in S_5$ such that the commutator $[\pi, \tau] := \pi \circ \tau \circ \pi^{-1} \circ \tau^{-1}$ is not the identity permutation. There is some freedom when choosing permutations for the construction that follows. To be more specific, we define the five permutations $\varphi_0, \dots, \varphi_4$ as



In general, we can define $\varphi_i = (1\ 2\ 3\ 4\ 5)^i \circ \varphi_0 \circ (1\ 2\ 3\ 4\ 5)^{-i}$ for any $i \in \mathbb{Z}$ but, of course, only the remainder of the index modulo 5 is relevant.

It is easy to check that $\varphi_0 = \varphi_5 = [\varphi_3, \varphi_4]$ and $\varphi_0^{-1} = \varphi_5^{-1} = [\varphi_1, \varphi_3]$. We can therefore write each element $\varphi \in F := \{\varphi_0, \ldots, \varphi_4, \varphi_0^{-1}, \ldots, \varphi_4^{-1}\}$ as $\varphi = [\varphi', \varphi'']$ for some other elements $\varphi', \varphi'' \in F$. More concretely, we have

$$\varphi_i = [\varphi_{i+3}, \varphi_{i+4}], \qquad \varphi_i^{-1} = [\varphi_{i+1}, \varphi_{i+3}].$$

Barrington's Theorem

We now state a central theorem due to Barrington, which we specialize to permutations from the set F defined above. For self-containedness and illustration, we give the elegant and constructive proof in full. Recall from Chapter 11 that the depth of a circuit C is the maximum number of $\wedge$ and $\vee$ gates on a path in C.

Theorem 11.4 (Barrington [B89])**.** *For any Boolean circuit C of depth d and $\varphi \in F$ there exists a branching program $B = B(C)$ of width 5 and $N \leq 4^d$ instructions that φ-computes C.*

Proof. The proof works by induction on the length d' of the longest path in C. If $d' = 0$, then we also have $d = 0$ and the output node is labeled with a constant 0, a constant 1 or the index j of a variable. In these cases, the trivial branching programs with a single instruction of the form $(\_, \mathsf{id}, \mathsf{id})$, $(\_, \varphi, \varphi)$ or $(j, \mathsf{id}, \varphi)$, respectively, φ-compute C (here, _ is a placeholder for an arbitrary variable index).

Now assume $d' > 0$. If the output node is labeled „¬“, then the value at its unique predecessor is computed by a circuit C' with longest path of length $d' - 1$. Therefore, there is a branching program B' that φ^{-1}-computes C' with at most 4^d instructions. Let (j, π, π') be the last instruction of B'. Replacing it with $(j, \varphi \circ \pi, \varphi \circ \pi')$ yields a branching program B that φ-computes C since we have

$$B(\vec{v}) = \varphi \Leftrightarrow B'(\vec{v}) = \mathsf{id} \Leftrightarrow C'(\vec{v}) = 0 \Leftrightarrow C(\vec{v}) = 1$$

and for similar reasons $B(\vec{v}) = \mathsf{id} \Leftrightarrow C(\vec{v}) = 0$.

If the output node is labeled $\wedge$, then values at its two predecessors are computed by two circuits C' and C'' with longest path of length at most $d' - 1$ and depth at most $d - 1$. We previously observed that we can write $\varphi = [\varphi', \varphi'']$ for two permutations $\varphi', \varphi'' \in F$. Let $B'_{\varphi'}$ and $B'_{\varphi'^{-1}}$ be two branching programs that φ'-compute and φ'^{-1}-compute C', respectively, and similarly $B''_{\varphi''}$ and $B''_{\varphi''^{-1}}$ be two branching programs that φ'-compute and φ''^{-1}-compute C'', respectively.

We obtain B as the concatenation of these four branching programs. Depending on the values $r' = C'(v_1, \ldots, v_k)$ and $r'' = C''(v_1, \ldots, v_k)$ we get the following behavior of B:

$$B(\vec{v}) = B'_{\varphi'}(\vec{v}) \circ B''_{\varphi''}(\vec{v}) \circ B'_{\varphi'^{-1}}(\vec{v}) \circ B''_{\varphi''^{-1}}(\vec{v})$$
$$= \begin{cases} \varphi' \circ \varphi'' \circ \varphi'^{-1} \circ \varphi''^{-1} = [\varphi', \varphi''] & = \varphi \text{ if } r' = r'' = 1 \\ \mathsf{id} \circ \varphi'' \circ \mathsf{id} \circ \varphi''^{-1} & = \mathsf{id} \text{ if } r' = 0,\ r'' = 1 \\ \varphi' \circ \mathsf{id} \circ \varphi'^{-1} \circ \mathsf{id} & = \mathsf{id} \text{ if } r' = 1,\ r'' = 0 \\ \mathsf{id} \circ \mathsf{id} \circ \mathsf{id} \circ \mathsf{id} & = \mathsf{id} \text{ if } r' = r'' = 0 \end{cases}$$

Since $C(\vec{v}) = 1 \Leftrightarrow r' = r'' = 1$, this means B indeed φ-computes C. □

Implementing Branching Programs with Cards

We first describe how the encoding $P = P(C)$ is obtained from C, as the format of P already contributes to hiding details about C, especially the pattern in which variables are used. Firstly, by Barrington's Theorem (Theorem 11.4) there is a branching program $B = B(C)$ that φ_0^{-1}-computes C with $N \leq 4^d$ instructions. We now transform B into a *normalized branching program* B' by preceding each instruction (j, π_0, π_1) of B with the $j-1$ dummy instructions $(1, \mathsf{id}, \mathsf{id}), \ldots, (j-1, \mathsf{id}, \mathsf{id})$ and appending to it the $k-j$ dummy instructions $(j+1, \mathsf{id}, \mathsf{id}), \ldots, (k, \mathsf{id}, \mathsf{id})$. This means that B' accesses all variables periodically in canonical order. Note that B' contains $kN \leq k \cdot 4^d$. (In addition, we may choose to pad B' to a longer program B'' of length kN' if we wish to hide the length of B' and thus of B.) Clearly, B' exhibits the same behavior as B. The sequence P is now simply obtained by concatenating the kN sequences encoding the permutations occurring in the description of B'.

Theorem 11.5. *For any $k, N \geq 0$ there exists a secure card-based protocol $\mathcal{P}$ with the following properties:*

1. *The input sequences are all sequences (V, P) where*
   - *V encodes the values of k Boolean variables $(v_1, \ldots, v_k) \in \{0,1\}^k$ using the deck $k \cdot [\clubsuit, \heartsuit]$.*
   - *P encodes a normalized branching program B of length kN with one bit output using the deck $2kN \cdot [1, 2, 3, 4, 5]$.*
2. *The output is two cards encoding $B(v_1, \ldots, v_k)$.*
3. *In addition to the cards encoding the inputs, the helping deck $[2 \cdot \heartsuit, 5 \cdot \clubsuit]$ is used. Each execution of the protocol performs $2kN$ shuffle actions.*

Proof. The protocol is described in Protocol 11.6. We denote by capital letters the sets of positions on which the corresponding parts of the input (denoted by lower case letters) are present at the start of the protocol. Additionally, there are helping cards

for $i \leftarrow 0$ **to** $N-1$ **do**
  for $j \leftarrow 1$ **to** k **do**
    $\mathsf{sort}^*\, V_j \uparrow (\Pi_0^{(ik+j)}, \Pi_1^{(ik+j)})$
    $\mathsf{sort}\, \Pi_0^{(ik+j)} \uparrow \mathrm{Q}$
result Q_R

Protocol 11.6. Executing a branching program.

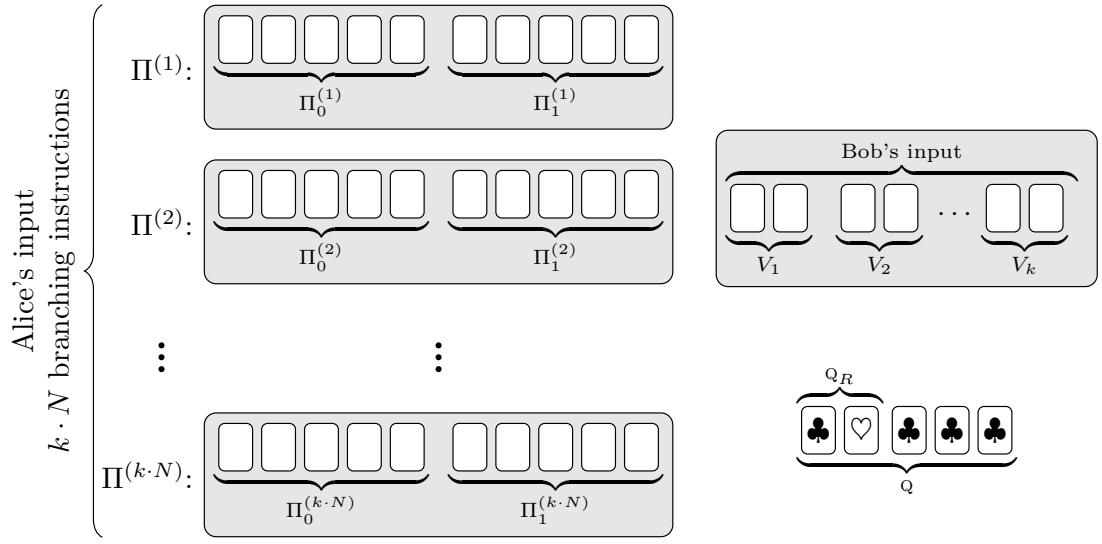


Figure 11.3.: Overview of the branching program construction. Alice's input is the branching program $((j^{(i)}, \pi_0^{(i)}, \pi_1^{(i)}))_{1 \le i \le N} \in (\{1, \ldots, k\} \times S_5 \times S_5)^N$ in normalized form.

present in positions Q that initially contain the sequences ♣♡♣♣♣ as well as two cards to support the sort^*-operation (not shown in Figure 11.3).

Consider an iteration of the inner loop with $r = ki + j$. First, the encodings of the two permutations $\pi_0^{(r)}$ and $\pi_1^{(r)}$ (in positions $\Pi_0^{(r)}$ and $\Pi_1^{(r)}$) are swapped if v_j (in position V_j) is 1 and left as is otherwise. Hence, an encoding of $\pi_{v_j}^{(r)}$ ends up in position $\Pi_0^{(r)}$, from where it is obliviously applied to the sequence in Q. For correctness, note that by assumption the normalized branching program φ_0^{-1}-computes C, i.e., if the output is 0, in total we perform id on the cards in Q, which results in a 0 being encoded in Q_R. If C outputs 1, then φ_0^{-1} is applied to the cards of Q, resulting in ♡♣♣♣♣, as φ_0^{-1} maps $2 \mapsto 1$, yielding an encoded 1 in Q_R.

Security of $\mathcal{P}$ follows again from the fact that the protocol is only composed of valid sort operations and Corollary 8.1. □

Remark 11.5 (Reusability of the Program)**.** To allow for reusing the branching program after its execution, we would need to write the executed permutation of each step back

into its register and to undo any conditional swaps. In more formal terms, we replace the sort command in the second line of the inner loop of Protocol 11.6 with its starred variant. To undo the swap, we repeat the first line of the inner loop after the second line.

A Note Regarding Active Security

Note that a malicious Alice might learn something about the input passed to the program by choosing the permutations of the program in such a way that the output (the first two cards in Q after the protocol run) is not ♣♡ or ♡♣, but ♣♣. If we want to avoid this, we can initialize Q with ♣♡♣♡♣ (replacing the penultimate ♣ with a ♡), and instead of opening just the first two cards at the end, we have to ensure that the content of the register gets mapped to a single bit, without revealing anything else. For this, note that after a protocol run of a legal program, Q contains one of two configurations namely ♣♡♣♡♣ if id was applied, and ♡♣♣♣♡ if φ_0^{-1} was applied. Important here, is that in the first case, the ♡s have distance 1 and in the second case distance 0, which is invariant over random cuts, and represents the two possible configuration classes (orbits w.r.t. random cuts) in the five-card trick [dB90]. We cannot use the five-card trick directly, as its output is not in committed format, however. To overcome this, we can make use of the recent five-card AND protocol of [AHMS18] (cf. Figure 7.14), which starts with a situation as above and then outputs a bit commitment to the AND value in a (restart-free) Las Vegas fashion.

Moreover, for active security in all the protocols in this chapter, one should additionally implement the shuffle operation with active security as in Chapter 12. For ease of implementing the coupled shuffles, we recommend to use envelopes to avoid additional helping cards, as in Figure 8.8.

11.5. Conclusion

We give four card-efficient and conceptually simple protocols for executing a universal machine model in a secure multiparty computation protocol, hence achieving Private Function Evaluation. These are for circuits, Turing and word-RAM machines and branching programs, giving the user a palette of options, from which they can choose the most suitable one. We give the concrete numbers of necessary cards for each of the models, carefully reusing helping cards where possible. We additionally discuss several adaptions, e.g., on how to execute these in a non-destructive way that lets us reuse the program multiple times.

Our results can also be interpreted as a straightforward instantiation of Oblivious RAM (ORAM), making heavy use of the fact that we can physically and obliviously move around "RAM cells", which is not possible in the usual cryptographic ORAM model. By stating these classical cryptography problems, such as constructing ORAM or program obfuscation in the language of card-based cryptography, it might not only be of didactic use in explaining these to students, but also provide some insight into the constructions in the classical cryptographic realm.

12. Active Security for Card-based Protocols

This chapter is mainly based on [KW17], with some adaptations to the common notation and a new part on input-aware protocols and their security.

In the formal computational model of Mizuki and Shizuya [MS14a], a protocol specification may use any of the most general shuffling operations, namely applying a permutation from an *arbitrary permutation set* chosen according to an arbitrary distribution. This computational model is very useful when showing impossibility results and *lower bounds* on cards, cf. Chapter 9, 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, cf. Section 8.3.

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 work with "honest-but-curious" assumptions, with no guarantees when one of the players deviates from the protocol.

Related Work. Other works have investigated the question of active attacks, albeit with a different focus. Mizuki and Shizuya [MS14b] address active security against adversaries who deviate from the input encoding, e.g., giving input $(\heartsuit, \heartsuit)$ instead of $(\heartsuit, \clubsuit)$. We sketch in Section 12.9 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. Shinagawa et al. [SMS$^+$15b] describe a method against injection attacks in their model using polarizing plates. Recently and independently, Ueda et al. [UNH$^+$16] give an elaborate implementation of the special case of random bisection cuts, including experiments showing the real-world security of the shuffle. Moreover, Shinagawa et al. [SMS$^+$15a] give a security notion that takes into account a number of players.

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 cases where this is non-trivial.

Our Contribution. At several places in the literature the open question of achieving actively secure shuffles and protocols is posed. In this chapter, we answer a significant part of this question by explaining how any protocol in the model of [MS14a] that is

restricted to *uniform closed shuffles* can be transformed into an actively secure protocol using only a linear number of helping cards. Uniform closed shuffles, namely those that rearrange the cards according to a *uniform* distribution on a permutation *group*, have already been identified in Section 6.7 as a natural class of operations.

Furthermore, 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 three-card AND protocol by Karun Singh as described in [MWS15, Sect. 3.2] 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 [KWH15, Sect. 8]. Independently from our work, these operations are used in [NTM+16] to more efficiently perform an instance of the millionaires' problem with cards and in [NSIO17] for the case of a three-input voting protocol. The way they are used there causes the protocol to require input awareness for the players, which is usually not required by, e.g., a committed format protocol. For a correct analysis however, one has to introduce two operations, namely privatePerm and inputPerm, with the first allowing to do a permutation privately, without anyone looking, and the second additionally marks this permutation as an *input*, which is to be protected by corresponding notions of security.

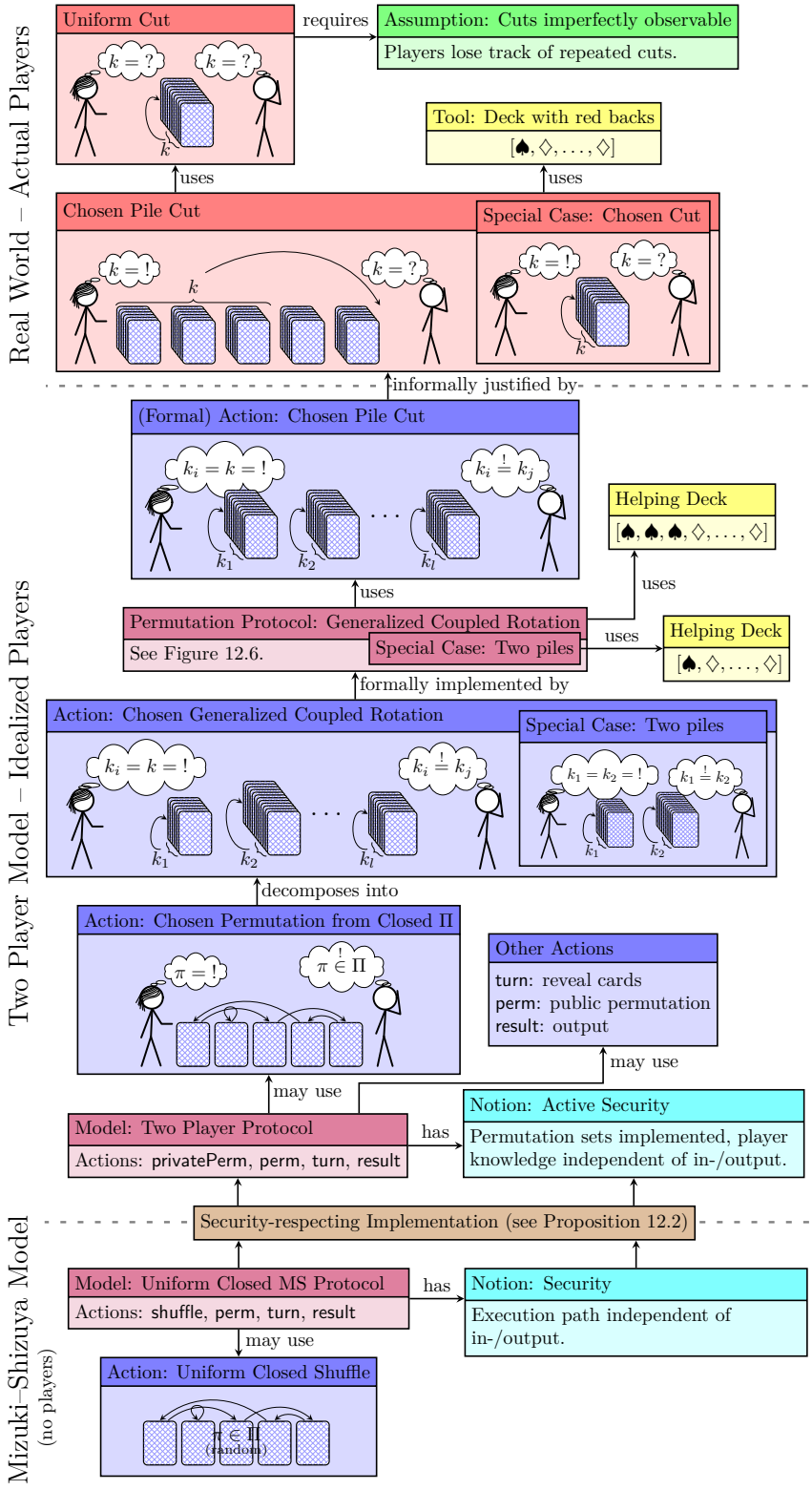
12.1. Implementing Cuts and Pile Cuts with Choice

A set $\Pi \subseteq S_n$ of permutations has an *actively secure implementation with choice*, or *is implemented* for short, if there is a procedure that allows 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.

Example: Bisection Cut with Envelopes. Mizuki and Sone [MS09] 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 role of the envelopes is to ensure that the two groups of cards stay together and the ordering within a group 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. [MN06a; MN10a].

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



A *uniform cut* (p. 174) rotates a pile of cards by a uniformly random value unknown to Alice and Bob. From this we build *chosen cuts* (p. 174) leaving a pile rotated by a value chosen by Alice but unknown to Bob. When generalized to *chosen pile cuts* (p. 175) 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. 176) this is extended to the case where piles may have different sizes. This yields *chosen coupled rotations* (p. 177) in the case of two piles and *chosen generalized coupled rotations* (p. 177) in the case of more than two piles.

These are powerful enough to build arbitrary *chosen permutations from a closed permutation set* (p. 180). 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. 181) 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. 45) 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. 184) if the original protocol is *secure* (p. 182). This *security-respecting implementation* (p. 185) 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.

Figure 12.1.: Overview of the content of this chapter. The images of Alice and Bob are adapted from xkcd (by Randall Munroe), which is licensed as CC-BY-NC-2.5.

want to "either do nothing or rotate the sequence by l positions" for some $0 < l < n$, i.e., implement $\Pi_l = \{\mathsf{id}, (1\ 2\ \ldots\ n)^l\}$. Nishimura et al. [NNH+15; NNH+18] describe a corresponding mechanism 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 Section 12.8).

While we are very fond of creative ideas such as these, we shall make it our mission to implement card based protocols using only one tool: additional cards.

Cutting the Cards

By the *cut on n cards* we mean the permutation set $\Pi = \langle(1\ \ldots\ n)\rangle$ and, ordinarily, Alice would *cut* a pile of n cards by taking the top-most k cards (for some $0 \leq k < n$) from the top of the pile and 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 [dB90], 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 the uniform stable distribution, yielding 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 [UNH+16] found it to work well in practice even for $n = 2$ and $c = 3$.[1] 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 $n \cdot (m + 1)$ cards with n cards of red back, each preceded by m cards of blue back. From a "uniform cut" on such a pile, we expect a cut by $0 \leq k < n \cdot (m + 1)$ where $\lfloor k/(m + 1) \rfloor$ is uniformly distributed in $\{0, \ldots, n - 1\}$ and independent of the observable part $k \pmod{m + 1}$. 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 12.2.

[1] If not satisfied, the reader may be more inclined to accept some variant of Berry's turntable as described by Verhoeff [V14, Sect. 3], cf. also our discussion on page 37 of Chapter 5. There, cards are attached to a wheel-of-fortune-esque device.

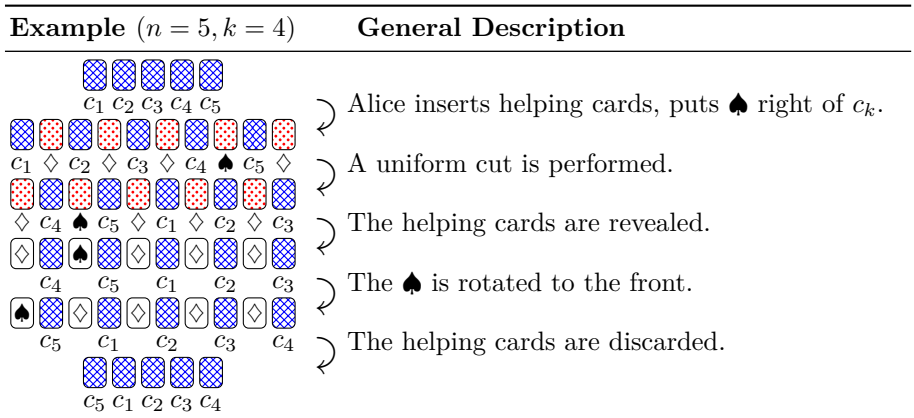


Figure 12.2.: Alice cuts a pile of n cards, here $(c_1, \ldots, c_5)$, with back at position k with a helping deck of n helping cards $[\spadesuit, 4 \cdot \diamondsuit]$ with back . In this illustration we annotated face-down cards with the symbol they contain.

Figure 12.3.: 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)*.

Alice is handed the helping deck $[\spadesuit, (n-1) \cdot \diamondsuit]$ with red backs and secretly rearranges these cards in her hand, putting ♠ 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 ♠ 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 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 m 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 nm cards where only cutting by multiples of m is allowed, see Figure 12.3. In that interpretation, the i-th pile is made up of the cards in positions $(i-1)m + 1, \ldots, im$.

We apply the same procedure as before with n helping cards, except this time, instead of a single blue card we have m blue cards (a pile) before each of the n gaps that Alice may fill with her red deck $[\spadesuit, (n-1) \cdot \diamondsuit]$. Now the special $\spadesuit$-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 $n \cdot m$ cards by a multiple of m as desired. Note that, uniform (non-chosen) pile cuts have been proposed in [ICM15] as "pile-scramble shuffles", with an implementation using rubber bands, clips or envelopes.

Summary. If $\Pi = \langle (1\ 2\ \ldots\ n \cdot m)^m \rangle$ for $n, m \in \mathbb{N}$, then Π is implemented with active security and choice using the helping deck $[\spadesuit, (n-1) \cdot \diamondsuit]$. For $m = 1$ it is called a *cut*, for $m > 1$ a *pile cut*. We use the same name for conjugates of Π, i.e., if card positions 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 $\diamondsuit$.

12.2. Permutation Protocols for Arbitrary Groups

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

For notation, when applying a permutation π of X to a set $S \subseteq X$ we write $\pi(S) := \{\pi(s) \colon s \in S\}$. 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 $s \in S$.

Definition 12.1. 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 $\Gamma \colon \{n+1, \ldots, n+|\mathcal{H}|\} \to \mathcal{H}$, 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 $\{1, \ldots, n\}$ (i.e., $\forall \pi \in \Pi \colon \pi(\{1, \ldots, n\}) = \{1, \ldots, n\}$), 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 $1, \ldots, n$). We place the sequence Γ next to it, at positions $n+1, \ldots, n+|\mathcal{H}|$, 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 $\{1, \ldots, n\}$, 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 $\Pi_1, \ldots, \Pi_k$ 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 $\Pi' = \mathsf{comp}(\mathcal{P})|_{\{1,\ldots,n\}}$ using $\mathcal{H}$ (and, possibly, helping cards to implement Π_i).

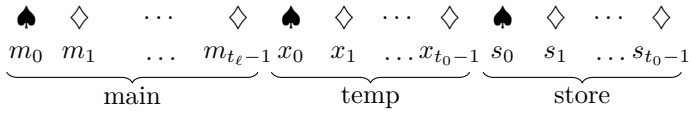
Alice can freely pick any $\pi' \in \Pi'$; using an appropriate decomposition, all checks will succeed. In this case, Bob knows that the performed permutation is from Π'. No player learns anything about the object cards (only helping cards are turned) and conditioned on Alice being honest, the outcome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

Coupled Rotations. Let $\varphi = (1\ 2\ \ldots\ s)$, $\psi = (s{+}1\ s{+}2\ \ldots\ s{+}t)$, and assume $s < t$. For $\pi = \psi \circ \varphi = \varphi \circ \psi$ we call $\Pi = \{\pi^k \colon 0 \le k < s\}$ 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 $[\spadesuit, (t{-}1) \cdot \diamondsuit]$ available in positions $H = \{h_0, h_1, \ldots, h_{t-1}\}$ with $\spadesuit$ at position h_0. Then define $\hat{\varphi} := \varphi \circ (h_0\ \ldots\ h_{s-1})$ and $\hat{\psi} := \psi \circ (h_0\ \ldots\ h_{t-1})^{-1}$ and consider the permutation protocol $\mathcal{P}$ in Figure 12.5 (left), and Figure 12.4 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 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

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

Products, Conjugates and Syntactic Sugar. The protocol in Figure 12.5 (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 12.5 (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 $\pi = \varphi_0 \circ \cdots \circ \varphi_m$ for $m \ge 2$ and increasingly ordered cycle lengths $t_0 \le t_1 \le t_2 \le \ldots \le t_m$. We aim to implement $\Pi = \{\pi^k \colon 0 \le k < t_0\}$ using $t_m + 2 \cdot t_0$ helping cards, originally available in the following positions which we label as shown.



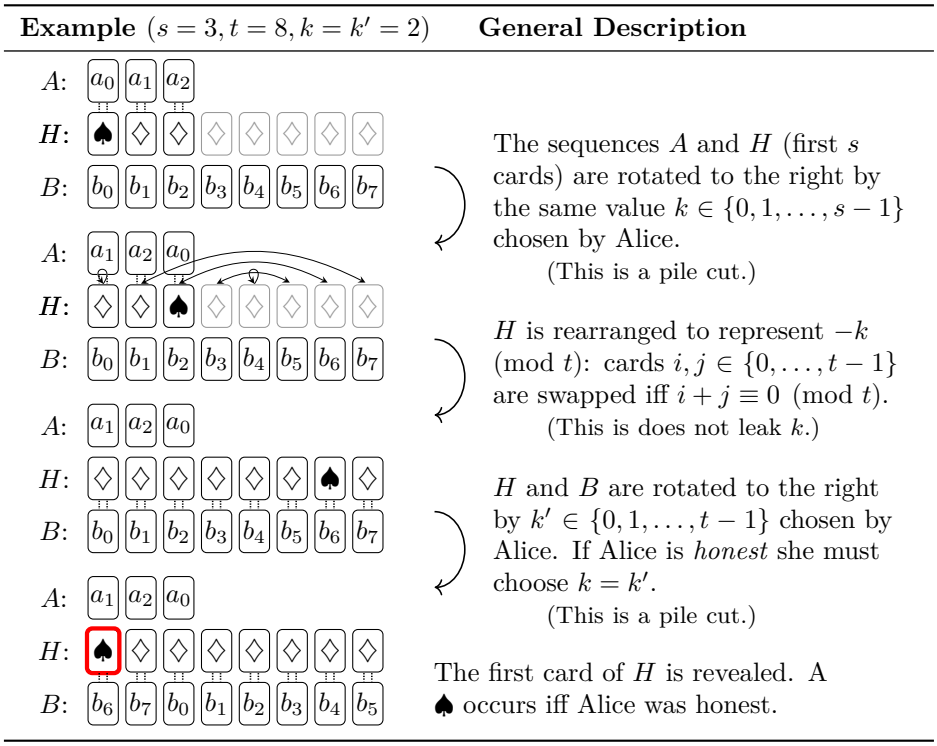


Figure 12.4.: The sequence A of length s and B of length t are to be rotated by the same value k chosen privately by Alice. A helping sequence ensures that the same value is used. All cards are face-down, except for the highlighted card in the last step. The dotted lines indicate that cards are belonging 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.

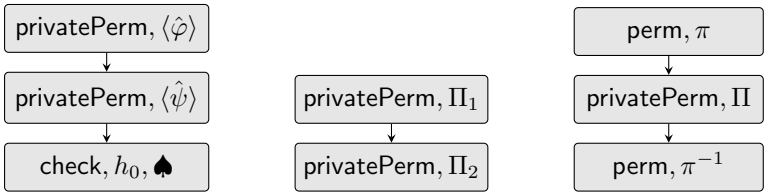


Figure 12.5.: Protocols implementing a coupled rotation (*left*), the product of two permutation sets (*middle*) and the conjugation of a permutation set (*right*).

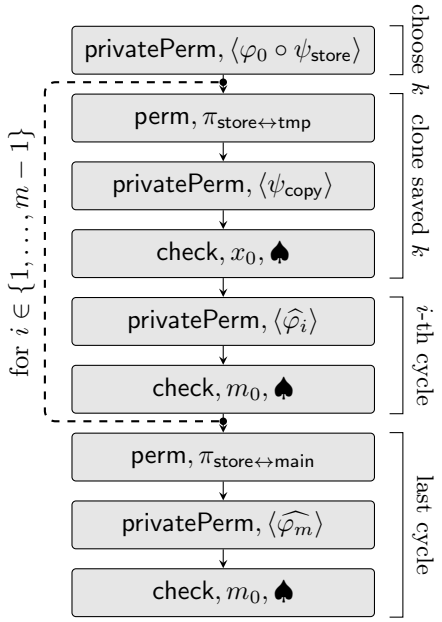


Figure 12.6.: Protocol to implement a generalized coupled rotation with $m+1$ cycles of length $t_0, t_1, \ldots, t_m$. Notation is explained in the text.

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 $\varphi_0^{k_0} \circ \cdots \circ \varphi_m^{k_m}$ and checks will ensure $k_1 = k_2 = \ldots = k_m$. 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 $i \in \{1, 2, \ldots, m-1\}$ 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 12.6.

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

We now check that this implements the generalized coupled rotation Π using the helping cards $[3 \cdot ♠, (t_m + 2t_0 - 3) \cdot ♢]$. The main ingredient is the loop invariant:

> *If $\pi \in S_{n+2t_0+t_m}$ 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.*

This invariant can be proved 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, $\pi|_{\{1,\dots,n\}} = \varphi_0^k$ 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 m-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 $\pi|_{\{1,\dots,n\}} = \varphi_m^k \circ \ldots \circ \varphi_0^k$ for some $0 \leq k < t_0$.

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

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 $\langle \pi \rangle = \{\pi^0, \dots, \pi^{k-1}\}^m$, where k is the length of the shortest cycle in the cycle decomposition of π and $m = \lceil \mathsf{ord}(\pi)/(k-1) \rceil$. 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 177), we are done. □

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

12.3. Computational Model with Two Players

In the following, two players jointly manipulate a sequence of cards to compute a 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.[2]

Two Player Protocols. A *two player protocol* is a tuple $(\mathcal{D}, U, H, Q, A)$ where, as in Mizuki–Shizuya protocols, $\mathcal{D}$ is a deck, U is a set of input sequences, H a card sequence with helping cards, Q is a set of states, and A an action function that however, maps to $\mathsf{PlayerAction}_\ell$, which can be perm, turn, result, and privatePerm, with parameters as explained below. All input sequences have the same length and are formed by cards from $\mathcal{D}$, with the helping sequence H concatenated at the beginning of the protocol. The total length of the card sequence is then denoted by ℓ.

When a protocol is *executed on an input sequence* $I \in U$, we start with the face-down sequence $\Gamma = I$ in initial state $q_0 \in Q$ and empty *permutation traces* $\mathcal{T}_1$ and $\mathcal{T}_2$ for players 1 and 2, respectively. Execution proceeds according to A, with one of the actions as in Section 6.3, except that the shuffle action is replaced as follows:

- $(\mathsf{privatePerm}, p, \Pi, \mathcal{F}(\cdot))$, for a player $p \in \{1, 2\}$, a permutation set $\Pi \subseteq S_\ell$ 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 $\mathcal{F}(\mathcal{T}_p)$.

The execution yields an *execution trace* $(I, O, \mathcal{T}_1, \mathcal{T}_2, V)$, containing input, output, permutation traces of the players and the visible sequence, see Figure 12.7 for an example. The output of non-terminating protocols is $O = \bot$.

Note that we will use permutation protocols from Section 12.2 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 occurring in a privatePerm action is implemented using $\mathcal{H}$ (in the sense of Section 12.1).

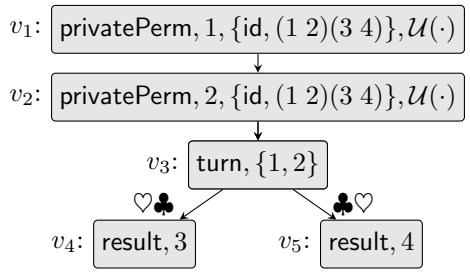
The way we define it, existence, implementability and security of a protocol are separate issues. Security is discussed next.

12.4. 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 V in which the sequence of encountered actions and their parameters are implicit. For each such action she may have obtained

[2] An explanation of our security notions follows in Section 12.4.



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

$$U = \{(\clubsuit, \heartsuit, \clubsuit, \clubsuit), (\clubsuit, \heartsuit, \heartsuit, \clubsuit), (\heartsuit, \clubsuit, \clubsuit, \clubsuit), (\heartsuit, \clubsuit, \heartsuit, \clubsuit)\},$$

Q, A: as shown on the left

Figure 12.7.: A protocol example in the two player model, with possible execution trace: $(I = (\heartsuit, \clubsuit, \heartsuit, \clubsuit), O = (\heartsuit), \mathcal{T}_1 = (\mathsf{id}), \mathcal{T}_2 = ((1\ 2)(3\ 4)), V = (v_1, v_2, v_3, v_5))$. This is an actively secure implementation of the AND protocol in [M16a, Sect. 3.2]. (Here, we use an ad-hoc "instruction tree diagram" to hide details about the state.) The first two cards encode an input a as $(\clubsuit, \heartsuit) \mathrel{\widehat{=}} 0$, $(\heartsuit, \clubsuit) \mathrel{\widehat{=}} 1$, the third card encodes an input b as $\clubsuit \mathrel{\widehat{=}} 0$, $\heartsuit \mathrel{\widehat{=}} 1$. This encoding is also used for output $a \wedge b$.

additional information during its execution. To get a complete picture, we carefully go through all types of actions:

- turn actions reveal some card symbols. However, each outcome is already implicit in V.
- 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 12.1) 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 V and $\mathcal{T}_p$. Therefore it is adequate to define:

Definition 12.2 (Passive Security)**.** A two player protocol $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ is *secure against passive attackers* if for any random variable $I \in U$ the following holds: If $(I, O, \mathcal{T}_1, \mathcal{T}_2, V)$ is the execution trace when executing $\mathcal{P}$ with honest players on input I, then (I, O) is independent of $(\mathcal{T}_p, V)$ 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 can be called an *oblivious delegated computation* where the computation is performed on secret data provided by a third party, and the output is not revealed afterwards to the executers.

Note that this setting differs from the *standard MPC 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 then 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 $(\mathcal{T}_p, V)$, conditioning the probability space on E will retain the independence of (I, O) and $(\mathcal{T}_p, V)$.

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. MWS15; NTM+16; NSIO17] cannot be transfered 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, as we are in 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.[3] 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. 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 this chapter is concerned with. We may argue that the cards could be put into envelopes, and any attempt to reveal its contents

[3] 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.

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 in the following definition:

Definition 12.3. Let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a two player protocol.

1. A *permutation attack* ξ *on* $\mathcal{P}$ *as player* $p \in \{1, 2\}$ specifies for each visible sequence trace (prefix) v, upon which an action of the form $(\mathsf{privatePerm}, p, \Pi, \mathcal{F}(\cdot))$ would follow, a permutation $\xi(v) \in \Pi$. Replacing such $\mathcal{F}(\cdot)$ with the (point) distributions that always choose $\xi(v)$, yields the *attacked protocol* $\mathcal{P}^\xi$.

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

$$\Pr[V^\xi = v] > 0 \Longrightarrow \Pr[(I, O^\xi) = (i, o) \,|\, V^\xi = v] = \Pr[(I, O) = (i, o)]. \qquad (\star)$$

3. 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:

Definition 12.4. A two player protocol $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ has an *actively secure implementation* if each permutation set Π occurring in a $\mathsf{privatePerm}$ 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 $(\forall V^\xi)$, the best guess for the in- and output (distribution of (I, O^ξ), given V^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 V. For similar reasons, a permutation attacker can never cause a protocol with permutation security to run forever.[4]

- 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.

[4] Protocols that almost surely output $\perp$ are a pathological exception.

- 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 p learns nothing if player $3 - p$ 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 12.2. *Let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a passively secure two player protocol where for each action of form $(\mathsf{privatePerm}, p, \Pi, \mathcal{F}(\cdot))$ and each permutation trace $\mathcal{T}_p$ of player p, $\mathcal{F}(\mathcal{T}_p)$ has support Π[5]. If for each attack ξ the attacked protocol $\mathcal{P}^\xi$ terminates with probability 1[6], 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 $(I, O, \mathcal{T}_1, \mathcal{T}_2, V)$ and $(I, O^\xi, \mathcal{T}_1^\xi, \mathcal{T}_2^\xi, V^\xi)$ be the execution traces of $\mathcal{P}$ and $\mathcal{P}^\xi$.

Let v be any visible sequence trace with $\Pr[V^\xi = v] > 0$ and t the permutation trace that ξ prescribes for player p if he observes v (whenever $V^\xi = v$, then $\mathcal{T}_p^\xi = t$). For any i, o we have:

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

From the first to the second line, note that firstly, since v 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 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 $\mathcal{P}$. □

In $\mathcal{P}$, a permutation attacker can only choose permutations that an honest player may have chosen randomly, so non-trivial information she obtains about in- and output is also available to a passive attacker with positive probability. The protocol from Figure 12.7 has active security precisely for this reason.

[5] Otherwise, active attackers may pick $\pi \in \Pi$ which honest players never choose.
[6] this excludes a pathological case

12.5. Implementing Restricted Mizuki–Shizuya Protocols

In [MS14a], 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.[7] 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 [MS09].

For a definition of Mizuki–Shizuya protocols, see Section 6.3. Note that this model abstracts from concrete players and instead of $\mathsf{privatePerm}$ actions uses the common $(\mathsf{shuffle}, \Pi, \mathcal{F})$ operations, and instead of having separate permutation traces for the players, there is a single permutation trace $\mathcal{T}$.

Implementing Uniform Closed Mizuki–Shizuya Protocols. We are ready to state our main theorem of this chapter.

Theorem 12.1. *Let $\mathcal{P} = (\mathcal{D}, U, H, Q, A)$ be a secure Mizuki–Shizuya protocol using only uniform closed shuffles. Then there is a two player protocol $\hat{\mathcal{P}} = (\mathcal{D}, U, H \parallel \hat{H}, \hat{Q}, \hat{A})$ with actively secure implementation computing the same function as $\mathcal{P}$.*

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, which are given in $\hat{H}$. Here, n is the total length of the card sequences in $\mathcal{P}$.

We sketch the proof here and give the formal proof below. 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 $\mathsf{privatePerm}$ 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 12.2 also secure against permutation attacks. Moreover, by Proposition 12.1 all Π are implemented using the stated helping decks, so $\hat{\mathcal{P}}$ has an actively secure implementation.

Proof. As already mentioned, we obtain $\hat{\mathcal{P}}$ by replacing each shuffle action in $\mathcal{P}$ by two $\mathsf{privatePerm}$ actions. More precisely, let $\{(q_1, v_1), (q_2, v_2), \dots\}$ be the pairs consisting of $q_i \in Q$ and (non-empty) visible sequence traces v_i, such that the last visible sequence is denoted as v_i^+, and to which the action function assigns a following shuffle action. Let for each i denote the action by $A(q_i, v_i^+) = (q', (\mathsf{shuffle}, \Pi_i, \mathcal{U}_i))$, where Π_i is some group

[7] Excluding the use case of non-committed input protocols from [MWS15] and [NTM+16], where the input is provided by a choice of $\mathsf{privatePerm}$ operations by a player, requiring input awareness/knowledge.

and $\mathcal{U}_i$ the uniform distribution on Π_i. The new set $\hat{Q}$ is obtained from Q by replacing each q_i with two elements $q_i^{(1)}$ and $q_i^{(2)}$ with $\hat{A}(q_i^{(1)}, v_i^+) = (q_i^{(2)}, (\mathsf{privatePerm}, 1, \Pi_i, \mathcal{U}_i))$, and $\hat{A}(q_i^{(2)}, v_i') = (q', (\mathsf{privatePerm}, 2, \Pi_i, \mathcal{U}_i))$, where v_i' is the visible sequence after privatePerm by player 1. Everything else remains 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 $(\pi_1, \pi_2, \dots)$ of permutations with $\pi_i \in \Pi_i$. We shall imagine that $\mathcal{P}$ is executed by first choosing a permutation scheme $\mathcal{T} = (\pi_1, \pi_2, \dots)$ uniformly at random (each π_i uniformly at random from Π_i and independent of the rest)[8] and then executing the protocol as usual, except that we now use the chosen permutations, i.e., when reaching a shuffle action for pair (q_i, v_i) we are determined to use π_i.

Clearly, this does not affect the execution of $\mathcal{P}$ in the following sense. If I is a random input then the tuple $(I, O, V, \mathcal{T}^V)$ 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 $\mathcal{T}^V$ we mean the projection of the scheme $\mathcal{T}$ to those components used in the execution, i.e., those on vertices occurring in V (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 in situation $(q_i^{(1)}, v_i)$ and player 2 use τ_i when situation $(q_i^{(2)}, v_i \,\|\, v_i')$ is encountered. Then the tuple $(I, \hat{O}, \hat{V}, \mathcal{T}_1^{\hat{V}}, \mathcal{T}_2^{\hat{V}})$ 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, we conclude that $\mathcal{P}$ and $\hat{\mathcal{P}}$ compute the same function.

Passive Security. Note that V is determined by $(I, \mathcal{T})$, meaning there is a deterministic measurable function g with $V = g(I, \mathcal{T})$. If ext is the function acting on pairs (q_i, v_i) by replacing occurrences q_i with the two states $q_i^{(1)}$ and $q_i^{(2)}$ as described

[8] 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 $\mathcal{F}_i := \{\Pi_1 \times \cdots \times \Pi_{i-1} \times \{\pi_i\} \times \Pi_{i+1} \times \cdots : \pi_i \in \Pi_i\}$.

above, then we have by construction $\hat{V} = (\mathsf{ext} \circ g)(I, \mathcal{T}_2 \circ \mathcal{T}_1)$. We now see that $(I, O, V) = (I, f(I, \mathcal{T}), g(I, \mathcal{T}))$ has exactly the same distribution as

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

Therefore, the passive security of $\mathcal{P}$ reflected in $(I, O) \perp V$ translates into $(I, \hat{O}) \perp \mathsf{ext}^{-1}(\hat{V})$ which implies $(I, \hat{O}) \perp \hat{V}$. 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
$\Rightarrow \mathcal{T}_p \perp (I, \mathcal{T}_2 \circ \mathcal{T}_1, \hat{O}, \hat{V})$	Since $\hat{O} = f(I, \mathcal{T}_2 \circ \mathcal{T}_1)$, $\hat{V} = (\mathsf{ext} \circ g)(I, \mathcal{T}_2 \circ \mathcal{T}_1)$
$\Rightarrow \mathcal{T}_p \perp (I, \hat{O}, \hat{V})$	Projection
$\Rightarrow (\mathcal{T}_p, \hat{V}) \perp (I, \hat{O})$	Using $\hat{V} \perp (I, \hat{O})$
$\Rightarrow (\mathcal{T}_p^{\hat{V}}, \hat{V}) \perp (I, \hat{O})$	$\mathcal{T}_p^{\hat{V}}$ is a function of $\mathcal{T}_p$ and $\hat{V}$.

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 12.2.

Implementation. By Proposition 12.1, 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 12.4. □

12.6. 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.

12.7. On the Interplay of Our Definitions

Note that our definition of *implemented with active security and choice* makes sure that the privatePerm actions of a two player protocol can be executed in a way that makes our security definitions adequate.

Consider some action $(\mathsf{privatePerm}, \text{Alice}, \Pi, \mathcal{F}(\cdot))$, e.g., $\Pi = \{\mathsf{id}, (1\ 2\ 3)(4\ 5\ 6\ 7)\}$ with probabilities $1/3$ and $2/3$ for the two options. The fact that Π is implemented with active security and choice (by virtue of being subset of a coupled rotation) guarantees:

- Alice may choose $\pi \in \Pi$ according to the distribution (privately in her head or using some private random device) and follow a sequence of actions that ultimately will permute the current sequence of cards by π. She will remember π for later, in particular, we record π in her permutation trace and let future distributions depend on π.

- Alice's choice is hidden from Bob. Also, neither player learns anything about the current sequence as no cards from it are being turned. In particular it makes sense to *not* attribute any observations to the players during privatePerm operations (other than Alice "observing" her choice π).

- Alice cannot perform any $\pi \notin \Pi$, which justifies that Definition 12.3 only considers attacks that respect Π.

- The procedure implementing Π fails gracefully if Alice cheats, without revealing anything about the object cards. Note that we do not consider the case of aborts in two-player protocols. This is reasonable: Firstly, the only thing an attacker can hope to achieve is destroy information (by shuffling cards in non-permitted ways, blowing his cover in the process) without learning anything he would not have learned otherwise. Secondly, all protocols we suggested can recover even if checks fail[9]. In our cases, the check operations would have to reveal an entire "register" of helping cards and would be succeeded by permutations that "fix" any discrepancy between expected result and actual result.

Note that previously we allowed for permutation protocols to fail (if a check action fails). This is not modeled here "without loss of generality", as there is no advantage for an attacker to blow his cover like that.[10]

[9] Admittedly, we cannot recover from Alice flipping over the table or running off with the cards, though.

[10] If he just wants to sabotage the calculation with no intention of staying hidden, he might as well throw over the table. But see also Section 12.7 for an explanation of how aborts in (our) permutation protocols can be avoided altogether.

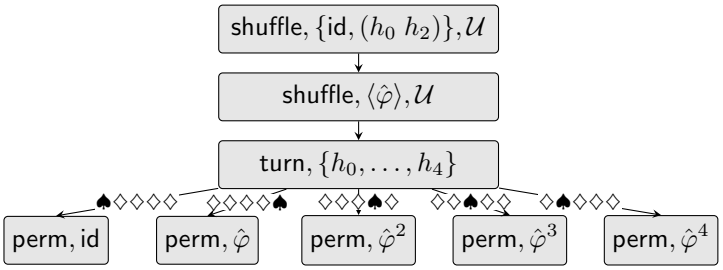


Figure 12.8.: Protocol implementing $(\mathsf{shuffle}, \Pi = \{\mathsf{id}, (1\ 2\ 3\ 4\ 5)^3\}, \mathcal{U})$ with five helping cards (details explained in the text).

What if a player aborts a permutation protocol? In our formulation, if Alice is dishonest and causes a check of a permutation protocol to fail, then it is aborted with no general way of recovering from such a state. This may be undesirable: It would certainly be better if computations could always continue, even in the presence of a dishonest player. In fact, it is possible to adapt the concept of a permutation protocol to achieve this in the case of generalized coupled rotations. The check operations would have to reveal the entire register and would be succeeded by permutations (dynamically responding to the result of the check) that "fix" the discrepancy between expected result and actual result. In that case, all but the first privatePerm action can actually be replaced by the corresponding uniform shuffle operation, i.e., uniform pile cuts. We eventually decided against the more complicated concept of permutation protocols to focus on the core ideas.

12.8. 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 [CHL13], 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 12.8 which is a "protocol implementing a shuffle" (albeit not in our formal sense as defined above). We assume that we initially have five object cards in positions 1 through 5 and a helping deck $\mathcal{H} = [\spadesuit, 4 \cdot \diamondsuit]$ originally lying in positions h_0 through h_4 (say $h_i = i + 6$). The $\spadesuit$ 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 $0 \leq k < 5$, leaving $\spadesuit$ 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 (mod 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 12.5. 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.

12.9. Achieving Input Integrity

Mizuki and Shizuya [MS14b] consider malicious players disrespecting the input format, in their case by giving ♣♣ or ♡♡ as an input, even though only ♡♣ and ♣♡ are permitted. Note that we usually leave this problem aside when assuming that inputs are already lying on the table when the protocol starts. However, see Figure 12.9 for a "state diagram" of their protocol, which is only a schematic, as the protocol is not secure in our sense. This is because the players learn whether the input was well-formed or not, contradicting independence of visible sequence trace V and input sequence I.

It is beneficial for composability/modularity to not assume knowledge about the inputs from the players running the protocol, as they might work with hidden intermediate results of the surrounding protocol.

We can think of the following strategy to integrate a player input procedure into our model, encompassing input security. One example is to have a unique starting sequence such as ♣♡♣♡ on the table, and have player 1 applying permutation (1 2) for input 1, and id otherwise. Afterwards, player 2 applies permutation (3 4) if he wants to input 1, and id otherwise. Then we arrive at the usual starting situation. This procedure however cannot be achieved with the model as we defined it, as distributions in $\mathsf{privatePerm}$ actions do not formally depend on player inputs, and if they would, the permutations done by the command are for introducing uncertainty, but are not themselves protected by our formalism. That is, one may learn something about the permutation done, as long as this does not give anything away about the input sequence/value that is to be protected, as in usual card-based protocols.

Hence, we introduce an additional formal action $(\mathsf{inputPerm}, p, \Pi, \mathsf{val})$ and an input trace $\mathcal{I}_p$ for player $p \in \{1, 2\}$, which behaves as follows:

$(\mathsf{inputPerm}, p, \Pi, \mathsf{val})$ for a player $p \in \{1, 2\}$, a permutation set $\Pi \subseteq S_\ell$ and val being a valuation that maps permutation $\pi \in \Pi$ to bits $\{0, 1\}^k$ if player p is supposed to input k bits at this step. This specifies which Boolean input the respective permutation corresponds to and is allowed to vary for the inputs of the players, as this is the case in the protocols from the literature. For simplicity, we assume it is a bijection between Π and $\{0, 1\}^k$. (If multiple permutations are needed to encode for an input, as in Figure 12.14, we omit it from the action specification

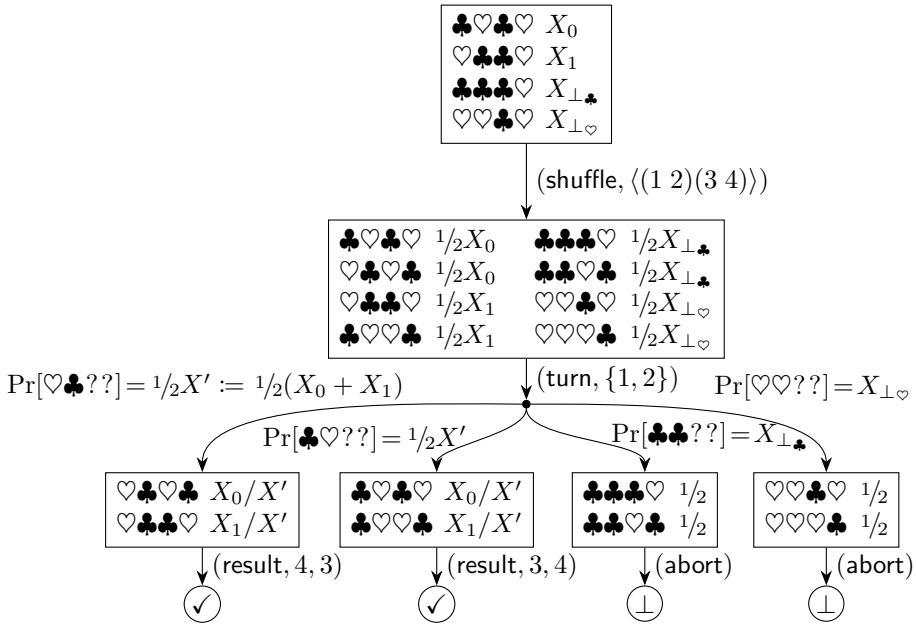


Figure 12.9.: A protocol for detecting if two cards, supposed to encode a bit in standard format, actually do so. For reference, see [MS14b]. Note that this is not a formal state tree as in Definition 7.3, the protocol does not fulfill the usual notion of security. We sketch how this can nevertheless be depicted, by allowing rational functions and constants in place of polynomials. Note however that nothing about the bit is leaked, if the input is not ♣♣ or ♡♡.

and specify it globally). Here, 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 input trace $\mathcal{I}_p$. If player p is *honest* she picks π according to her intended input.

In general, this allows the input phase and the computation phase to be arbitrarily interleaved. This may allow to save cards if not all bits are needed at the same time. However, the notions of passive and active security would need to be updated, since now the output of the protocol is not necessarily independent of the permutation trace of a player. Hence, as discussed above, we confine these actions to the start of the protocol, before anything interesting is visible. In the case of passive security we would want only that the input of the other player and the output *conditioned on* I_p are protected. For active security the notion is more tricky still – we will leave this as future work.

Let us nevertheless specify a variant of our state diagram formalism, which is compatible with inputs via permutations. See Figure 12.10 for illustration. Let N be the number of players, and $P = \{1, \ldots, N\}$ be a set formally representing them. Here, before any player inputs, there is supposed to be a unique sequence H on the table and U is the empty set. We assign to this sequence in the start state the polynomial consisting only of the formal variable $X_{\mathcal{T}_1,\ldots,\mathcal{T}_N}$, where $\mathcal{T}_p$ is the current permutation

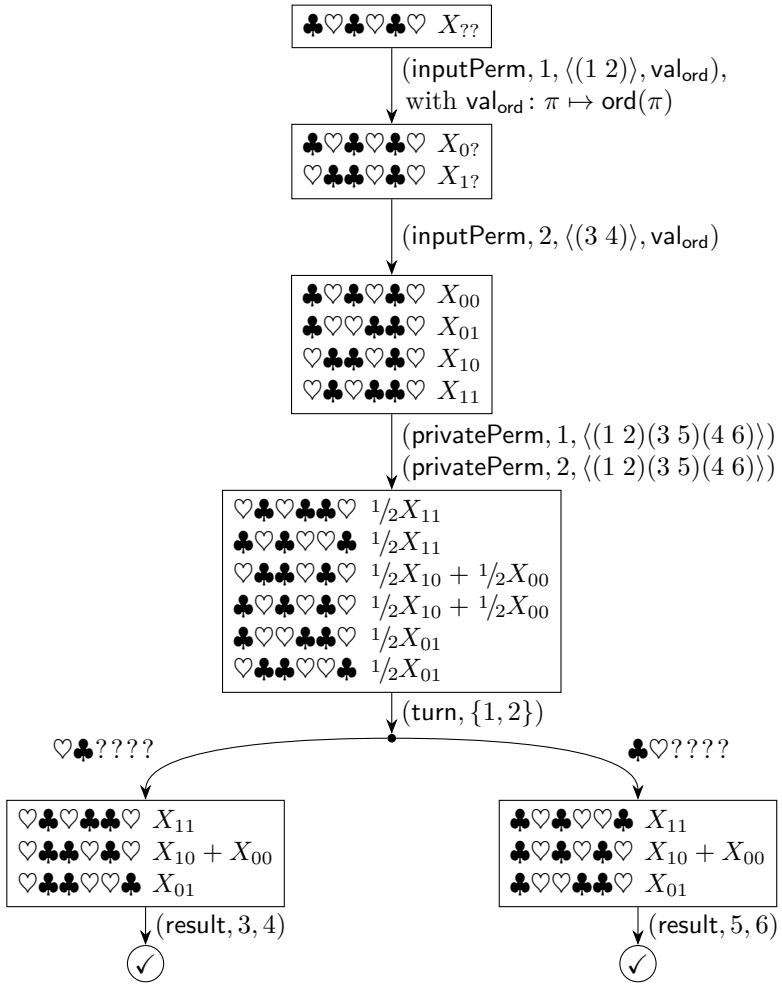


Figure 12.10.: Prepending the six-card AND protocol of Mizuki and Sone [MS09] by a separate input phase. Here, the shuffle is implemented by two privatePerm operations.

trace of player p. In this case we denote the empty sequence by () or ? and start with variable $X_{()\ldots()} = X_{?\ldots?}$.

When an $(\mathsf{inputPerm}, p, \Pi, \mathsf{val})$ is encountered, we update the variables according to the possible inputs by player p. As there is no possibility for ambiguity, we usually write the result of val directly at the respective position, as in Figure 12.10. The only exception is Figure 12.14, where we evaluate via the val_p as soon as the input is uniquely determined. A bit more formally, by evaluate in this paragraph, we mean to write $X_{\mathsf{val}_1(\mathcal{T}_1),\ldots,\mathsf{val}_N(\mathcal{T}_N)}$, if val_p are the input valuations of players $p \in \{1, \ldots, N\}$.

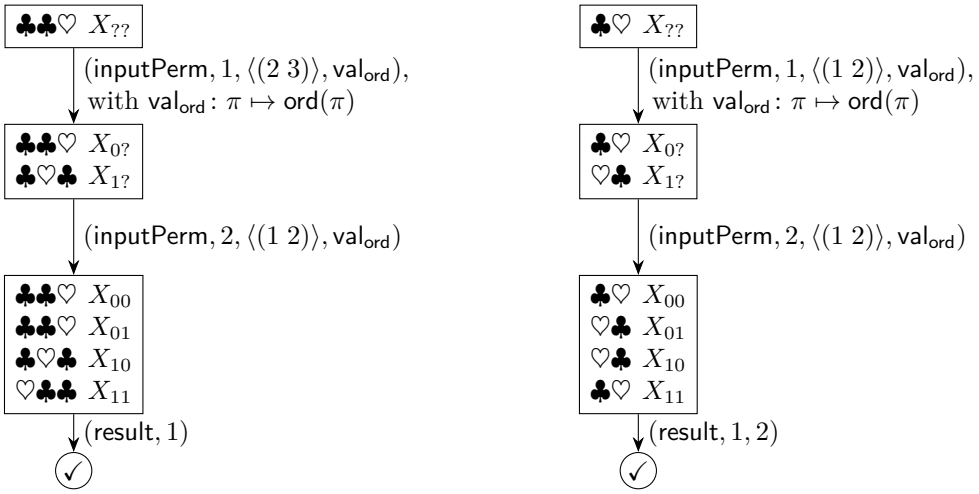


Figure 12.11.: On the *left*, we show the state tree of a three-card AND protocol [MWS15, Sect. 3.2] requiring input awareness, and with non-standard (single-card) output encoding. On the *right* is the state tree of a two-card XOR protocol of [NSIO17] requiring input awareness, and with standard output encoding $[\![\cdot]\!]_{>}$.

12.10. Protocols with Input Awareness

Using the formalism of the previous section, let us describe several protocols using inputPerm operations from the literature. For some of them, our discussion on having an actively secure implementation with choice for a shuffle helps us to show that they do *not* plausibly offer active security. This section contains new, unpublished results that will be integrated in an updated version of [KW17]. As a side-note: we can transform every protocol requiring input awareness to one without, if the permutation is encoded as in Section 8.6, and by using a sort protocol, at the cost of the necessary additional cards.

As an introduction, let us first show two very simple protocols for computing AND and XOR from [MWS15, Sect. 3.2] and [NSIO17], respectively, in Figure 12.11. For example, in the AND protocol (using start sequence ♣♣♡), the first player can either do id or (2 3), with val$_{\text{ord}}$(id) = 0 and val$_{\text{ord}}$((2 3)) = 1. The second player does essentially the same operation, but on the first two cards. In total, the ♡ is moved to the first position if and only if both players chose to apply their respective transposition (of order 1). As both permutation sets are chosen cuts, the inputPerm action has an actively secure implementation with choice. Additionally, as it only acts on two cards, we might plausibly argue that it has such an implementation even without any additional cards, by allowing the inputting players to possibly swap the card pair behind their back.

Recently, Nakai et al. [NSIO17] proposed a protocol with input awareness for computing the majority of three bits. In our interpretation, this can be written as in the left state diagram of Figure 12.12. Observe that the permutation set of player 2 is not closed. If the player has the option to apply (2 3) or (3 4), we cannot think of

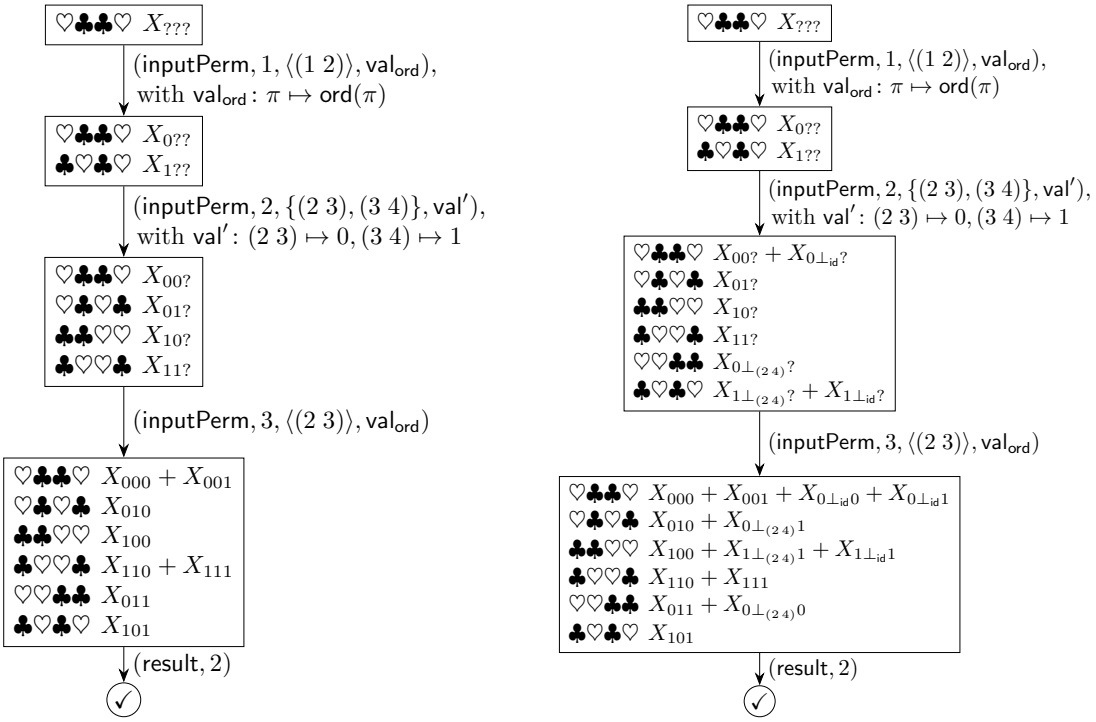


Figure 12.12.: State tree of three-inputs majority protocol by [NSIO17] on the *left*, with active attack possibility on the *right*, where val′ maps $\pi \mapsto \perp_\pi$ if $\pi \notin \{(2\,3), (3\,4)\}$. The non-standard output valuation val$_\heartsuit$ maps $\heartsuit \mapsto 1$ and $\clubsuit \mapsto 0$.

a mechanism that prevents the player from doing id or (2 4). Of course, note that the authors do claim security only in the honest-but-curious setting. On the right of Figure 12.12, we depict an attack on the protocol by also evaluating the possibilities of the two other permutations, which our valuation formally maps to $\perp_{\text{id}}$ or $\perp_{(2\,4)}$, respectively. Using this, one can observe that, e.g., player 2 can force the result to be 1 via applying (2 4), even though the other players' input is 0. (This can be read from the sequence ♡♡♣♣ having probability $X_{011} + X_{0\perp_{(2\,4)}0}$.) For comparison, note that in committed-format protocols, the current protocol using the fewest cards for three-input majority is [NMS13] utilizing eight cards.

In Figure 12.13, we give an alternative four-card majority protocols, with deck $\mathcal{D} = [\clubsuit, \clubsuit, \clubsuit, \heartsuit]$, which also does not offer active security[11], but is conceptually very simple – similar to the AND protocol we saw before. Here each player cyclically rotates the so-far relevant cards by one if he inputs a 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 conceals which one it was. In the end, the non-standard valuation outputs 1 if the two first cards are not equal. (Note that this idea can be generalized

[11] At least not without helping cards, otherwise one could use the implementation of a subset of a cut from p. 176 ("Summary")

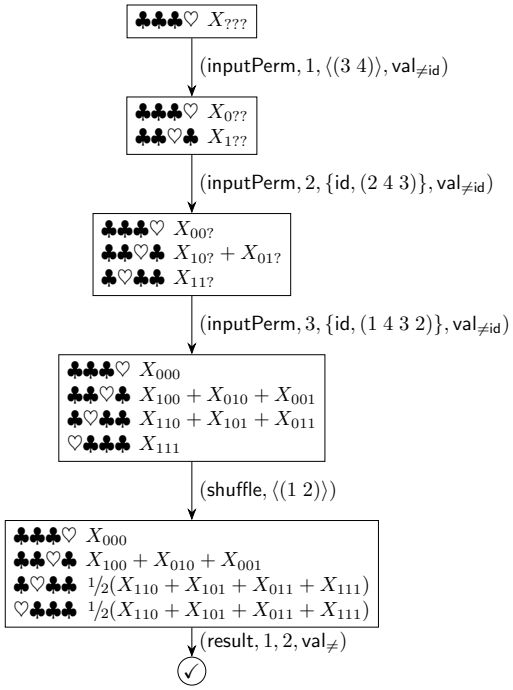


Figure 12.13.: State tree of three-inputs majority protocol with deck $\mathcal{D} = [\clubsuit, \clubsuit, \clubsuit, \heartsuit]$, with non-closed permutation sets (suffering the same attack), but being conceptually simpler. The idea is that players rotate the sequence by one, if their input is 1, or do nothing otherwise. In the end, if more than one player rotated, a heart ends up in either the first or the second column. An additional shuffle obscures which of both was the case. The non-standard output valuation $\mathsf{val}_{\neq}$ maps $\clubsuit\heartsuit \mapsto 1$, $\heartsuit\clubsuit \mapsto 1$, and $\clubsuit\clubsuit \mapsto 0$.

for more than three inputs.) Here, the shuffles are also non-closed, allowing for a similar active attack.

For an illustration of the generality of our formalism, consider the three-card three-input majority function from [WKS+18] in Figure 12.14. It is very interesting that one can reduce the number of cards to three, although at the cost of a second input phase for players 1 and 3. Note that having such a second input phase for the player makes it even harder to guarantee active security, in particular if not all pairs of choices that a player can make in the two phases do encode for an input. Hence, there is an active attack, written in the terms

$$
\begin{aligned}
A_{\clubsuit\clubsuit\heartsuit} =& X_{(\mathsf{id})\perp_{\mathsf{id}}1} + X_{(\pi_1)\perp_{\mathsf{id}}0} + X_{(\mathsf{id})\perp_{\pi_{24}}0} \\
A_{\clubsuit\heartsuit\clubsuit} =& X_{(\pi_1)\perp_{\mathsf{id}}1} + X_{(\mathsf{id})\perp_{\mathsf{id}}0} + X_{(\pi_1)\perp_{\pi_{24}}1} + X_{(\pi_1)\perp_{\pi_{24}}0} \\
A_{\heartsuit\clubsuit\clubsuit} =& X_{(\mathsf{id})\perp_{\pi_{24}}1} \\
B_{\clubsuit\clubsuit\heartsuit} =& X_{\perp_{\mathsf{id}}\perp_{\mathsf{id}}1} + X_{1\perp_{\mathsf{id}}0} + X_{\perp_{\mathsf{id}}10} + X_{\perp_{\mathsf{id}}01} + X_{\perp_{\mathsf{id}}\perp_{\pi_{24}}0} + X_{0\perp_{\mathsf{id}}1} + X_{\perp_{\pi_{15}}\perp_{\mathsf{id}}0} \\
& + X_{\perp_{\pi_{15}}00} + X_{0\perp_{\pi_{24}}0}
\end{aligned}
$$

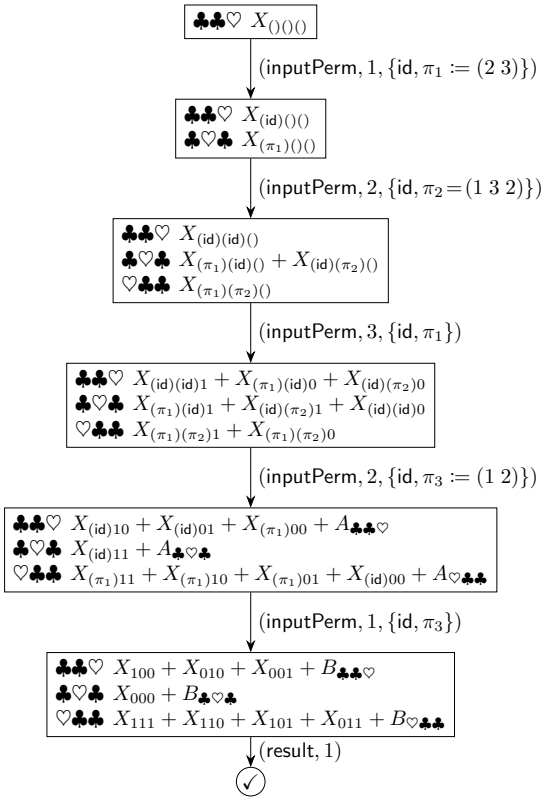


Figure 12.14.: State tree of three-inputs majority protocol with deck $\mathcal{D} = [♣, ♣, ♡]$ of [WKS+18]. As soon as a player has input all the required permutations and the valuation value is unique, we write $\mathsf{val}_p(I_p)$ into the index instead of the full input trace I_p. The input valuation differs by player, let val_p be the input valuation of player p, mapping input traces to bits. Here, $\mathsf{val}_1 \colon (\pi_1, \mathsf{id}) \mapsto 1, (\mathsf{id}, \pi_3) \mapsto 0$, $\mathsf{val}_2 \colon (\pi_2, \mathsf{id}) \mapsto 1, (\mathsf{id}, \pi_3) \mapsto 0$, and $\mathsf{val}_3 \colon (\mathsf{id}) \mapsto 1, (\pi_1) \mapsto 0$.

$$
\begin{aligned}
B_{♣♡♣} =& X_{1\perp_{\mathsf{id}}1} + X_{\perp_{\mathsf{id}}11} + X_{\perp_{\mathsf{id}}\perp_{\mathsf{id}}0} + X_{1\perp_{\pi_{24}}1} + X_{1\perp_{\pi_{24}}0} \\
&+ X_{\perp_{\pi_{15}}11} + X_{\perp_{\pi_{15}}10} + X_{\perp_{\pi_{15}}01} + X_{0\perp_{\pi_{24}}1} \\
B_{♡♣♣} =& X_{\perp_{\mathsf{id}}\perp_{\pi_{24}}1} + X_{\perp_{\mathsf{id}}00} + X_{\perp_{\pi_{15}}\perp_{\mathsf{id}}1} + X_{0\perp_{\mathsf{id}}0} + X_{\perp_{\pi_{15}}\perp_{\pi_{24}}1} + X_{\perp_{\pi_{15}}\perp_{\pi_{24}}0},
\end{aligned}
$$

where $\perp_{\mathsf{id}}$ denotes that the player falsely input id twice and $\perp_{\pi_{ij}}$ denotes that the player first input π_i then π_j, which is not a valid choice.

12.11. 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 *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.

Open Problems. Some card-minimal protocols, e.g., the general k-ary boolean function protocol of Section 8.3, 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 for these functions 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 Section 12.8 how any subset of a cut can be implemented.

Moreover, it is interesting which functionalities are realizable – at all or more efficiently – in the two player setting compared to the computational model of [MS14a], possibly by composing privatePerm operations in more sophisticated ways than done in Section 12.5. Moreover, we can formalize protocols using permutations as inputs in our model using an inputPerm action. Here, it is interesting to find protocols with the minimal number of cards for different functionalities with only closed permutation choice sets, a prerequisite to achieve *active* security using our framework.

Part III.
Secure Human-Friendly Payment

13. Introduction

This part on Secure Electronic Payment and Human–Server Interaction is directly based on the paper "Your Money or Your Life—Modeling and Analyzing the Security of Electronic Payment in the UC Framework" [AGH+19b], with some changes to account for the slightly different viewpoint taken on in this thesis.

"Your money, or your life!" – surrender your belongings or face death. This threat was used by bandits in England until the 19th century [O81]. As people often needed to carry all their valuables with them when traveling, banditry was a lucrative (albeit dangerous) endeavor. Today, electronic money transfer (EMT) systems alleviate the need to have one's valuables at hand, but introduce new threats as well. Instead of resorting to violence, modern thieves may compromise their victim's bank account. Once they are widely deployed, insecure EMT systems are notoriously difficult to transition away from – magnetic stripes are still in use today. The current state-of-the-art payment standard EMV (short for *Europay International, MasterCard and VISA*, also known as "Chip and PIN") improves on this, but falls short of providing a secure solution to payment (or money withdrawal), as shown by its many weaknesses described in literature.

Among these are practical attacks, such as (i) "cloning" chip cards by pre-computing transaction messages (so-called "pre-play attacks") [BCM+14], (ii) disabling the personal identification number (PIN) verification of stolen cards by intercepting the communication between chip card and point of sale (POS) device [MDAB10], (iii) tricking an innocent customer into accepting fraudulent transactions by relaying transaction data from a different POS (so-called "relay attacks") [DM07].

Upon close examination of these attacks one finds that these issues mainly stem from *two major false assumptions* which are baked into the design of the EMV protocol: (i) that the communication between all protocol participants (e.g., between the chip card and the POS) cannot be intercepted, and (ii) that the POS (or the automated teller machine (ATM)) itself is trustworthy. Even though these assumptions are critical for the security of EMV, they are not explicitly stated in the standardization documents [E11a; E11b; E11c]. We suggest that this is mainly because EMV has been created by a functionality-focused engineering process in which problems are fixed as they occur and features are added when necessary, rather than a design process that uses formal models and techniques. Modern cryptographic protocols in contrast are designed by first providing a formal description of the protocol, explicitly stating all necessary assumptions and then giving a proof of security. This does not make cryptographic protocols unbreakable, but it does make their potential breaking points explicit. Therefore, we argue that it is necessary to start developing electronic payment protocols by using the same methodology of rigorous formal modeling as has already been established in cryptography.

13.1. Our Contribution

In this work, we give a novel formal model for electronic payment based on the Universal Composability (UC) framework by Canetti [C01], which incorporates a stronger, but also more realistic adversarial model than has been used for the design of EMV. We first give a *formal description* of electronic payment which works for both payment at a POS and for the withdrawal of cash at an ATM. Second, we provide an ideal functionality for electronic payment, which captures the desired security guarantees for such protocols. Our model can also be used in the case where one participant is human.

We then prove a set of *general requirements* for designing such protocols. These requirements can act as a guideline for future protocol designers. Based on these results, we argue that a number of current payment systems are insecure already on a conceptual level. Inspired by this analysis, we propose a simple electronic payment protocol which mainly requires secure communication between the bank and the initiator of a transaction. We propose to realize this with a smartphone, as is common in many modern payment protocols. However, unlike these protocols, our protocol can be proven secure if *either* the smartphone *or* the ATM/POS device behaves honestly, whereas all other protocols we analyzed need to trust at least one of them exclusively.

13.2. Related Work

Secure Human–Server Communication. Basin, Radomirovic, and Schläpfer [BRS15] give an enumeration of minimal topologies of channels between a human (restricted in its abilities), a trusted server, a possibly corrupted intermediary and a trusted device, that realize an authenticated channel between the human and the server. Our work differs in two main aspects: Their model uses either fully secure or untrusted channels only and cannot account for just authenticated or just confidential communication, which is important in our setting due to the presence of CCTV cameras or shoulder-surfing. For example, we assume that everything displayed at the ATM or a user's smartphone is not confidential, while entering a PIN at the PIN pad can be done in a confidential way, by suitably covering the pad in the process. Second, our model is given in the UC framework, which gives stronger guarantees and composability, as well as security for concurrent and interleaved execution, compared to the stand-alone setting they consider.

Alternative Hardware Assumptions. As we will see later in Section 14.2, the *confirmation of payment information* by the user is an important sub-problem we aim to solve for achieving secure payment. A possible solution is "Display TAN" [B18] providing a smartcard with a display to show the transaction data. Smart-Guard [DBR16] uses such smartcards with a display together with an encrypting keyboard fixed to the card to achieve a functionality which may be used for payment. These strong hardware assumptions allow for flexible trust assumptions, accounting for several combinations of trusted/hacked status of the involved devices. Their protocol comes with a formally

verified security proof, albeit not in the UC framework. For our construction we do not propose a new kind of hardware device, but rely on the user's smartphone.

Ecash and Cryptocurrencies. Besides human–server payment protocols, there is also electronic cash, first invented by [CFN90], and modern decentralized cryptocurrencies, such as Bitcoin [GKL15], which can be used to transfer money. In general, these have very different design goals, as they care to establish an electronic money system with certain anonymity/pseudonymity properties, without the possibility to double-spend and in particular, without a trusted bank. In contrast, we are concerned with the authenticated transmission of the transaction data from a human user to the bank. To the best of our knowledge, there is no UC-based model of electronic payment as presented in our work.

EMV. EMV is not only a single payment protocol, but a complete protocol suite for electronic payment (cf. [E11a; E11b; E11c]). Protocols that are EMV-compliant might just implement the EMV interface while using another secure protocol. This means that, while there are multiple attacks against the EMV *payment protocol*, not every protocol with EMV in its name is automatically insecure. In addition to the attacks mentioned previously, there are other attacks as described by Chothia et al. [CGdR+15] and Emms et al. [EAF+14].

Anderson et al. [ABC+12] discuss whether EMV is a monolithic system reducing the possibilities for innovation. Since we use the UC framework for our model, we inherently support non-monolithic, modular systems. Sub-protocols that UC-realize each other can be exchanged for one another. Furthermore, [ABC+12] explore the possibility to use smartcards (as used by EMV) for other applications.

Degabriele et al. [DLP+12] investigate the joint security of encryption and signatures in EMV using the same key-pair. A scheme based on elliptic curves (as it is used in EMV) is proven secure in their model. However, as they conclude, their proof does not eliminate certain kinds of protocol-level attacks. Cortier et al. [CFF+17] present an EMV-compliant protocol using trusted enclaves and prove the security of their protocol using TAMARIN [BCD+18]. Both approaches lack the modularity, composability and concurrent security provided by the UC framework.

13.3. The Universal Composability Framework

The Universal Composability (UC) framework, introduced by Canetti in 2001 [C01], is a widely established tool for proving the security of cryptographic protocols based on the real-world–ideal-world paradigm. The desired security properties of a protocol are described in terms of a so-called *ideal functionality*, which can be seen as an incorruptible third party carrying out the desired task by definition. The ideal functionality *explicitly* captures the allowed influence an adversary can have and the knowledge he can gain during an execution of the protocol. Informally, a protocol π is said to UC-realize an ideal functionality $\mathcal{F}$ if there is no interactive distinguisher $\mathcal{Z}$ (the so-called *environment*)

that can distinguish between the execution of π and the execution of (the ideal protocol of) $\mathcal{F}$.

The framework is specifically well-suited for our case, as it already incorporates an adversary that can control all communication between the protocol parties. If one wants to deviate from this (e.g., when secure communication is available) one must explicitly add new functionalities for communication to the model (so-called *hybrid functionalities*), making the security assumptions of the protocol explicit.

One of the main advantages of the UC framework is that it, unlike stand-alone security models, brings a strong composition theorem. This allows for breaking protocols into smaller components and proving their security individually.

Let us give a brief review of the Universal Composability (UC) framework. It is a tool for modeling and proving the security of multi-party protocols by comparing their execution to an idealized version of the protocol, in accordance to the well-known *real-ideal* paradigm. In the UC framework, participants in a protocol π are modeled as Interactive Turing Machines (ITMs), which interact with each other by exchanging messages. The execution of such a protocol happens in the context of two additional ITMs: the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. The adversary represents the party which wants to attack the protocol, and it has full control over all exchanged messages. Further, the adversary can *corrupt* specific protocol participants (depending on the chosen model of corruption either at the beginning (static corruption) or at any point (adaptive corruption) of the protocol execution) at which point it will gain full control over their execution. The environment represents the perception of the execution from an outside point of view and it will try to use information gained by the adversary about interactions happening during a protocol execution to distinguish if it observes a real or an ideal protocol execution.

Since the adversary is able to control all communication, authenticated communication cannot be realized in the *bare* (UC) model. Even if authentic communication is assumed in the *plain* model, no non-trivial functionality can be realized [CF01]. However, one can employ so-called hybrid functionalities. A *hybrid functionality* $\mathcal{F}$ is an ITM which can be accessed directly by all participants of a protocol execution without the adversary being able to interrupt, learn or change that communication (with the exception of what is explicitly stated in the ideal functionality's description). This can, for example, be used to model a key distribution process prior to the protocol execution or a secure channel between two participants. The exact behavior of $\mathcal{F}$ must specified in advance and can include communication with the adversary. For example, a communication link using a hybrid functionality $\mathcal{F}$ such as $\mathcal{F}_{\text{AUTH}}$ might be set up in a way that $\mathcal{A}$ can read and block arbitrary network messages but cannot change its contents. The execution of a protocol is turn-based. If an *instance* of an ITM (ITI) is activated, it can perform computations and send a single message to either another party or a hybrid functionality. Then its turn ends. If an ITI receives a message, it is the next to be activated. The first ITI to be activated is the environment $\mathcal{Z}$.

Because parties may behave indeterministically, their outputs are modeled as distributions. Further, since protocol runs are parameterized (e.g., by the security parameter λ), the following definition uses probability ensembles. The output of the whole protocol

is the output of $\mathcal{Z}$ and we assume, without loss of generality, that it consists of one bit. The distribution of all outputs of $\mathcal{Z}$ is a probability ensemble of the input z and the security parameter λ.

Definition 13.1 (Ensemble of a protocol execution)**.** We denote the random variable which describes the execution of a protocol π with adversary $\mathcal{A}$, environment $\mathcal{Z}$, input z, security parameter λ as $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$. The set of random distributions $\{\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in N, z\in\{0,1\}^*}$ is denoted as $\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

In the following, we are going to compare two ensembles of probability execution. To this end, we define the indistinguishability of two binary probability ensembles x and Y as follows:

Definition 13.2 (Indistinguishability)**.** Two binary ensembles X and Y are *indistinguishable* ($X \approx Y$), if for all $c, d \in \mathbb{N}$ there exists a $k_0 \in \mathbb{N}$, so that for all $k > k_0$ and all $a \in \bigcup_{\kappa\leq k^d}\{0,1\}^\kappa$, it holds:

$$|\Pr[X(k, a) = 1] - \Pr[Y(k, a) = 1]| < k^{-c}.$$

The security of a protocol execution is based on a comparison with an execution of an idealized version of the protocol: the *ideal protocol*. The ideal protocol contains the *ideal functionality* $\mathcal{F}$, which completely realizes the properties of the analyzed protocol. In the ideal protocol, all parties only act as dummy parties which directly give their input to the ideal functionality and receive back their output without performing any computation themselves. The ideal functionality may communicate with the adversary in order to model the influence the adversary is allowed to have. We call the ideal-world adversary the "adversary simulator" $\mathcal{S}$. Modeling real-world communication networks, the real-world adversary $\mathcal{A}$ is able to block all communication between parties (even e.g., in the $\mathcal{F}_{\text{AUTH}}$-hybrid model). $\mathcal{A}$ could, for example, interrupt a protocol execution in a way that (honest) parties make no output because the protocol execution has not finished. In the ideal execution, where the corresponding output is determined by the ideal functionality, we thus allow the ideal-world adversary $\mathcal{S}$ to do the same with respect to party outputs. To this end, ideal functionalities usually handle party output by making public or private *delayed outputs*. At each delayed output, the adversary is asked to confirm the output. This confirmation may happen at any later point in the execution. Furthermore, delayed outputs do not have to be confirmed in the order they are generated by the ideal functionality. If the delayed output is public, the adversary learns the complete output. If the delayed output is private, the adversary learns only some specified parts of the output. Note that UC security does not model an "absolute" security guarantee but a guarantee relative to the defined ideal functionality.

Definition 13.3 (Ideal protocol)**.** Let $\mathcal{F}$ be an ideal functionality. Then, the ideal protocol for $\mathcal{F}$ is denoted as $\mathsf{IDEAL}_{\mathcal{F}}$.

Based on that notion, we now formalize the indistinguishability of two *subroutine-respecting* protocols in the UC framework (cf. [C01, Def. 5]). Informally, a protocol π is

subroutine-respecting if "the only input/output interface between each instance of π and other protocol instances in the system is done by the main parties of π", cf. [C01, Sect. 5.1]. The simulator's job is to simulate the presence of $\mathcal{A}$ to the environment, so that it cannot distinguish the real protocol execution from the idealized version. The security notion requires that there is a successful simulator for every adversary.

Definition 13.4 (UC-emulation)**.** Let π and φ be two subroutine-respecting PPT protocols. Then π *UC-emulates* the protocol φ, denoted by $\pi \geq \varphi$, if for all PPT $\mathcal{A}$ there exists a PPT $\mathcal{S}$, so that for all balanced PPT $\mathcal{Z}$, it holds:

$$\mathsf{EXEC}_{\pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{EXEC}_{\varphi,\mathcal{S},\mathcal{Z}}$$

We can now formally state what it means for a protocol to *(UC-)realize* a functionality.

Definition 13.5 (UC-security)**.** A protocol π (securely) *UC-realizes* an ideal functionality $\mathcal{F}$, if π UC-emulates the corresponding ideal protocol $\mathsf{IDEAL}_{\mathcal{F}}$. If the ideal functionality is clear from the context, we also say that π is *UC-secure*.

For the sake of simplicity, we often write $\pi \geq \mathcal{F}$ instead of $\pi \geq \mathsf{IDEAL}_{\mathcal{F}}$.

The UC framework is a powerful instrument for analyzing the security of protocols because it provides a composition theorem. Informally speaking, the composition theorem states that if π securely realizes an ideal functionality $\mathcal{F}$, one can use π instead of the hybrid functionality $\mathcal{F}$ in other protocols without compromising security: Let $\rho^{\mathcal{F}}$ be a protocol that makes subroutine calls to a polynomial number of instances of the ideal functionality $\mathcal{F}$. The UC composition theorem guarantees that if $\pi \geq \mathcal{F}$, then $\rho^{\pi} \geq \rho^{\mathcal{F}}$. This composition theorem can be generalized to general protocol emulation as follows:

Theorem 13.1 (UC composition theorem)**.** *Let π, φ be subroutine-respecting protocols such that $\pi \geq \varphi$. Let ρ^{φ} be an arbitrary protocol that makes subroutine calls to a polynomial number of instances of φ. Then, $\rho^{\pi} \geq \rho^{\varphi}$.*

14. A Model for Electronic Payment

As a basis for our model, observe the process of withdrawing cash at an ATM. First, there is the bank and its customer, Alice. Second, there is the money dispensing unit inside the ATM. Assuming authenticated communication from Alice to the bank and from the bank to the money dispensing unit, secure payment is easy: Alice communicates the amount of cash she needs and the identity of the money dispensing unit she expects to receive the cash from. The bank then instructs the money dispensing unit to dispense the money. However, Alice is a human and therefore cannot perform cryptographic operations required for a classical channel establishment protocol. Thus, Alice needs another party which offers a user interface to her and communicates with the bank, namely an ATM.

This does not only apply to cash withdrawal but can be extended to EMT in general. To this end, think of Alice as the *initiator* of a transaction and the money dispensing unit as the *receiver*. The process of money withdrawal can now be framed as a payment of money from Alice's account to the account of the money dispensing unit (which, upon receiving money, promptly outputs cash) using the ATM as an (input) *device*. The same works for the point of sale: here, the device's owner (e.g., the supermarket) is the receiver.

Regarding our adversarial model, as discussed earlier, we make no assumption about the trustworthiness of the ATM whatsoever and do assume that the adversary has control over all communication. We do make certain assumptions regarding the trustworthiness of different protocol participants. First, we assume the money dispensing unit (or receiver in general) to be trusted. If it is under adversarial control, the adversary could simply dispense money at will. Second, since our work focuses on the challenges that arise from the interaction of humans with untrustworthy devices over insecure communication, we do not model the bank's book-keeping and therefore assume the bank to be incorruptible. Third, for reasons of simplicity, our model only considers a single bank, even though in practice most transactions involve at least two banks. This is justified, however, as banks in general can communicate securely with each other.

14.1. Modeling Electronic Payment in the UC framework

In the following, to simplify the model, we consider the case of *static corruption*, where parties may only be corrupted prior to protocol execution. Extending our work to adaptive corruption is left for future work.

We denote the set of initiators as S_{I}, the set of receivers as S_{R}, the set of devices as S_{D} and the bank as B. We also define a mapping $D\colon S_{\mathrm{R}} \to S_{\mathrm{D}}$ of receivers to single devices ($D(R)$) to explicitly name which device belongs to which receiver.

In order to model the adversary's probability of successfully attacking credentials like PINs, we introduce a parametrized distribution $\mathcal{D}$. Let X denote the event of a successful attack. Then $\mathcal{D}\colon A \to F_X$ maps a value d (e.g., the amount) from a domain A (e.g., $\mathbb{Q}$) to a probability mass function $f_{d,X} \in F_X$ over $\{\texttt{confirm}, \texttt{reject}\}$. An adversary's success probability of correctly guessing a four-digit PIN chosen uniformly at random with one try could be modeled as follows: $\mathcal{D}(m_\$) = f_X$ for all $m_\$ \in \mathbb{Q}$ with $f_X(\texttt{confirm}) = \frac{1}{10000}$, $f_X(\texttt{reject}) = \frac{9999}{10000}$. $\mathcal{D}$ could also map different $d \in A$ to different $f_{X,d}$, modeling that transactions with small amounts require less protection than ones with bigger amounts. $\mathcal{F}_\mathcal{D}$ is the ideal functionality $\mathcal{F}$ parametrized with $\mathcal{D}$. Ideal functionalities may have additional parameters, either implicit or explicit ones passed as arguments, e.g., $\mathcal{F}_\mathcal{D}(A, B)$.

In the best possible scenario, ideal payment would work as follows: the initiator submits his desired transaction data to an ideal functionality, which then notifies the bank and the receiver about who paid which amount of money to whom without involvement of the adversary whatsoever. In our adversarial model, no payment protocol realizes this strong ideal functionality: an attacker who controls all communication will at least be able to observe that a transaction takes place, even if he cannot see or change its contents. What is more, such a strict security definition would ignore the fact that in all payment protocols which rely on the initiator being protected by a short secret (like a PIN), an attacker always has a small chance of success by guessing the secret correctly.

Our ideal functionality for electronic payment is thus designed with regards to the following principles: (i) The adversary always gains access to all transaction data. An electronic payment operation can be secure (that is all participants of the transaction get notified about the correct and non-manipulated transaction data) without the transaction data being secret. (ii) The adversary can always successfully change the transaction data at will with a small probability (e.g., if he guesses the PIN correctly). (iii) The payment operation occurs in three stages. In the first stage, the initiator inputs his intended transaction data which the adversary can change at will. This models that a corrupted input device will always be able to change the human initiator's transaction data, even if it will be detected at a later stage. In the second and third stage, the receiver and the bank are notified about the transaction data. The resulting functionality is depicted in Figure 14.1.

14.2. Confirmation is Key

Since the human initiator of a transaction cannot be sure that an untrusted input device correctly processes his transaction data, he needs a way of confirming the transaction data with the bank before the transaction is processed. We formalize this confirmation mechanism within the ideal functionality $\mathcal{F}_{\text{CONF}}$ (specified in Figure 14.2). $\mathcal{F}_{\text{CONF}}$ is a two-party functionality which allows a sender to transmit a message and the receiver of the message to *confirm* or *reject* it. As with the ideal payment functionality, the adversary gets the chance to force a confirmation with a certain probability, modeling

The Ideal Functionality for Electronic Payment $\mathcal{F}_{\text{PAY},\mathcal{D}}(I, B, R)$.

Parametrized by a set of receivers S_{R}, a designated receiver $R \in S_{\text{R}}$, a set of initiators S_{I}, an initiator $I \in S_{\text{I}}$, the bank B and a parametrized distribution $\mathcal{D}$.
Initialize $I' = I$, $R' = R$, *attacked* = `no`.
Assertion: At any time, $I, I' \in S_{\text{I}}$ and $R, R' \in S_{\text{R}}$. If the assertion is violated, halt.

Phase 1: Collecting Information

1. Upon receipt of message $(\texttt{transfer}, sid, R, m_\$)$ from I: Send $(sid, I, R, m_\$)$ to the adversary, receive $(sid, I', R', m'_\$)$ and output $(\texttt{input-received}, sid, I', R', m'_\$)$ to B.

Phase 2: Confirmation and Execution

2. Resume upon instruction by the adversary.
3. If I' is honest, $(I', R', m'_\$) \neq (I, R, m_\$)$ and *attacked* = `no`, halt.
4. Make a public delayed output of $(\texttt{received}, sid, I', m'_\$)$ to R'.

Phase 3: Ensuring Consistency

5. Resume upon instruction by the adversary and make a public delayed output of $(\texttt{processed}, sid, I', R', m'_\$)$ to B. Halt upon confirmation by the adversary.

Attack

- Upon receiving an input $(\texttt{attack}, sid)$ in Phase 2 from the adversary, sample an element $b \in \{\texttt{confirm}, \texttt{reject}\}$ according to $\mathcal{D}(m'_\$)$. If $b = \texttt{confirm}$, set *attacked* = `yes`, otherwise set *attacked* = `no`. Return $(\texttt{attack}, sid, attacked)$ to the adversary. Ignore all further `attack` queries.

Figure 14.1.: The ideal functionality $\mathcal{F}_{\text{PAY}}$ for electronic payment.

the insecurity inherent to real-world protocols which use short secrets. Note that he can always force the confirmation to be rejected.

To realize $\mathcal{F}_{\text{PAY}}$, we need authenticated communication from the bank to the receiver, so that the receiver can be notified of the transaction. For most real-world payment protocols, this authenticated communication is easy to establish, since receivers are electronic devices and not humans. In the case of cash withdrawal, the bank owns the money dispensing unit and can pre-distribute cryptographic keys to establish authenticated communication.

Using $\mathcal{F}_{\text{CONF}}$ and $\mathcal{F}_{\text{AUTH}}$ [C01, Sect. 6.3], we propose a protocol π_{PAY} which realizes $\mathcal{F}_{\text{PAY}}$. This protocol is informally depicted in Figure 14.3. The comprehensive formal description of the protocol can be found in the full version [AGH+19a]. Having defined all required protocols and functionalities, we are now ready to state our theorem.

Theorem 14.1. *Let I, B, R, and $D(R)$ ITMs, where I is human, and B and R are honest. Then, π_{PAY}, informally depicted in Figure 14.3, UC-realizes $\mathcal{F}_{\text{PAY},\mathcal{D}}(I, B, R)$ in the $\mathcal{F}_{\text{AUTH}}(B, R), \mathcal{F}_{\text{AUTH}}(R, B), \mathcal{F}_{\text{CONF},\mathcal{D}}(B, I)$-hybrid model.*

The Ideal Functionality for Confirmation $\mathcal{F}_{\text{CONF},\mathcal{D}}(S, C)$.

Parametrized by the message sender S, the respective confirmer C and a parametrized distribution $\mathcal{D}$.
Initialize *attacked* = `no`, *initiated* = `no`, *completed* = `no`.

- Upon receiving (`initiate`, *sid*, C, m) from ITI S, make a public delayed output of (`initiate`, *sid*, S, m) to C and set *initiated* = `yes`. Ignore all subsequent `initiate` messages.
- Upon receiving (`reply`, *sid*, S, b) from ITI C when *initiated* = `yes`, *completed* = `no` and $b \in \{$`confirm`, `reject`$\}$: Make a public delayed output of (`answer`, *sid*, C, b) to S. Upon confirmation from the adversary, set *completed* = `yes` and halt.
- Upon receiving (`force-confirm`, *sid*) from the adversary, assert that C is honest, *initiated* = `yes`, *completed* = `no` and *attacked* = `no`. If this holds, set *attacked* = `yes` and sample an element $b \in \{$`confirm`, `reject`$\}$ according to $\mathcal{D}(m)$. If b = `confirm`, set *completed* = `yes` and make a public delayed output of (`answer`, *sid*, C, `confirm`) to S and halt upon confirmation by the adversary. Otherwise, return (`fail`, *sid*) to the adversary.
- Upon receiving (`force-reject`, *sid*) from the adversary, assert that C is honest, *initiated* = `yes`, *completed* = `no` and *attacked* = `no`. If this holds, set *attacked* = `yes` and *completed* = `yes`, make a public delayed output of (`answer`, *sid*, C, `reject`) to S and halt upon confirmation by the adversary. Otherwise, return (`fail`, *sid*) to the adversary.

Figure 14.2.: The ideal functionality for confirmation of messages.

The technical proof is given in the full version [AGH+19a]. Even though this might seem unsurprising at first, this allows us to break down the complexity of realizing $\mathcal{F}_{\text{PAY}}$ into two easier problems: realizing a confirmation mechanism between the initiator and the bank and realizing authenticated communication between the receiver and the bank.

14.3. How Our Model Captures Existing Attacks

One of our main motivations for establishing a new formal model for electronic payment is to make trust assumptions explicit in order to detect unrealistic ones which enable practical attacks like [BCM+14], [MDAB10] and [DM07]. Thus, our model needs to be able to capture these kinds of attacks. Protocols analyzed within our framework must be insecure if they allow for these attacks. In the following, we explain how this is achieved.

Changing Transaction Data. An adversary controlling all communication or the input device can easily change transaction data. Protocols which allow this unconditionally

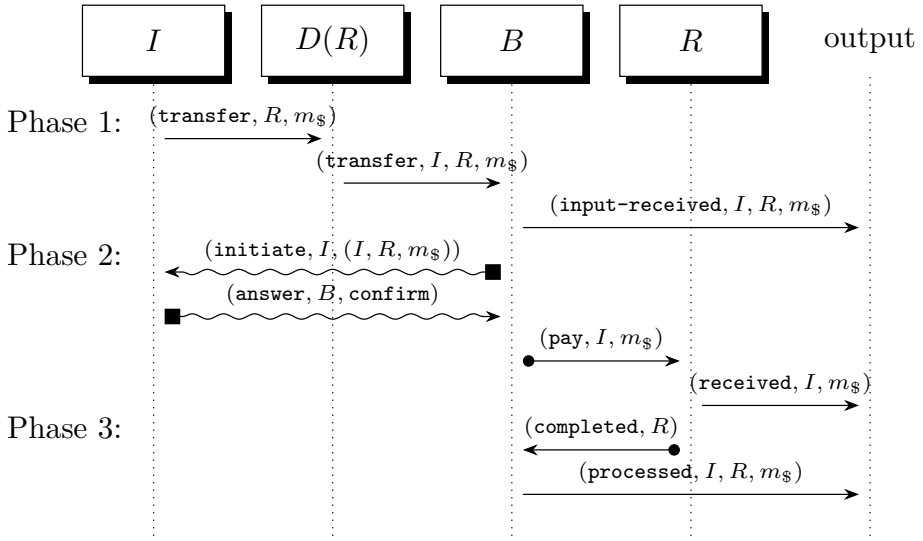


Figure 14.3.: The protocol π_{PAY} realizing $\mathcal{F}_{\mathrm{PAY},\mathcal{D}}(I, B, R)$ using $\mathcal{F}_{\mathrm{CONF},\mathcal{D}}(B, I)$, $\mathcal{F}_{\mathrm{AUTH}}(B, R)$ and $\mathcal{F}_{\mathrm{AUTH}}(R, B)$, the latter two depicted as $\bullet\!\!\rightarrow$. The use of an imperfect $\mathcal{F}_{\mathrm{CONF},\mathcal{D}}$ is depicted via $\blacksquare\!\!\rightarrow$. The protocol is between the human initiator I, the ATM $D(R)$, the bank B and the money dispenser R. The protocol proceeds in three phases, namely (1) the information collection phase, (2) the confirmation and execution phase and (3) the phase which ensures a consistent view on what happened.

are insecure in our model, since $\mathcal{F}_{\mathrm{PAY}}$ only allows to change the transaction data successfully if the adversary mounts a successful attack (i.e., guesses the initiator's PIN in the real world) or the (possibly changed) initiator is corrupted.

Relay Attacks. The aim of a relay attack [DM07] is to get Alice to authorize an unintended transaction, which benefits the attacker, by relaying legitimate protocol messages between the POS device she uses to pay for goods to another POS device which Alice uses at the same time. If Alice's input device is corrupted, she cannot know with certainty which transaction data she authorizes. Depending on the point of view, this amounts to either changing the *receiver* of a transaction initiated by Alice or changing the *initiator* of a transaction initiated by a third party Carol. Thus, in our model, this attack is just a special case of *changing transaction data*.

15. Requirements for Secure Payment

The core challenge when realizing $\mathcal{F}_{\text{PAY}}$ is the authenticated transmission of transaction data from the (human) initiator to the bank. This can also be captured formally: the functionality $\mathcal{F}_{\text{PAY}}$ can be used to implement the ideal authenticated communication functionality $\mathcal{F}_{\text{AUTH}}$ between initiator and bank (up to the attack success probability captured by the distribution $\mathcal{D}$) by encoding the message as an amount to be transmitted. We use this insight to establish several guidelines for the design of secure payment protocols: First, we state a *necessary* condition for protocols that realize $\mathcal{F}_{\text{PAY}}$: they must use setups that are strong enough to realize authenticated communication between the (human) initiator and the bank. Protocol designers can use this condition as an easily checkable criterion for the *insecurity* of payment protocols. Second, we state several setups that are *sufficient* for realizing $\mathcal{F}_{\text{PAY}}$.

For the proofs, we define an ideal functionality $\mathcal{F}_{\text{AUTH},\mathcal{D}}$, analogous to $\mathcal{F}_{\text{CONF}}$ and $\mathcal{F}_{\text{PAY}}$, that allows the adversary to change the message to be sent with a certain probability parametrized by $\mathcal{D}$. Analogous to $\mathcal{F}_{\text{CONF}}$ as shown in Figure 14.2, we define an ideal functionality $\mathcal{F}_{\text{AUTH},\mathcal{D}}$ that allows the adversary to change the message transmitted or force the transmission of messages according to some parametrized probability distribution $\mathcal{D}$. For the sake of an easier exposition, we consider ideal functionalities like $\mathcal{F}_{\text{AUTH},\mathcal{D}}$ that model the transmission of *only one* message. This is in line with the protocols we consider. If multiple messages have to be transmitted over the same "channel", this model does not adequately capture reality, as an adversary would be able to attack *each* transmission independently. In this case, ideal functionalities for channels like $\mathcal{F}_{\text{SC}}$ (cf. [CK02]) can be adapted the same way.

15.1. Establishing Criteria for Secure Electronic Payment

In this section, we establish necessary and sufficient criteria for secure electronic payment. Let $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I,R)$ denote the imperfect ideal authenticated communication functionality between parties I and R, and $\mathcal{F}_{\text{SMT},\mathcal{D}}(I,R)$ the corresponding ideal secure message transfer functionality (where successful attacks relative to $\mathcal{D}$ results in loss of secrecy and authenticity). For a formal description, see the full version [AGH+19a]. Throughout this section, let I, B, R be ITMs, where I is human[1], B is honest and $\mathcal{D}$ a parametrized distribution. We obtain the following theorem:

Theorem 15.1. *There exists a protocol π that UC-realizes $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I,R)$ in the $\mathcal{F}_{\text{PAY},\mathcal{D}}(I,\star,R)$-hybrid model, where $\star$ is an arbitrary protocol party.*

[1]Note that our results hold for arbitrary I.

Ideal functionality $\mathcal{F}_{\text{AUTH},\mathcal{D}}(S,R)$.

Parametrized by a sender S, a receiver R and a parametrized probability distribution $\mathcal{D}$.
Initialize *initiated* = `no`, *attacked* = `no`.

- Upon receiving an input $(\texttt{Send}, S, R, sid, m)$ from ITI S, set *initiated* = 1 and generate a public delayed output $(\texttt{Sent}, S, R, sid, m)$ to R and halt.
- Upon receiving $(\texttt{force-send}, sid, m', v)$ from the adversary: If *initiated* = `yes` and the $(\texttt{Sent}, S, R, sid, m)$ output has already been delivered to R or if *attacked* = `yes`, do nothing. Otherwise, sample $r \leftarrow \mathcal{D}(m')$ and set *attacked* = `yes`. If $r = v$, output $(\texttt{Sent}, S, R, sid, m')$ to R and halt. Otherwise, output $(\texttt{fail}, sid)$ to the adversary.

Figure 15.1.: Ideal functionality $\mathcal{F}_{\text{AUTH},\mathcal{D}}$.

Proof Sketch. First, let us outline the description of the protocol π. Upon receiving input $(\texttt{Send}, S, R, sid, m)$, S starts an interaction with $\mathcal{F}_{\text{PAY}}$ where S acts as initiator, receiver and device while R acts as the bank. S sends $(\texttt{transfer}, sid', S, m)$ to $\mathcal{F}_{\text{PAY}}$. When receiving $(\texttt{processed}, sid', S, R, m)$, R outputs $(\texttt{Sent}, I, sid, m')$.

We now consider how to simulate when $\mathcal{Z}$ has changed either the sender or the amount (corresponding to the message). When $\mathcal{F}_{\text{AUTH},\mathcal{D}}$ asks $\mathcal{S}$ for confirmation about the delayed output, $\mathcal{S}$ waits until receiving $(\texttt{attack}, v)$ from $\mathcal{Z}$ and sends $(\texttt{force-send}, sid, amount', v)$ to $\mathcal{F}_{\text{AUTH},\mathcal{D}}$. If the output is $(\texttt{fail}, sid)$, $\mathcal{S}$ reports $(\texttt{fail}, sid)$ to $\mathcal{Z}$. If the attack is successful, B outputs the correct value in the interaction with $\mathcal{F}_{\text{PAY}}$. □

Note that authenticated communication between R and S (corresponding to bank and receiver) is not necessary as π executes $\mathcal{F}_{\text{PAY}}$ only until the first message to R (corresponding to the bank) is sent.

In particular, Theorem 15.1 implies that protocols without any authenticated communication or only between the bank and the receiver cannot realize $\mathcal{F}_{\text{PAY}}$:

Corollary 15.1. *Let π be a protocol that is in the $\mathcal{F}_{\text{AUTH}}(B,R)$, $\mathcal{F}_{\text{AUTH}}(R,B)$-hybrid model only (in particular, there is no authenticated communication between I and B). Then there is no protocol ρ in the bare model such that ρ^{π} UC-realizes $\mathcal{F}_{\text{PAY},\mathcal{D}}(I,B,R)$ if $\mathcal{D}$ admits the adversary at least a non-negligible successful attack probability.*

This insight can be generalized and gives a *necessary condition*: A protocol π that realizes $\mathcal{F}_{\text{PAY},\mathcal{D}}(I,B,R)$ must use setups that can be used to realize $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I,B)$.

Theorem 15.2 (Necessary Requirements for Setups)**.** *Let F be a set of ideal functionalities. Let Π be the set of all subroutine-respecting protocols with the set of protocol parties $P \subseteq \{I,R,B\}$ that use only ideal functionalities in F. If there is no protocol $\pi \in \Pi$ such that π^F realizes $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I,B)$, then there is no protocol $\rho \in \Pi$ such that ρ^F realizes $\mathcal{F}_{\text{PAY},\mathcal{D}}(I,B,R)$.*

Theorem 15.2 can be easily shown using Theorem 15.1 and the UC composition theorem:

Proof. Let $F' \subseteq F$. Suppose for the sake of contradiction that there exists a protocol $\rho \in \Pi$ such that $\rho^{F'} \geq \mathcal{F}_{\text{PAY}}(I, R, B, \mathcal{D})$ and for all protocols $\pi \in \Pi$, it holds that $\pi^{F'} \not\geq \mathcal{F}_{\text{AUTH},\mathcal{D}}(I, B)$. By Theorem 15.1, there exists a protocol $\tau \in \Pi$ that UC-realizes $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I, B)$ in the $\mathcal{F}_{\text{PAY}}(I, R, B, \mathcal{D})$-hybrid model. By the UC composition theorem, it follows that $\tau^{\rho^{F'}} \geq \mathcal{F}_{\text{AUTH},\mathcal{D}}(I, R)$. By setting $\pi := \tau^{\rho}$, we obtain a contradiction that there is no protocol $\pi \in \Pi$ such that $\pi^{F'}$ UC-realizes $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I, B)$. □

Conversely, it is easy to see that $\mathcal{F}_{\text{PAY},\mathcal{D}}$ can be realized by (also) using $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I, B)$, for example. Several sufficient requirements are stated in the following theorem:

Theorem 15.3 (Sufficient Requirements)**.** *Let π be a protocol that UC-realizes*

1. $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I, B)$, or
2. $\mathcal{F}_{\text{SMT},\mathcal{D}}(B, I)$, or
3. $\mathcal{F}_{\text{CONF},\mathcal{D}}(B, I)$.

Then, there exists a protocol ρ s.t. ρ^{π} UC-realizes $\mathcal{F}_{\text{PAY},\mathcal{D}}(I, B, R)$ in the $\mathcal{F}_{\text{AUTH}}(B, R)$, $\mathcal{F}_{\text{AUTH}}(R, B)$-hybrid model.

Proof Sketch. (i) holds because $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I, B)$ can be used instead of $\mathcal{F}_{\text{CONF},\mathcal{D}}(B, I)$ in π_{PAY}. (ii) holds because $\mathcal{F}_{\text{SMT},\mathcal{D}}(I, B)$ can be used to realize $\mathcal{F}_{\text{AUTH},\mathcal{D}}(I, B)$. (iii) follows from Theorem 14.1. □

15.2. Towards Realistic Assumptions

Protocols are built on assumptions to achieve security. However, there often is a huge discrepancy regarding to how realistic these assumptions are. EMV relies on the security of the ATM which is often publicly accessible and offers a large attack surface. Unpatched operating systems and exposed USB interfaces are only two examples for vulnerabilities that have been exploited successfully. As explained in Section 14.2, a secure protocol can be constructed by establishing a confirmation mechanism. However, if the input device is corrupted, an additional device is required.

Such additional devices could for example be TAN generators or smartphones. In principle these allow for the creation of protocols that are secure in our model. However, smartphones, which are increasingly used to replace smartcards, regularly call for attention because of vulnerabilities. They are complex systems connected to the Internet and are thus more vulnerable to attacks – especially if they are operated by people without expertise in IT security. However, this dilemma can be resolved by requiring trust in only *one of the two devices*. We call this property 1-of-2 (*one-out-of-two*) security (which is, in the case of authentication, also known as multi-factor authentication). This means that a protocol is still secure if one of the two devices is corrupted, no matter which one. We argue that, in addition to realizing $\mathcal{F}_{\text{PAY}}$, payment protocols should support this property in order to further reduce the attack surface.

16. Analysis of Current Payment Protocols

In this chapter, we use our acquired insights to analyze current protocols for withdrawing cash, paying at the POS, and online banking. Table 16.1 summarizes our findings. Our model allows for a structured and fast categorization of payment protocols on a conceptual level, even without a detailed protocol description. Even though EMV is the most widely used standard for payments, we do not elaborate on its security in this chapter. As mentioned before, its design incorporates at least two assumptions that do not hold, as several attacks have been demonstrated. Current payment protocols such as Google Pay, Apple Pay, Samsung Pay, Microsoft Pay and Garmin Pay provide an app that uses the EMV contactless standard to communicate with existing POS devices via near-field communication [S16; Va]. Since they rely on Consumer Device Cardholder Verification Method, the user is authenticated by the mobile device exclusively. Currently, these apps use a PIN, a fingerprint or face recognition and thus do not incorporate a second device such as the POS device for authentication. Therefore the security of the protocol is solely based on the mobile device.

The protocols discussed in this section make additional implicit assumptions, which we believe to be plausible, but want to make explicit. These include the following: (i) An additional trusted device beside the input device. This is a plausible assumption if the device is simple, less so if it is a smartphone. However, using an additional device could enable protocols to be 1-of-2-secure. (ii) Authenticated communication between the initiator of a transaction and an additional personal device. This is a realistic assumption, since the initiator owns the device. Likewise the initiator can authenticate themselves to the device, e.g., by unlocking the screen of a mobile device. (iii) Confidential communication from the initiator to the ATM, which can be realized by covering the PIN pad with one's hand if the ATM is not compromised. (iv) Confidential communication from the ATM to the bank. This can be realized using public-key cryptography.

In the following, we examine multiple protocols for cash withdrawal and online banking.

Cardless Cash. Cardless Cash [C18] is an app-based protocol for cash withdrawal offered by numerous banks in Australia. In its most simple variant, it works as follows: After registration, the app can be used to create a "cash code" by entering the desired amount and a phone number. The phone number is used to send a PIN via SMS and allows to permit someone else to withdraw cash. To dispense the cash, the PIN has to be entered at the ATM alongside the cash code. The security of the protocol is solely based on the ATM, since all relevant information is entered there and no additional confirmation mechanism is established.

VR-mobileCash. VR-mobileCash [Vb] is another app-based protocol for cash withdrawal offered by Volks- und Raiffeisenbanken, a German association of banks. Upon registration, the user receives the mobile personal identification number (mPIN), which has to be entered on the ATM later on to confirm a transaction. To withdraw cash, the user has to enter the desired amount in the app. After selecting mobile payment at the ATM, the ATM shows a mobile transaction identification number (mTIN) which has to be entered in the app. The ATM then shows the requested amount and asks the user to enter the mPIN. If the mPIN is correct the ATM dispenses the requested amount of cash.

Although not stated explicitly in the public documentation, the mobile device has to be online during the transaction, as the ATM is informed about the transaction data. If the mobile device is corrupted but the ATM is honest, a user can detect an attack because he has to confirm the transaction by entering the mPIN at the ATM and thus verifies the location of the ATM. However, a *corrupted ATM* can employ a relay attack by displaying the mTIN of another corrupted ATM and forwarding the entered mPIN to it thus allowing the second corrupted ATM to dispense the cash. This could be fixed by adding a serial number imprinted on the ATM which is also displayed in the app after entering the mTIN. Thereby VR-mobileCash could potentially realize $\mathcal{F}_{\text{PAY}}$ and even be 1-of-2-secure.

chipTAN. ChipTAN [P18] is a protocol for online banking widely used in Germany. Here, the initiator uses a computer as an input device and possesses two additional personal devices: a transaction authentication number (TAN) generator and a smartcard. The TAN generator is used to confirm transactions and thus realizes a confirmation mechanism. This works as follows: First, a transaction has to be requested in the browser. Then, the banking website shows a flickering code. The user puts the smartcard into the TAN generator and scans the flickering code. After reviewing the transaction data presented on the personal device, he presses a button which reveals a TAN that has to be entered into the website.

This protocol satisfies all requirements for a secure realization of $\mathcal{F}_{\text{PAY}}$ by establishing a confirmation channel that allows a user to detect tampering of the transaction data. What is more, the protocol potentially provides a form of 1-of-2 security, since as long as either the input device or alternatively the TAN generator together with the smartcard are uncorrupted, there exists a confirmation mechanism from the bank to the initiator. This is only true for single transactions, however (see [R09] for details).

photoTAN. PhotoTAN (or QR-TAN) is a variant of chipTAN, where the code to transmit data to the TAN generator is encrypted by the bank. Furthermore, a smartphone can be used as an alternative to a special-purpose TAN generator. In our model, this encryption does not have an impact on security, since the transaction data is not confidential and is displayed on the smartphone nonetheless. However, some banking apps for photoTAN [D18; C13] show the TAN immediately after scanning the code and before the transaction data have been confirmed by the user. Thus, in the scenario of

Table 16.1.: Comparison of different payment protocols. A protocol is marked as offline, if the additional device does not require an Internet connection during the payment process. The security of a protocol is put in parentheses if it meets our requirements for a secure protocol but has not been proven secure.

Protocol	**Offline**	**Secure**	**Applicable for**
Cardless Cash	✓	×	Withdrawal
VR-mobileCash	×	×	Withdrawal
chipTAN	✓	(✓: 1-of-2)	Online banking
photoTAN	✓	(✓: 1-of-2)	Online banking
L-Pay (our scheme)	✓	✓: 1-of-2	Withdrawal, PoS

cash withdrawal, an attacker that corrupted an ATM and deploys a camera, monitoring the ATM, could change the submitted transaction data at the ATM, read the TAN from the victim's display and confirm the transaction without the initiator's consent.

17. Realizing Secure Electronic Payment

In Section 14.2, we gave a protocol π_{PAY} realizing $\mathcal{F}_{\text{PAY},\mathcal{D}}(I, R, B)$ in the $\mathcal{F}_{\text{AUTH}}(B, R)$, $\mathcal{F}_{\text{AUTH}}(R, B)$, $\mathcal{F}_{\text{CONF},\mathcal{D}}(B, I)$-hybrid model. While realizing $\mathcal{F}_{\text{AUTH}}$ between the bank and the receiver is simple, realizing $\mathcal{F}_{\text{CONF},\mathcal{D}}(B, I)$ in a way suitable for humans is a challenge under realistic trust assumptions.

The protocols in Chapter 16 use one or more additional devices, such as smartphones, smartcards or TAN generator to give the initiator a confirmation capability. Yet all cash withdrawal protocols still need a trusted ATM. In the following, we improve on this by presenting a simple offline protocol called L-Conf (informally described by $\pi_{\text{L-Conf}}$ in Figure 17.1). It is inspired by chipTAN and photoTAN which use similar mechanisms. Our protocol is secure even if either the additional device A, such as the initiator's smartphone, or the input device is compromised. We call this property *one-out-of-two security*, formally defined as follows:

Definition 17.1 (One-out-of-two security)**.** Let X_1, X_2 be Boolean variables, π a protocol and $\mathcal{F}$ an ideal functionality. We say that π *UC-realizes* $\mathcal{F}$ *with one-out-of-two security relative to* X_1 *and* X_2, if $X_1 \vee X_2$ implies that π UC-realizes $\mathcal{F}$.

$\pi_{\text{L-Conf}}$ can be used with π_{PAY} to realize $\mathcal{F}_{\text{PAY}}$. We call the resulting protocol L-Pay. The protocol starts with a setup phase: The bank B and the initiator I agree on a PIN and the initiator's smartphone shares keys with the bank for an authenticated secret-key encryption scheme.

The main part, depicted in Figure 17.1, consists of the execution of two protocols π_1 and π_2, each realizing $\mathcal{F}_{\text{CONF}}(B, I)$ under different assumptions. By combining their results, the composed protocol $\pi_{\text{L-Conf}}$ realizes $\mathcal{F}_{\text{CONF}}(B, I)$ even if either the input device or the additional device is compromised.

In π_1, the bank first encrypts the transaction data together with a fresh one-time TAN. The ciphertext is then transmitted to the initiator's input device, displayed appropriately, transferred to the smartphone (e.g., by scanning a QR code) and is decrypted. The TAN is only shown after the transaction data has been checked and *explicitly confirmed* by the initiator. Afterwards, the initiator enters the TAN into the input device.

In order to achieve security even if the initiator's smartphone is corrupted, π_2 requires the initiator to also check and confirm the transaction by entering his PIN into the input device (confidentially over $\mathcal{F}_{\text{Confid}}$), which is then sent to the bank confidentially. Only if the bank receives both the correct TAN and PIN, it considers the transaction to be confirmed. Now, if only the initiator's smartphone is corrupted, the adversary is able to present false transaction data to them or even to perform the confirmation himself. However, this would be noticed immediately, since the transaction data shown on the

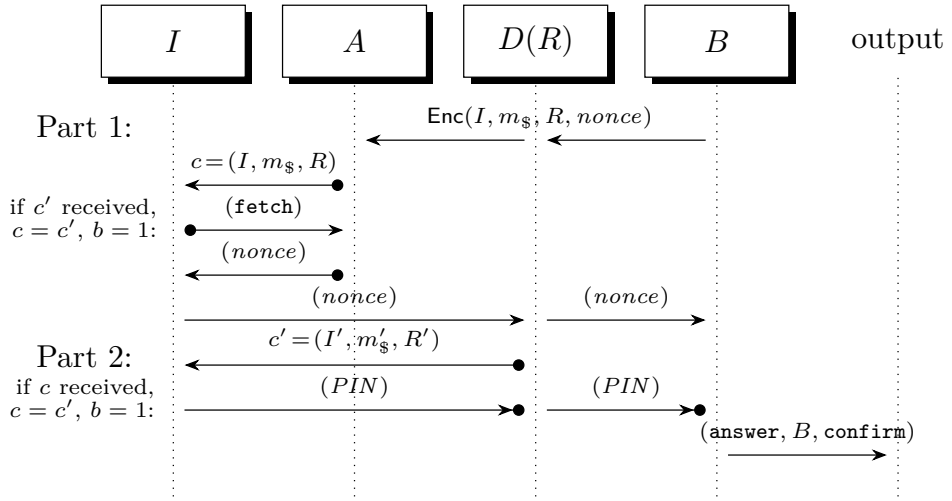


Figure 17.1.: Main phase of $\pi_{\text{L-Conf}}$ realizing $\mathcal{F}_{\text{CONF},\mathcal{D}}(B, I)$ using authenticated and confidential channels drawn as $\bullet\!\!\rightarrow$ and $\rightarrow\!\!\bullet$, respectively. The protocol is between the human initiator I, his personal device A, the ATM $D(R)$ and the bank B. The bit $b \in \{0, 1\}$ indicates, whether I wants to confirm, hence $(nonce)$ and (PIN) are only sent in this case.

input device would be wrong and the initiator would not enter his PIN. Conversely, if only the input device is malicious and displays wrong transaction data, the initiator will notice this using their smartphone.

Theorem 17.1. *Let I, B, $D(R)$ and A be ITMs, where I is human. Let S be the domain of $\mathcal{D}_1$ and $\mathcal{D}_2$, let π_1 UC-realize $\mathcal{F}_{\text{CONF},\mathcal{D}_1}(B, I)$ if A is honest and let π_2 UC-realize $\mathcal{F}_{\text{CONF},\mathcal{D}_2}(B, I)$ if $D(R)$ is honest. Then, $\pi_{\text{L-Conf}}$ UC-realizes $\mathcal{F}_{\text{CONF},\mathcal{D}_3}(B, I)$ in the $\mathcal{F}_{\text{AUTH}}(A, I)$, $\mathcal{F}_{\text{AUTH}}(I, A)$, $\mathcal{F}_{\text{AUTH}}(D(R), I)$, $\mathcal{F}_{\text{Confid}}(I, D(R))$, $\mathcal{F}_{\text{Confid}}(D(R), B)$-hybrid model where for all $x \in S$:*

$$\mathcal{D}_3(m_\$)(x) := \begin{cases} \max(\mathcal{D}_1(m_\$)(\texttt{confirm}), \mathcal{D}_2(m_\$)(\texttt{confirm})), & x = \texttt{confirm}, \\ 1 - \max(\mathcal{D}_1(m_\$)(\texttt{confirm}), \mathcal{D}_2(m_\$)(\texttt{confirm})), & x = \texttt{reject}. \end{cases}$$

Proof Sketch. The protocol $\pi_{\text{L-Conf}}$ (Figure 17.1) can be interpreted as the composition of two confirmation protocols π_1 (Part 1) and π_2 (Part 2) UC-realizing $\mathcal{F}_{\text{CONF},\mathcal{D}_1}(B, I)$ if A is honest, and $\mathcal{F}_{\text{CONF},\mathcal{D}_2}(B, I)$ if $D(R)$ is honest, respectively (omitting the message from B to $D(R)$ to initiate Part 2). Let $b \in \{\texttt{confirm}, \texttt{reject}\}$ denote the initiator's input and let $b_1, b_2 \in \{\texttt{confirm}, \texttt{reject}\}$ denote the outputs of π_1 and π_2 as received by B, respectively. After having received b_1 and b_2, B outputs b', which is `confirm` if $b_1 = b_2 =$ `confirm`, and `reject` otherwise. By definition, $b' =$ `confirm` while $b =$ `reject` holds with probability upper-bounded by $\max(\mathcal{D}_1(m_\$)(\texttt{confirm}), \mathcal{D}_2(m_\$)(\texttt{confirm}))$. Thus, $\pi_{\text{L-Conf}}$ UC-realizes $\mathcal{F}_{\text{CONF},\mathcal{D}_3}(B, I)$ with one-out-of-two security relative to the assumptions that A or $D(R)$ is honest, respectively. □

For the complete construction and proof, see the full version [AGH+19a].

Part IV.
Conclusion and Outlook

Conclusion

This chapter contains a conclusion of the thesis, and draws from the introductory and conclusion sections of all the papers used in this thesis. We proceed in the inverse order of the parts.

On Human-Friendly Secure Payment

Designing secure payment protocols poses a particular challenge. They typically involve a human user who is not capable of performing cryptographic operations and therefore needs an intermediate device (e.g., an ATM) to interface with the protocol, which can not always be trusted. In this work we introduce a formal model for the security of such protocols. In particular, we do not assume all intermediate devices to be trusted. We use the UC framework, guaranteeing strong security and composability even in concurrent and interleaved executions.

With our model, we develop a set of basic requirements for electronic payment protocols without which no protocol can be considered secure. Based on these results, we discuss different current payment protocols and find that most do not realize these requirements. We then specify a protocol called L-Pay (based upon chipTAN and photoTAN), which uses an additional smartphone or TAN generator as a user's device and which is secure in our model even if either the ATM or the user's device is honest.

On Cryptography with a Deck of Cards

We believe that our work on card-based cryptography may well be described as transformative for the field. Among other, we

- discuss variants of the computational model and propose a new way of speaking about protocols via state diagrams, which allow to act as witness from which security and correctness can directly be read,
- identify a suitable security notion which allows to capture committed format and non-committed format protocols in the same framework.
- construct card-minimal AND protocol using four cards, both in the two-color and in the standard deck setting with non-closed (or non-uniform) and uniform closed shuffles, respectively. We also give a five-card AND protocol exhibiting a finite running time behavior in the two-color deck case.

Table 17.1.: Minimum number of cards required by committed format AND and n-COPY protocols, subject to the requirements specified in the first two columns. The second column specifies shuffle restrictions. See also Figure 9.7.

Running Time	**Shuffle Restr.**	**#Cards**	**Protocol**	**Lower Bound**
AND Protocols:				
restart-free LV	closed	4	Theorem 8.1	– (trivial)
restart-free LV	uniform	4	Theorem 8.2	– (trivial)
restarting LV	uniform closed	5	[AHMS18, Sect. 2]	Theorem 9.6
restart-free LV	uniform closed			
finite	–	5	Theorem 8.4	Theorem 9.1
finite	uniform			
finite	closed	6	[MS09, Sect. 3]	Theorem 9.5
finite	uniform closed			
COPY Protocols:				
restarting LV	–	$2n+1$	[NNH$^+$18, Sect. 5]	[KKW$^+$17, Sect. 8]
restart-free LV	uniform			
restarting LV	closed	$2n+2$	[MS09, Sect. 5]	Theorem 9.7, [KKW$^+$17, Sect. 9]
finite	–			
finite	uniform closed			

Table 17.2.: Comparison of protocols for arbitrary k-ary Boolean functions. In the terminology of [SM19], a *type-1 deck* is a deck in which all symbols are distinct, except from one symbol that may occur more than once and a type-2 deck contains at least two symbols at least twice. In both deck types no symbol should occur more that half the size of the deck. For the special case of symmetric functions, one can, e.g., reduce the size of the deck in [NHMS15] to $2k+2$.

Deck	**Running Time**	**Shuffles**	**Reference**
$k \cdot [\heartsuit, \clubsuit]$	restarting LV	uniform non-closed	Theorem 8.5
$(k+3) \cdot [\heartsuit, \clubsuit]$	finite	uniform closed	[NHMS15]
Type-2, $2k+6$ cards	finite	uniform closed	[SM19]
Type-1, $2k+7$ cards	finite	uniform closed	[SM19]
$[1, \ldots, 2k+8]$	finite	uniform closed	[SM19]

Table 17.3.: Minimum number of cards required by committed format AND and base conversion protocols for standard decks, subject to the requirements specified in the first two columns. The second column specifies shuffle restrictions. 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 8.6	– (trivial)
finite	–	$\geq 5^{a}, \leq 8$	[M16b, Sect. 3.4]	Theorem 9.9
finite	uniform closed			
Disjoint Base Convert Protocols:				
finite	uniform closed	4	[M16b, Sect. 3.2]	– (trivial)
Overlapping Base Convert Protocols:				
Las Vegas	random cuts	3	Theorem 8.7	– (trivial)
finite	–	5	Theorem 8.8	Theorem 9.8
finite	uniform closed			

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

- prove these and other protocols from the literature for AND, COPY and base convert as optimal for their respective parameters. This completes the landscape of tight lower bounds with all practicality restrictions we identified in the beginning, at least for the two-color deck case. For this, we extensively extend the toolset of proof techniques for impossibility results in card-based cryptography. As a corollary, we show that restarting in AND and COPY protocols cannot decrease the number of cards.
- additionally introduce a formal verification technique to card-based cryptography, which allows us to show a certain protocol to have a shortest run, w.r.t. the practicality restrictions that the protocol fulfills.
- construct four different protocols for private function evaluations allowing for some choice in the computational model that the function is specified in. These are conceptually relatively simple and easy to understand by its use of the introduced sorting protocol abstraction
- show how a passively secure protocol can be transformed into an actively secure version, given certain conditions. Additionally, we discuss the active security of several protocols with input awareness from the literature.

A survey of all card-minimal protocols with respect to all combinations of restrictions are given in Table 17.1. For a survey of protocols computing arbitrary Boolean functions, we refer the reader to Table 17.2. We have three key parameters in describing the properties

of protocols: the deck used (in particular its size), whether it is a finite-runtime or a Las Vegas algorithm, and which type of shuffles are used in the protocols. Finally, Table 17.3 summarizes our results for protocols with a standard deck.

Outlook and Future Work

For the case of *human-friendly electronic payment*, similar to [BRS15], future work could explore the generation of a complete set of possible channel topologies allowing for electronic payment in our model and with strong UC security guarantees. Also, one important security mechanism missing in our model is time (e.g., for arguing about the security of timestamps), which is impossible to model in the standard UC framework however. Extensions exist that model time [KMTZ13] which could be incorporated in our model in the future. Since we assume the bank to be trusted, we limited our model to a single bank and disregarded the problem of book-keeping. Fture work could expand our model to include these features.

For card-based cryptography, there are two main directions to explore: deriving tight lower bounds for standard deck protocols, or even for arbitrary decks, possibly refining our formal verification method to automatize the process. Moreover, the case of protocols with input awareness, as discussed in Section 12.10 is not yet developed much, with no lower bounds on the number of cards yet. Finally, it would be nice to have a rigorous didactic study to back up anecdotal evidence of its usefulness in classroom scenarios.

Bibliography

[A] R. Abbott. *Eleusis and Eleusis Express*. URL: http://www.logicmazes.com/games/eleusis/ (visited on Nov. 9, 2018).

[AAA+05] M. H. Albert, R. E. L. Aldred, M. D. Atkinson, H. P. van Ditmarsch, and C. C. Handley. "Safe communication for card players by combinatorial designs for two-step protocols". In: *Australasian Journal of Combinatorics* 33 (2005), pp. 33–46. URL: https://ajc.maths.uq.edu.au/pdf/33/ajc_v33_p033.pdf.

[ABC+12] R. J. Anderson, M. Bond, O. Choudary, S. J. Murdoch, and F. Stajano. "Might Financial Cryptography Kill Financial Innovation? – The Curious Case of EMV". In: *Financial Cryptography and Data Security. 15th International Conference*. Revised Selected Papers: FC 2011 (Gros Islet, St. Lucia, Feb. 28–Mar. 4, 2011). Ed. by G. Danezis. LNCS 7035. Springer, 2012, pp. 220–234. DOI: 10.1007/978-3-642-27576-0_18.

[ABL+17] D. Achenbach, A. Borcherding, B. Löwe, J. Müller-Quade, and J. Rill. "Towards Realising Oblivious Voting". In: *E-Business and Telecommunications. 13th International Joint Conference*. Revised Selected Papers: ICETE 2016 (Lisbon, Portugal, July 16–28, 2016). Ed. by M. S. Obaidat. CCIS 764. Springer, 2017, pp. 216–240. DOI: 10.1007/978-3-319-67876-4_11.

[AGH+19a] D. Achenbach, R. Gröll, T. Hackenjos, A. Koch, B. Löwe, J. Mechler, J. Müller-Quade, and J. Rill. *Your Money or Your Life—Modeling and Analyzing the Security of Electronic Payment in the UC Framework*. Full version of the paper. 2019. URL: https://crypto.iti.kit.edu/fileadmin/User/Mechler/AGHKLMMQR19.pdf.

[AGH+19b] D. Achenbach, R. Gröll, T. Hackenjos, A. Koch, B. Löwe, J. Mechler, J. Müller-Quade, and J. Rill. "Your Money or Your Life—Modeling and Analyzing the Security of Electronic Payment in the UC Framework". In: *Financial Cryptography and Data Security. 23rd International Conference*. Revised Selected Papers: FC 2019 (Frigate Bay, Saint Kitts and Nevis, Feb. 18–22, 2019). Ed. by I. Goldberg and T. Moore. LNCS. Springer, 2019. Cryptology ePrint Archive, Report 2019/924. In press.

[AHMS18] Y. Abe, Y. Hayashi, T. Mizuki, and H. Sone. "Five-Card AND Protocol in Committed Format Using Only Practical Shuffles". In: *Proceedings of the 5th ACM ASIA Public-Key Cryptography Workshop*. APKC 2018 (Incheon, Republic of Korea, June 4, 2018). Ed. by K. Emura, J. H. Seo, and Y. Watanabe. ACM, 2018, pp. 3–8. DOI: 10.1145/3197507.3197510.

[ALMR16] D. Achenbach, B. Löwe, J. Müller-Quade, and J. Rill. "Oblivious Voting: Hiding Votes from the Voting Machine in Bingo Voting". In: *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications*. Volume 4: 13th International Conference on Security and Cryptography, SECRYPT 2016 (Lisbon, Portugal, July 16–28, 2016). Ed. by C. Callegari, M. van Sinderen, P. G. Sarigiannidis, P. Samarati, E. Cabello, P. Lorenz, and M. S. Obaidat. SciTePress, 2016, pp. 85–96. DOI: 10.5220/0005964300850096.

[AMR15] D. Achenbach, J. Müller-Quade, and J. Rill. "Universally Composable Firewall Architectures Using Trusted Hardware". In: *Cryptography and Information Security in the Balkans. 1st International Conference*. Revised Selected Papers: BalkanCryptSec 2014 (Istanbul, Turkey, Oct. 16–17, 2014). Ed. by B. Ors and B. Preneel. LNCS 9024. Springer, 2015, pp. 57–74. DOI: 10.1007/978-3-319-21356-9_5.

[AMSY16] F. Armknecht, D. Moriyama, A. Sadeghi, and M. Yung. "Towards a Unified Security Model for Physically Unclonable Functions". In: *Topics in Cryptology – CT-RSA 2016*. Proceedings: The Cryptographers' Track at the RSA Conference (San Francisco, CA, USA, Feb. 29–Mar. 4, 2016). Ed. by K. Sako. LNCS 9610. Springer, 2016, pp. 271–287. DOI: 10.1007/978-3-319-29485-8_16.

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

[B12] B. Blanchet. "Security Protocol Verification: Symbolic and Computational Models". In: *Principles of Security and Trust. 1st International Conference*. Proceedings: POST 2012 (Tallinn, Estonia, Mar. 24–Apr. 1, 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.

[B18] Borchert IT-Sicherheit UG. *Display-TAN Mobile Banking: Secure and Mobile*. 2018. URL: http://www.display-tan.com/ (visited on Jan. 5, 2019).

[B19] B. Broadnax. "New Frameworks for Concurrently Composable Multi-Party Computation". PhD thesis. Karlsruhe Institute of Technology (KIT), 2019. 161 pp. DOI: 10.5445/IR/1000091915.

[B89] D. A. M. Barrington. "Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1". In: *Journal of Computer and System Sciences* 38.1 (1989), pp. 150–164. DOI: 10.1016/0022-0000(89)90037-8.

[BBKL17] O. Biçer, M. A. Bingöl, M. S. Kiraz, and A. Levi. *Towards Practical PFE: An Efficient 2-Party Private Function Evaluation Protocol Based on Half Gates*. 2017. Cryptology ePrint Archive, Report 2017/415.

[BCD+18] D. Basin, C. Cremers, J. Dreier, S. Meier, R. Sasse, and B. Schmidt. *Tamarin prover*. 2018. URL: https://tamarin-prover.github.io/ (visited on July 30, 2019).

[BCIK03] J. Balogh, J. A. Csirik, Y. Ishai, and E. Kushilevitz. "Private computation using a PEZ dispenser". In: *Theoretical Computer Science* 306.1–3 (2003), pp. 69–84. DOI: 10.1016/S0304-3975(03)00210-X.

[BCM+14] M. Bond, O. Choudary, S. J. Murdoch, S. P. Skorobogatov, and R. J. Anderson. "Chip and Skim: Cloning EMV Cards with the Pre-play Attack". In: *2014 IEEE Symposium on Security and Privacy*. Proceedings: S&P 2014 (Berkeley, CA, USA, May 18–21, 2014). IEEE Computer Society, 2014, pp. 49–64. DOI: 10.1109/SP.2014.11.

[BFSK11] C. Brzuska, M. Fischlin, H. Schröder, and S. Katzenbeisser. "Physically Uncloneable Functions in the Universal Composition Framework". In: *Advances in Cryptology – CRYPTO 2011*. Proceedings: 31st Annual Cryptology Conference (Santa Barbara, CA, USA, Aug. 14–18, 2011). Ed. by P. Rogaway. LNCS 6841. Springer, 2011, pp. 51–70. DOI: 10.1007/978-3-642-22792-9_4.

[BGI+01] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. "On the (Im)possibility of Obfuscating Programs". In: *Advances in Cryptology – CRYPTO 2001*. Proceedings: 21st Annual International Cryptology Conference (Santa Barbara, CA, USA, Aug. 19–23, 2001). Ed. by J. Kilian. LNCS 2139. Springer, 2001, pp. 1–18. DOI: 10.1007/3-540-44647-8_1.

[BGLS03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps". In: *Advances in Cryptology – EUROCRYPT 2003*. Proceedings: International Conference on the Theory and Applications of Cryptographic Techniques (Warsaw, Poland, May 4–8, 2003). Ed. by E. Biham. LNCS 2656. Springer, 2003, pp. 416–432. DOI: 10.1007/3-540-39200-9_26.

[BKM+18] B. Broadnax, A. Koch, J. Mechler, T. Müller, J. Müller-Quade, and M. Nagel. *Fortified Universal Composability: Taking Advantage of Simple Secure Hardware Modules*. 2018. Cryptology ePrint Archive, Report 2018/519. In submission.

[BRS15] D. A. Basin, S. Radomirovic, and M. Schläpfer. "A Complete Characterization of Secure Human-Server Communication". In: *IEEE 28th Computer Security Foundations Symposium*. Proceedings: CSF 2015 (Verona, Italy, July 13–17, 2015). Ed. by C. Fournet, M. W. Hicks, and L. Viganò. IEEE Computer Society, 2015, pp. 199–213. DOI: 10.1109/CSF.2015.21.

[C01] R. Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *42nd Annual Symposium on Foundations of Computer Science*. Proceedings: FOCS 2001 (Las Vegas, Nevada, USA, Oct. 14–17, 2001). IEEE Computer Society, 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[C13] Commerzbank AG. *Das photoTAN-Lesegerät*. 2013. URL: https://www.commerzbank.de/portal/media/a-30-sonstige-medien/pdf/themen/sicherheit-1/Flyer_Lesegeraet.pdf (visited on July 16, 2019).

[C18] Commonwealth Bank of Australia. *Cardless Cash*. 2018. URL: https://www.commbank.com.au/digital-banking/cardless-cash.html (visited on Dec. 14, 2018).

[CCC+09] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, A. T. Sherman, and P. L. Vora. "Scantegrity II: End-to-End Verifiability by Voters of Optical Scan Elections Through Confirmation Codes". In: *IEEE Transactions on Information Forensics and Security* 4.4 (2009), pp. 611–627. DOI: 10.1109/TIFS.2009.2034919.

[CCC+10] R. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Herrnson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P. L. Vora. "Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy". In: *Proceedings of the 19th USENIX Security Symposium* (Washington, DC, USA, Aug. 11–13, 2010). USENIX Association, 2010, pp. 291–306. URL: http://www.usenix.org/events/sec10/tech/full_papers/Carback.pdf.

[CDPW07] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. "Universally Composable Security with Global Setup". In: *Theory of Cryptography. 4th Theory of Cryptography Conference*. Proceedings: TCC 2007 (Amsterdam, The Netherlands, Feb. 21–24, 2007). Ed. by S. P. Vadhan. LNCS 4392. Springer, 2007, pp. 61–85. DOI: 10.1007/978-3-540-70936-7_4.

[CF01] R. Canetti and M. Fischlin. "Universally Composable Commitments". In: *Advances in Cryptology – CRYPTO 2001*. Proceedings: 21st Annual International Cryptology Conference (Santa Barbara, CA, USA, Aug. 19–23, 2001). Ed. by J. Kilian. LNCS 2139. Springer, 2001, pp. 19–40. DOI: 10.1007/3-540-44647-8_2.

[CFF+17] V. Cortier, A. Filipiak, J. Florent, S. Gharout, and J. Traoré. "Designing and Proving an EMV-Compliant Payment Protocol for Mobile Devices". In: *2017 IEEE European Symposium on Security and Privacy*. Proceedings: EuroS&P 2017 (Paris, France, Apr. 26–28, 2017). IEEE, 2017, pp. 467–480. DOI: 10.1109/EuroSP.2017.19.

[CFN90] D. Chaum, A. Fiat, and M. Naor. "Untraceable Electronic Cash". In: *Advances in Cryptology – CRYPTO '88*. Proceedings: 8th Annual International Cryptology Conference (Santa Barbara, CA, USA, Aug. 21–25, 1988). Ed. by S. Goldwasser. LNCS 403. Springer, 1990, pp. 319–327. DOI: 10.1007/0-387-34799-2_25.

[CGdR+15] T. Chothia, F. D. Garcia, J. de Ruiter, J. van den Breekel, and M. Thompson. "Relay Cost Bounding for Contactless EMV Payments". In: *Financial Cryptography and Data Security. 19th International Conference*. Revised Selected Papers: FC 2015 (San Juan, Puerto Rico, Jan. 26–30, 2015). Ed. by R. Böhme and T. Okamoto. LNCS 8975. Springer, 2015, pp. 189–206. DOI: 10.1007/978-3-662-47854-7_11.

[CHL13] E. Cheung, C. Hawthorne, and P. Lee. *CS 758 Project: Secure Computation with Playing Cards*. 2013. URL: https://cdchawthorne.com/writings/secure_playing_cards.pdf (visited on Aug. 23, 2019).

[CK02] R. Canetti and H. Krawczyk. "Universally Composable Notions of Key Exchange and Secure Channels". In: *Advances in Cryptology – EUROCRYPT 2002*. Proceedings: International Conference on the Theory and Applications of Cryptographic Techniques (Amsterdam, The Netherlands, Apr. 28–May 2, 2002). Ed. by L. R. Knudsen. LNCS 2332. Springer, 2002, pp. 337–351. DOI: 10.1007/3-540-46035-7_22.

[CK94] C. Crépeau and J. Kilian. "Discreet Solitary Games". In: *Advances in Cryptology – CRYPTO '93*. Proceedings: 13th Annual International Cryptology Conference (Santa Barbara, CA, USA, Aug. 22–26, 1993). Ed. by D. R. Stinson. LNCS 773. Springer, 1994, pp. 319–330. DOI: 10.1007/3-540-48329-2_27.

[CKL03] R. Canetti, E. Kushilevitz, and Y. Lindell. "On the Limitations of Universally Composable Two-Party Computation without Set-up Assumptions". In: *Advances in Cryptology – EUROCRYPT 2003*. Proceedings: International Conference on the Theory and Applications of Cryptographic Techniques (Warsaw, Poland, May 4–8, 2003). Ed. by E. Biham. LNCS 2656. Springer, 2003, pp. 68–86. DOI: 10.1007/3-540-39200-9_5.

[CKL04] E. M. Clarke, D. Kroening, and F. Lerda. "A Tool for Checking ANSI-C Programs". In: *Tools and Algorithms for the Construction and Analysis of Systems. 10th International Conference*. Proceedings: TACAS 2004 (Barcelona, Spain, Mar. 29–Apr. 2, 2004). Ed. by K. Jensen and A. Podelski. LNCS 2988. Springer, 2004, pp. 168–176. DOI: 10.1007/978-3-540-24730-2_15.

[CV12] R. Canetti and M. Vald. "Universally Composable Security with Local Adversaries". In: *Security and Cryptography for Networks. 8th International Conference*. Proceedings: SCN 2012 (Amalfi, Italy, Sept. 5–7, 2012). Ed. by I. Visconti and R. D. Prisco. LNCS 7485. Springer, 2012, pp. 281–301. DOI: 10.1007/978-3-642-32928-9_16.

[D15] R. Durham. *Skipjack*. Ed. by StevenGalbraith's Games. 2015. URL: https://www.thegamecrafter.com/games/skipjack (visited on Aug. 23, 2019). AsiaCrypt2015 edition.

[D18] Deutsche Bank. *photoTAN – schnell und einfach aktiviert.* 2018. URL: https://www.deutsche-bank.de/content/dam/deutschebank/de/shared/pdf/Photo_TAN_Smartphone_2.pdf (visited on Aug. 23, 2019).

[dB90] B. den Boer. "More Efficient Match-Making and Satisfiability: The Five Card Trick". In: *Advances in Cryptology – EUROCRYPT '89*. Proceedings: Workshop on the Theory and Application of Cryptographic Techniques (Houthalen, Belgium, Apr. 10–13, 1989). Ed. by J. Quisquater and J. Vandewalle. LNCS 434. Springer, 1990, pp. 208–217. DOI: 10.1007/3-540-46885-4_23.

[DBR16] M. Denzel, A. Bruni, and M. D. Ryan. "Smart-Guard: Defending User Input from Malware". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*. 13th IEEE International Conference on Ubiquitous Intelligence and Computing, UIC 2016 (Toulouse, France, July 18–21, 2016). IEEE Computer Society, 2016, pp. 502–509. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0089.

[DFK+14] D. Dachman-Soled, N. Fleischhacker, J. Katz, A. Lysyanskaya, and D. Schröder. "Feasibility and Infeasibility of Secure Computation with Malicious PUFs". In: *Advances in Cryptology – CRYPTO 2014*. Proceedings, Part II: 34th Annual Cryptology Conference (Santa Barbara, CA, USA, Aug. 17–21, 2014). Ed. by J. A. Garay and R. Gennaro. LNCS 8617. Springer, 2014, pp. 405–420. DOI: 10.1007/978-3-662-44381-1_23.

[DKM11] N. Döttling, D. Kraschewski, and J. Müller-Quade. "Unconditional and Composable Security Using a Single Stateful Tamper-Proof Hardware Token". In: *Theory of Cryptography. 8th Theory of Cryptography Conference*. Proceedings: TCC 2011 (Providence, RI, USA, Mar. 28–30, 2011). Ed. by Y. Ishai. LNCS 6597. Springer, 2011, pp. 164–181. DOI: 10.1007/978-3-642-19571-6_11.

[DKM12] N. Döttling, D. Kraschewski, and J. Müller-Quade. *David & Goliath Oblivious Affine Function Evaluation. Asymptotically Optimal Building Blocks for Universally Composable Two-Party Computation from a Single Untrusted Stateful Tamper-Proof Hardware Token*. 2012. Cryptology ePrint Archive, Report 2012/135.

[DKMN15a] N. Döttling, D. Kraschewski, J. Müller-Quade, and T. Nilges. "From Stateful Hardware to Resettable Hardware Using Symmetric Assumptions". In: *Provable Security. 9th International Conference*. Proceedings: ProvSec 2015 (Kanazawa, Japan, Nov. 24–26, 2015). Ed. by M. H. Au and A. Miyaji. LNCS 9451. Springer, 2015, pp. 23–42. DOI: 10.1007/978-3-319-26059-4_2.

[DKMN15b] N. Döttling, D. Kraschewski, J. Müller-Quade, and T. Nilges. "General Statistically Secure Computation with Bounded-Resettable Hardware Tokens". In: *Theory of Cryptography. 12th Theory of Cryptography Conference*. Proceedings, Part I: TCC 2015 (Warsaw, Poland, Mar. 23–25, 2015). Ed. by Y. Dodis and J. B. Nielsen. LNCS 9014. Springer, 2015, pp. 319–344. DOI: 10.1007/978-3-662-46494-6_14.

[DLP+12] J. P. Degabriele, A. Lehmann, K. G. Paterson, N. P. Smart, and M. Strefler. "On the Joint Security of Encryption and Signature in EMV". In: *Topics in Cryptology – CT-RSA 2012*. Proceedings: The Cryptographers' Track at the RSA Conference 2012 (San Francisco, CA, USA, Feb. 27–Mar. 2, 2012). Ed. by O. Dunkelman. LNCS 7178. Springer, 2012, pp. 116–135. DOI: 10.1007/978-3-642-27954-6_8.

[DM00] Y. Dodis and S. Micali. "Parallel Reducibility for Information-Theoretically Secure Computation". In: *Advances in Cryptology – CRYPTO 2000*. Proceedings: 20th Annual International Cryptology Conference (Santa Barbara, CA, USA, Aug. 20–24, 2000). Ed. by M. Bellare. LNCS 1880. Springer, 2000, pp. 74–92. DOI: 10.1007/3-540-44598-6_5.

[DM07] S. Drimer and S. J. Murdoch. "Keep Your Enemies Close: Distance Bounding Against Smartcard Relay Attacks". In: *Proceedings of the 16th USENIX Security Symposium* (Boston, MA, USA, Aug. 6–10, 2007). Ed. by N. Provos. USENIX Association, 2007. URL: https://www.usenix.org/conference/16th-usenix-security-symposium/keep-your-enemies-close-distance-bounding-against.

[DM96] J. D. Dixon and B. Mortimer. *Permutation groups*. Graduate Texts in Mathematics 163. New York: Springer-Verlag, 1996. DOI: 10.1007/978-1-4612-0731-3.

[DMMN13] N. Döttling, T. Mie, J. Müller-Quade, and T. Nilges. "Implementing Resettable UC-Functionalities with Untrusted Tamper-Proof Hardware-Tokens". In: *Theory of Cryptography. 10th Theory of Cryptography Conference*. Proceedings: TCC 2013 (Tokyo, Japan, Mar. 3–6, 2013). Ed. by A. Sahai. LNCS 7785. Springer, 2013, pp. 642–661. DOI: 10.1007/978-3-642-36594-2_36.

[DS13] I. Damgård and A. Scafuro. "Unconditionally Secure and Universally Composable Commitments from Physical Assumptions". In: *Advances in Cryptology – ASIACRYPT 2013*. Proceedings, Part II: 19th International Conference on the Theory and Application of Cryptology and Information Security (Bengaluru, India, Dec. 1–5, 2013). Ed. by K. Sako and P. Sarkar. LNCS 8270. Springer, 2013, pp. 100–119. DOI: 10.1007/978-3-642-42045-0_6.

[E11a] EMVCo. *EMV Integrated Circuit Card Specifications for Payment Systems*. Vol. 1: *Application Independent ICC to Terminal Interface Requirements*. 4 vols. Version 4.3. 2011. URL: https://www.emvco.com/wp-content/uploads/documents/EMV_v4.3_Book_1_ICC_to_Terminal_Interface_2012060705394541.pdf.

[E11b] EMVCo. *EMV Integrated Circuit Card Specifications for Payment Systems*. Vol. 2: *Security and Key Management*. 4 vols. Version 4.3. 2011. URL: https://www.emvco.com/wp-content/uploads/2017/05/EMV_v4.3_Book_2_Security_and_Key_Management_20120607061923900.pdf.

[E11c] EMVCo. *EMV Integrated Circuit Card Specifications for Payment Systems*. Vol. 3: *Application Specification*. 4 vols. Version 4.3. 2011. URL: https://www.emvco.com/wp-content/uploads/2017/05/EMV_v4.3_Book_3_Application_Specification_20120607062110791.pdf.

[EAF+14] M. Emms, B. Arief, L. Freitas, J. Hannon, and A. P. A. van Moorsel. "Harvesting High Value Foreign Currency Transactions from EMV Contactless Credit Cards Without the PIN". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS 2014 (Scottsdale, AZ, USA, Nov. 3–7, 2014). Ed. by G. Ahn, M. Yung, and N. Li. ACM, 2014, pp. 716–726. DOI: 10.1145/2660267.2660312.

[ES04] N. Eén and N. Sörensson. "An Extensible SAT-solver". In: *Theory and Applications of Satisfiability Testing. 6th International Conference*. Selected Revised Papers: SAT 2003 (Santa Margherita Ligure, Italy, May 5–8, 2003). Ed. by E. Giunchiglia and A. Tacchella. LNCS 2919. Springer, 2004, pp. 502–518. DOI: 10.1007/978-3-540-24605-3_37.

[FAN+17] D. Francis, S. R. Aljunid, T. Nishida, Y. Hayashi, T. Mizuki, and H. Sone. "Necessary and Sufficient Numbers of Cards for Securely Computing Two-Bit Output Functions". In: *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology*. Revised Selected Papers: 2nd International Conference on Cryptology in Malaysia (Kuala Lumpur, Malaysia, Dec. 1–2, 2016). Ed. by R. C. Phan and M. Yung. LNCS 10311. Springer, 2017, pp. 193–211. DOI: 10.1007/978-3-319-61273-7_10.

[FFN14] B. Fisch, D. Freund, and M. Naor. "Physical Zero-Knowledge Proofs of Physical Properties". In: *Advances in Cryptology – CRYPTO 2014*. Proceedings, Part II: 34th Annual Cryptology Conference (Santa Barbara, CA, USA, Aug. 17–21, 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.

[FFN15] B. A. Fisch, D. Freund, and M. Naor. "Secure Physical Computation Using Disposable Circuits". In: *Theory of Cryptography. 12th Theory of Cryptography Conference*. Proceedings, Part I: TCC 2015 (Warsaw, Poland, Mar. 23–25, 2015). Ed. by Y. Dodis and J. B. Nielsen. LNCS 9014. Springer, 2015, pp. 182–198. DOI: 10.1007/978-3-662-46494-6_9.

[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: *Compiler Construction. 23rd International Conference*. Proceedings: CC 2014 (Grenoble, France, Apr. 5–13, 2014). Ed. by A. Cohen. LNCS 8409. Springer, 2014, pp. 244–249. DOI: 10.1007/978-3-642-54807-9_15.

[FNW96] R. Fagin, M. Naor, and P. Winkler. "Comparing Information Without Leaking It". In: *Communications of the ACM* 39.5 (1996), pp. 77–85. DOI: 10.1145/229459.229469.

[FVBG17] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov. "IRON: Functional Encryption using Intel SGX". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS 2017 (Dallas, TX, USA, Oct. 30–Nov. 3, 2017). Ed. by B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu. ACM, 2017, pp. 765–782. DOI: 10.1145/3133956.3134106.

[FW96] M. J. Fischer and R. N. Wright. "Bounds on Secret Key Exchange Using a Random Deal of Cards". In: *Journal of Cryptology* 9.2 (1996), pp. 71–99. DOI: 10.1007/BF00190803.

[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.

[GGH+13] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits". In: *54th Annual IEEE Symposium on Foundations of Computer Science*. Proceedings: FOCS 2013 (Berkeley, CA, USA, Oct. 26–29, 2013). IEEE Computer Society, 2013, pp. 40–49. DOI: 10.1109/FOCS.2013.13.

[GIS+10] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. "Founding Cryptography on Tamper-Proof Hardware Tokens". In: *Theory of Cryptography. 7th Theory of Cryptography Conference*. Proceedings: TCC 2010 (Zurich, Switzerland, Feb. 9–11, 2010). Ed. by D. Micciancio. LNCS 5978. Springer, 2010, pp. 308–326. DOI: 10.1007/978-3-642-11799-2_19.

[GKL15] J. A. Garay, A. Kiayias, and N. Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: *Advances in Cryptology – EUROCRYPT 2015*. Proceedings, Part II: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (Sofia, Bulgaria, Apr. 26–30, 2015). Ed. by E. Oswald and M. Fischlin. LNCS 9057. Springer, 2015, pp. 281–310. DOI: 10.1007/978-3-662-46803-6_10.

[GKS17] D. Günther, Á. Kiss, and T. Schneider. "More Efficient Universal Circuit Constructions". In: *Advances in Cryptology – ASIACRYPT 2017*. Proceedings, Part II: 23rd International Conference on the Theory and Applications of Cryptology and Information Security (Hong Kong, China,

Dec. 3–7, 2017). Ed. by T. Takagi and T. Peyrin. LNCS 10625. Springer, 2017, pp. 443–470. DOI: 10.1007/978-3-319-70697-9_16.

[GKW17] R. Goyal, V. Koppula, and B. Waters. "Lockable Obfuscation". In: *58th IEEE Annual Symposium on Foundations of Computer Science*. FOCS 2017 (Berkeley, CA, USA, Oct. 15–17, 2017). Ed. by C. Umans. IEEE Computer Society, 2017, pp. 612–621. DOI: 10.1109/FOCS.2017.62.

[GNPR07] R. Gradwohl, M. Naor, B. Pinkas, and G. N. Rothblum. "Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles". In: *Fun with Algorithms. 4th International Conference*. Proceedings: FUN 2007 (Castiglioncello, Italy, June 3–5, 2007). Ed. by P. Crescenzi, G. Prencipe, and G. Pucci. LNCS 4475. Springer, 2007, pp. 166–182. DOI: 10.1007/978-3-540-72914-3_16.

[HKK+16] G. Hartung, B. Kaidel, A. Koch, J. Koch, and A. Rupp. "Fault-Tolerant Aggregate Signatures". In: *Public-Key Cryptography – PKC 2016*. Proceedings, Part I: 19th IACR International Conference on Practice and Theory in Public-Key Cryptography (Taipei, Taiwan, Mar. 6–9, 2016). Ed. by C. Cheng, K. Chung, G. Persiano, and B. Yang. LNCS 9614. Springer, 2016, pp. 331–356. DOI: 10.1007/978-3-662-49384-7_13.

[HKK+17] G. Hartung, B. Kaidel, A. Koch, J. Koch, and D. Hartmann. "Practical and Robust Secure Logging from Fault-Tolerant Sequential Aggregate Signatures". In: *Provable Security. 11th International Conference*. Proceedings: ProvSec 2017 (Xi'an, China, Oct. 23–25, 2017). Ed. by T. Okamoto, Y. Yu, M. H. Au, and Y. Li. LNCS 10592. Springer, 2017, pp. 87–106. DOI: 10.1007/978-3-319-68637-0_6.

[HMU07] D. Hofheinz, J. Müller-Quade, and D. Unruh. "Universally Composable Zero-Knowledge Arguments and Commitments from Signature Cards". In: *Tatra Mountains Mathematical Publications* 37 (2007): *MORAVIACRYPT 2005*. Ed. by D. Cvrček, V. Matyáš, K. Nemoga, and Š. Porubský, pp. 93–103. URL: https://tatra.mat.savba.sk/paper.php?id_paper=887.

[HPV17] C. Hazay, A. Polychroniadou, and M. Venkitasubramaniam. "Constant Round Adaptively Secure Protocols in the Tamper-Proof Hardware Model". In: *Public-Key Cryptography – PKC 2017*. Proceedings, Part II: 20th IACR International Conference on Practice and Theory in Public-Key Cryptography (Amsterdam, The Netherlands, Mar. 28–31, 2017). Ed. by S. Fehr. LNCS 10175. Springer, 2017, pp. 428–460. DOI: 10.1007/978-3-662-54388-7_15.

[HSN+17] Y. Hashimoto, K. Shinagawa, K. Nuida, M. Inamura, and G. Hanaoka. "Secure Grouping Protocol Using a Deck of Cards". In: *Information Theoretic Security. 10th International Conference*. Proceedings: ICITS 2017 (Hong Kong, China, Nov. 29–Dec. 2, 2017). Ed. by J. Shikata. LNCS 10681. Springer, 2017, pp. 135–152. DOI: 10.1007/978-3-319-72089-0_8.

[ICM15] R. Ishikawa, E. Chida, and T. Mizuki. "Efficient Card-Based Protocols for Generating a Hidden Random Permutation Without Fixed Points". In: *Unconventional Computation and Natural Computation. 14th International Conference*. Proceedings: UCNC 2015 (Auckland, New Zealand, Aug. 30–Sept. 3, 2015). Ed. by C. S. Calude and M. J. Dinneen. LNCS 9252. Springer, 2015, pp. 215–226. DOI: 10.1007/978-3-319-21819-9_16.

[ILM08] S. Izmalkov, M. Lepinski, and S. Micali. "Verifiably Secure Devices". In: *Theory of Cryptography. 5th Theory of Cryptography Conference*. Proceedings: TCC 2008 (New York, USA, Mar. 19–21, 2008). Ed. by R. Canetti. LNCS 4948. Springer, 2008, pp. 273–301. DOI: 10.1007/978-3-540-78524-8_16.

[ILMas] S. Izmalkov, M. Lepinski, S. Micali, and abhi shelat. *Transparent Computation and Correlated Equilibrium*. URL: https://economics.mit.edu/files/1082.

[IML05] S. Izmalkov, S. Micali, and M. Lepinski. "Rational Secure Computation and Ideal Mechanism Design". In: *46th Annual IEEE Symposium on Foundations of Computer Science*. Proceedings: FOCS 2005 (Pittsburgh, PA, USA, Oct. 23–25, 2005). IEEE Computer Society, 2005, pp. 585–595. DOI: 10.1109/SFCS.2005.64.

[ISW03] Y. Ishai, A. Sahai, and D. A. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *Advances in Cryptology – CRYPTO 2003*. Proceedings: 23rd Annual International Cryptology Conference (Santa Barbara, CA, USA, Aug. 17–21, 2003). Ed. by D. Boneh. LNCS 2729. Springer, 2003, pp. 463–481. DOI: 10.1007/978-3-540-45146-4_27.

[K07] J. Katz. "Universally Composable Multi-party Computation Using Tamper-Proof Hardware". In: *Advances in Cryptology – EUROCRYPT 2007*. Proceedings: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques (Barcelona, Spain, May 20–24, 2007). Ed. by M. Naor. LNCS 4515. Springer, 2007, pp. 115–128. DOI: 10.1007/978-3-540-72540-4_7.

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

[KKR+12] S. Katzenbeisser, Ü. Koçabas, V. Rozic, A. Sadeghi, I. Verbauwhede, and C. Wachsmann. "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon". In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Proceedings: 14th International Workshop on Cryptographic Hardware and Embedded Systems (Leuven, Belgium, Sept. 9–12, 2012). Ed. by E. Prouff and P. Schaumont. LNCS 7428. Springer, 2012, pp. 283–301. DOI: 10.1007/978-3-642-33027-8_17.

[KKW+17] J. Kastner, A. Koch, S. Walzer, D. Miyahara, Y. Hayashi, T. Mizuki, and H. Sone. "The Minimum Number of Cards in Practical Card-Based Protocols". In: *Advances in Cryptology – ASIACRYPT 2017*. Proceedings, Part III: 23rd International Conference on the Theory and Applications of Cryptology and Information Security (Hong Kong, China, Dec. 3–7, 2017). Ed. by T. Takagi and T. Peyrin. LNCS 10626. Springer, 2017, pp. 126–155. DOI: 10.1007/978-3-319-70700-6_5.

[KMTZ13] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. "Universally Composable Synchronous Computation". In: *Theory of Cryptography. 10th Theory of Cryptography Conference*. Proceedings: TCC 2013 (Tokyo, Japan, Mar. 3–6, 2013). Ed. by A. Sahai. LNCS 7785. Springer, 2013, pp. 477–498. DOI: 10.1007/978-3-642-36594-2_27.

[KSK19] A. Koch, M. Schrempp, and M. Kirsten. "Card-based Cryptography Meets Formal Verification". In: *Advances in Cryptology – ASIACRYPT 2019*. Proceedings: 25th Annual International Conference on the Theory and Application of Cryptology and Information Security (Kobe, Japan, Dec. 8–12, 2019). Ed. by S. Galbraith and S. Moriai. LNCS. Springer, 2019. In press.

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

[KW18] A. Koch and S. Walzer. *Private Function Evaluation with Cards*. 2018. Cryptology ePrint Archive, Report 2018/1113. In submission.

[KWH15] A. Koch, S. Walzer, and K. Härtel. "Card-Based Cryptographic Protocols Using a Minimal Number of Cards". In: *Advances in Cryptology – ASIACRYPT 2015*. Proceedings, Part I: 21st International Conference on the Theory and Application of Cryptology and Information Security (Auckland, New Zealand, Nov. 29–Dec. 3, 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.

[LMRS04] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. "Sequential Aggregate Signatures from Trapdoor Permutations". In: *Advances in Cryptology – EUROCRYPT 2004*. Proceedings: International Conference on the Theory and Applications of Cryptographic Techniques (Interlaken, Switzerland, May 2–6, 2004). Ed. by C. Cachin and J. Camenisch. LNCS 3027. Springer, 2004, pp. 74–90. DOI: 10.1007/978-3-540-24676-3_5.

[LMS16] H. Lipmaa, P. Mohassel, and S. Sadeghian. *Valiant's Universal Circuit: Improvements, Implementation, and Applications*. 2016. Cryptology ePrint Archive, Report 2016/017.

[M16a] T. Mizuki. "Card-based protocols for securely computing the conjunction of multiple variables". In: *Theoretical Computer Science* 622 (2016), pp. 34–44. DOI: 10.1016/j.tcs.2016.01.039.

[M16b] T. Mizuki. "Efficient and Secure Multiparty Computations Using a Standard Deck of Playing Cards". In: *Cryptology and Network Security. 15th International Conference*. Proceedings: CANS 2016 (Milan, Italy, Nov. 14–16, 2016). Ed. by S. Foresti and G. Persiano. LNCS 10052. 2016, pp. 484–499. DOI: 10.1007/978-3-319-48965-0_29.

[MAS13] T. Mizuki, I. K. Asiedu, and H. Sone. "Voting with a Logarithmic Number of Cards". In: *Unconventional Computation and Natural Computation. 12th International Conference*. Proceedings: UCNC 2013 (Milan, Italy, July 1–5, 2013). Ed. by G. Mauri, A. Dennunzio, L. Manzoni, and A. E. Porreca. LNCS 7956. Springer, 2013, pp. 162–173. DOI: 10.1007/978-3-642-39074-6_16.

[MDAB10] S. J. Murdoch, S. Drimer, R. J. Anderson, and M. Bond. "Chip and PIN is Broken". In: *2010 IEEE Symposium on Security and Privacy*. Proceedings: S&P 2010 (Berleley/Oakland, CA, USA, May 16–19, 2010). IEEE Computer Society, 2010, pp. 433–446. DOI: 10.1109/SP.2010.33.

[MK18] T. Mizuki and Y. Komano. "Analysis of Information Leakage Due to Operative Errors in Card-Based Protocols". In: *Combinatorial Algorithms. 29th International Workshop*. Proceedings: IWOCA 2018 (Singapore, July 16–19, 2018). Ed. by C. S. Iliopoulos, H. W. Leong, and W. Sung. LNCS 10979. Springer, 2018, pp. 250–262. DOI: 10.1007/978-3-319-94667-2_21.

[MKS07a] T. Mizuki, Y. Kugimoto, and H. Sone. "Secure Multiparty Computations Using a Dial Lock". In: *Theory and Applications of Models of Computation. 4th International Conference*. Proceedings: TAMC 2007 (Shanghai, China, May 22–25, 2007). Ed. by J. Cai, S. B. Cooper, and H. Zhu. LNCS 4484. Springer, 2007, pp. 499–510. DOI: 10.1007/978-3-540-72504-6_44.

[MKS07b] T. Mizuki, Y. Kugimoto, and H. Sone. "Secure Multiparty Computations Using the 15 Puzzle". In: *Combinatorial Optimization and Applications. 1st International Conference*. Proceedings: COCOA 2007 (Xi'an, China, Aug. 14–16, 2007). Ed. by A. W. M. Dress, Y. Xu, and B. Zhu. LNCS 4616. Springer, 2007, pp. 255–266. DOI: 10.1007/978-3-540-73556-4_28.

[MKS12] T. Mizuki, M. Kumamoto, and H. Sone. "The Five-Card Trick Can Be Done with Four Cards". In: *Advances in Cryptology – ASIACRYPT 2012*. Proceedings: 18th International Conference on the Theory and Application of Cryptology and Information Security (Beijing, China, Dec. 2–6, 2012). Ed. by X. Wang and K. Sako. LNCS 7658. Springer, 2012, pp. 598–606. DOI: 10.1007/978-3-642-34961-4_36.

[MMN18] J. Mechler, J. Müller-Quade, and T. Nilges. "Reusing Tamper-Proof Hardware in UC-Secure Protocols". In: *Public-Key Cryptography – PKC 2018*. Proceedings, Part I: 21st IACR International Conference on Practice and Theory of Public-Key Cryptography (Rio de Janeiro, Brazil, Mar. 25–29, 2018). Ed. by M. Abdalla and R. Dahab. LNCS 10769. Springer, 2018, pp. 463–493. DOI: 10.1007/978-3-319-76578-5_16.

[MN06a] T. Moran and M. Naor. "Polling with Physical Envelopes: A Rigorous Analysis of a Human-Centric Protocol". In: *Advances in Cryptology – EUROCRYPT 2006*. Proceedings: 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques (St. Petersburg, Russia, May 28–June 1, 2006). Ed. by S. Vaudenay. LNCS 4004. Springer, 2006, pp. 88–108. DOI: 10.1007/11761679_7.

[MN06b] T. Moran and M. Naor. "Receipt-Free Universally-Verifiable Voting with Everlasting Privacy". In: *Advances in Cryptology – CRYPTO 2006*. Proceedings: 26th Annual International Cryptology Conference (Santa Barbara, CA, USA, Aug. 20–24, 2006). Ed. by C. Dwork. LNCS 4117. Springer, 2006, pp. 373–392. DOI: 10.1007/11818175_22.

[MN10a] T. Moran and M. Naor. "Basing cryptographic protocols on tamper-evident seals". In: *Theoretical Computer Science* 411.10 (2010): *ICALP 2005 – Track C: Security and Cryptography Foundations*. Ed. by M. Yung, pp. 1283–1310. DOI: 10.1016/j.tcs.2009.10.023.

[MN10b] T. Moran and M. Naor. "Split-ballot voting: Everlasting privacy with distributed trust". In: *ACM Transactions on Information and System Security* 13.2 (2010), 16:1–16:43. DOI: 10.1145/1698750.1698756.

[MNS07] T. Moran, M. Naor, and G. Segev. "Deterministic History-Independent Strategies for Storing Information on Write-Once Memories". In: *Automata, Languages and Programming. 34th International Colloquium*. Proceedings: ICALP 2007 (Wroclaw, Poland, July 9–13, 2007). Ed. by L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki. LNCS 4596. Springer, 2007, pp. 303–315. DOI: 10.1007/978-3-540-73420-8_28.

[MS09] T. Mizuki and H. Sone. "Six-Card Secure AND and Four-Card Secure XOR". In: *Frontiers in Algorithmics. 3rd International Workshop*. Proceedings: FAW 2009 (Hefei, China, June 20–23, 2009). Ed. by X. Deng, J. E. Hopcroft, and J. Xue. LNCS 5598. Springer, 2009, pp. 358–369. DOI: 10.1007/978-3-642-02270-8_36.

[MS13] P. Mohassel and S. S. Sadeghian. "How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation". In: *Advances in Cryptology – EUROCRYPT 2013*. Proceedings: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (Athens, Greece, May 26–30, 2013). Ed. by T. Johansson and P. Q. Nguyen. LNCS 7881. Springer, 2013, pp. 557–574. DOI: 10.1007/978-3-642-38348-9_33.

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

[MS14b] T. Mizuki and H. Shizuya. "Practical Card-Based Cryptography". In: *Fun with Algorithms. 7th International Conference*. Proceedings: FUN 2014 (Lipari Island, Sicily, Italy, July 1–3, 2014). Ed. by A. Ferro, F. Luccio, and P. Widmayer. LNCS 8496. Springer, 2014, pp. 313–324. DOI: 10.1007/978-3-319-07890-8_27.

[MS17] T. Mizuki and H. Shizuya. "Computational Model of Card-Based Cryptographic Protocols and Its Applications". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 100-A.1 (2017): *Special Section on Cryptography and Information Security*, pp. 3–11. DOI: 10.1587/transfun.E100.A.3.

[MSN99] T. Mizuki, H. Shizuya, and T. Nishizeki. "Dealing Necessary and Sufficient Numbers of Cards for Sharing a One-Bit Secret Key". In: *Advances in Cryptology – EUROCRYPT '99*. Proceedings: International Conference on the Theory and Application of Cryptographic Techniques (Prague, Czech Republic, May 2–6, 1999). Ed. by J. Stern. LNCS 1592. Springer, 1999, pp. 389–401. DOI: 10.1007/3-540-48910-X_27.

[MU07] J. Müller-Quade and D. Unruh. "Long-Term Security and Universal Composability". In: *Theory of Cryptography. 4th Theory of Cryptography Conference*. Proceedings: TCC 2007 (Amsterdam, The Netherlands, Feb. 21–24, 2007). Ed. by S. P. Vadhan. LNCS 4392. Springer, 2007, pp. 41–60. DOI: 10.1007/978-3-540-70936-7_3.

[MUH+18] D. Miyahara, I. Ueda, Y. Hayashi, T. Mizuki, and H. Sone. "Analyzing Execution Time of Card-Based Protocols". In: *Unconventional Computation and Natural Computation. 17th International Conference*. Proceedings: UCNC 2018 (Fontainebleau, France, June 25–29, 2018). Ed. by S. Stepney and S. Verlan. LNCS 10867. Springer, 2018, pp. 145–158. DOI: 10.1007/978-3-319-92435-9_11.

[MWS15] A. Marcedone, Z. Wen, and E. Shi. *Secure Dating with Four or Fewer Cards*. 2015. Cryptology ePrint Archive, Report 2015/1031.

[N15] T. Nilges. "The Cryptographic Strength of Tamper-Proof Hardware". PhD thesis. Karlsruhe Institute of Technology, 2015. URL: http://nbn-resolving.de/urn:nbn:de:swb:90-518099.

[NFR+17] K. Nayak, C. W. Fletcher, L. Ren, N. Chandran, S. V. Lokam, E. Shi, and V. Goyal. "HOP: Hardware makes Obfuscation Practical". In: *24th Annual Network and Distributed System Security Symposium*. Proceedings: NDSS 2017 (San Diego, CA, USA, Feb. 26–Mar. 1, 2017). The Internet Society, 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/hop-hardware-makes-obfuscation-practical/.

[NHMS15] T. Nishida, Y. Hayashi, T. Mizuki, and H. Sone. "Card-Based Protocols for Any Boolean Function". In: *Theory and Applications of Models of Computation. 12th Annual Conference*. Proceedings: TAMC 2015 (Singapore, May 18–20, 2015). Ed. by R. Jain, S. Jain, and F. Stephan. LNCS 9076. Springer, 2015, pp. 110–121. DOI: 10.1007/978-3-319-17142-5_11.

[NHMS16] A. Nishimura, Y.-i. Hayashi, T. Mizuki, and H. Sone. "An Implementation of Non-Uniform Shuffle for Secure Multi-Party Computation". In: *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*. AsiaPKC@AsiaCCS (Xi'an, China, May 30–June 3, 2016). Ed. by K. Emura, G. Hanaoka, and R. Zhang. ACM, 2016, pp. 49–55. DOI: 10.1145/2898420.2898425.

[NMS13] T. Nishida, T. Mizuki, and H. Sone. "Securely Computing the Three-Input Majority Function with Eight Cards". In: *Theory and Practice of Natural Computing. 2nd International Conference*. Proceedings: TPNC 2013 (Cáceres, Spain, Dec. 3–5, 2013). Ed. by A. Dediu, C. Martín-Vide, B. Truthe, and M. A. Vega-Rodríguez. LNCS 8273. Springer, 2013, pp. 193–204. DOI: 10.1007/978-3-642-45008-2_16.

[NNH+15] A. Nishimura, T. Nishida, Y. Hayashi, T. Mizuki, and H. Sone. "Five-Card Secure Computations Using Unequal Division Shuffle". In: *Theory and Practice of Natural Computing. 4th International Conference*. Proceedings: TPNC 2015 (Mieres, Spain, Dec. 15–16, 2015). Ed. by A. Dediu, L. Magdalena, and C. Martín-Vide. LNCS 9477. Springer, 2015, pp. 109–120. DOI: 10.1007/978-3-319-26841-5_9.

[NNH+18] A. Nishimura, T. Nishida, Y. Hayashi, T. Mizuki, and H. Sone. "Card-based protocols using unequal division shuffles". In: *Soft Computing. A Fusion of Foundations, Methodologies and Applications* 22.2 (2018): *Selected Papers of the Fourth International Conference on the Theory and Practice of Natural Computing, TPNC 2015*. Ed. by A.-H. Dediu and C. Martín-Vide, pp. 361–371. DOI: 10.1007/s00500-017-2858-2.

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

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

[NS95] M. Naor and A. Shamir. "Visual Cryptography". In: *Advances in Cryptology – EUROCRYPT '94*. Proceedings: Workshop on the Theory and Application of Cryptographic Techniques (Perugia, Italy, May 9–12, 1994). Ed. by A. D. Santis. LNCS 950. Springer, 1995, pp. 1–12. DOI: 10.1007/BFb0053419.

[NSIO17] T. Nakai, S. Shirouchi, M. Iwamoto, and K. Ohta. "Four Cards Are Sufficient for a Card-Based Three-Input Voting Protocol Utilizing Private Permutations". In: *Information Theoretic Security. 10th International Conference*. Proceedings: ICITS 2017 (Hong Kong, China, Nov. 29–Dec. 2, 2017). Ed. by J. Shikata. LNCS 10681. Springer, 2017, pp. 153–165. DOI: 10.1007/978-3-319-72089-0_9.

[NTM+16] T. Nakai, Y. Tokushige, Y. Misawa, M. Iwamoto, and K. Ohta. "Efficient Card-Based Cryptographic Protocols for Millionaires' Problem Utilizing Private Permutations". In: *Cryptology and Network Security. 15th International Conference*. Proceedings: CANS 2016 (Milan, Italy, Nov. 14–16, 2016). Ed. by S. Foresti and G. Persiano. LNCS 10052. 2016, pp. 500–517. DOI: 10.1007/978-3-319-48965-0_30.

[O81] Old Bailey Proceedings Online, ed. *Trial of JOHN BUCKLEY, THOMAS SHENTON*. Version 8.0. 1781. URL: https://www.oldbaileyonline.org/browse.jsp?div=t17810912-37 (visited on July 7, 2018).

[OSVW13] R. Ostrovsky, A. Scafuro, I. Visconti, and A. Wadia. "Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions". In: *Advances in Cryptology – EUROCRYPT 2013*. Proceedings: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (Athens, Greece, May 26–30, 2013). Ed. by T. Johansson and P. Q. Nguyen. LNCS 7881. Springer, 2013, pp. 702–718. DOI: 10.1007/978-3-642-38348-9_41.

[P18] Postbank. *Postbank chipTAN*. 2018. URL: https://www.postbank.de/privatkunden/chiptan.html (visited on Aug. 23, 2019).

[PH10] S. Popoveniuc and B. Hosp. "An Introduction to PunchScan". In: *Towards Trustworthy Elections. New Directions in Electronic Voting*. Ed. by D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. A. Ryan, J. Benaloh, M. Kutylowski, and B. Adida. LNCS 6000. Berlin, Heidelberg: Springer, 2010, pp. 242–259. DOI: 10.1007/978-3-642-12980-3_15.

[PRTG02] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld. "Physical One-Way Functions". In: *Science* 297.5589 (2002), pp. 2026–2030. DOI: 10.1126/science.1074376.

[PST17] R. Pass, E. Shi, and F. Tramèr. "Formal Abstractions for Attested Execution Secure Processors". In: *Advances in Cryptology – EUROCRYPT 2017*. Proceedings, Part I: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques (Paris, France, Apr. 30–May 4, 2017). Ed. by J. Coron and J. B. Nielsen. LNCS 10210. 2017, pp. 260–289. DOI: 10.1007/978-3-319-56620-7_10.

[R09] RedTeam Pentesting GmbH. *Man-in-the-Middle Attacks against the chipTAN comfort Online Banking System*. 2009. URL: https://www.redteam-pentesting.de/publications/2009-11-23-MitM-chipTAN-comfort_RedTeam-Pentesting_EN.pdf (visited on Apr. 11, 2019).

[RI19] S. Ruangwises and T. Itoh. "AND Protocols Using only Uniform Shuffles". In: *Computer Science – Theory and Applications*. Proceedings: 14th International Computer Science Symposium in Russia, CSR 2019 (Novosibirsk, Russia, July 1–5, 2019). Ed. by R. van Bevern and G. Kucherov. LNCS 11532. Springer, 2019, pp. 349–358. DOI: 10.1007/978-3-030-19955-5_30. arXiv-id: 1810.00769 [cs.CR].

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[RSH19] A. Rastogi, N. Swamy, and M. Hicks. "Wys*: A DSL for Verified Secure Multi-party Computations". In: *Principles of Security and Trust. 8th International Conference*. Proceedings: POST 2019 (Prague, Czech Republic, Apr. 6–11, 2019). Ed. by F. Nielson and D. Sands. LNCS 11426. Springer, 2019, pp. 99–122. DOI: 10.1007/978-3-030-17138-4_5.

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

[S16] Smart Card Alliance. *Contactless EMV Payments: Benefits for Consumers, Merchants and Issuers*. 2016. URL: http://www.emv-connection.com/downloads/2016/06/Contactless-2-0-WP-FINAL-June-2016.pdf (visited on May 17, 2018).

[S99] B. Schneier. *The Solitaire Encryption Algorithm*. Version 1.2. 1999. URL: https://www.schneier.com/academic/solitaire/ (visited on July 11, 2019).

[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: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL 2016 (St. Petersburg, FL, USA, Jan. 20–22, 2016). Ed. by R. Bodík and R. Majumdar. ACM, 2016, pp. 256–270. DOI: 10.1145/2837614.2837655.

[SM18] K. Shinagawa and T. Mizuki. "Card-based Protocols Using Triangle Cards". In: *9th International Conference on Fun with Algorithms*. Proceedings: FUN 2018 (La Maddalena, Italy, June 13–15, 2018). Ed. by H. Ito, S. Leonardi, L. Pagli, and G. Prencipe. LIPIcs 100. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018, 31:1–31:13. DOI: 10.4230/LIPIcs.FUN.2018.31.

[SM19] K. Shinagawa and T. Mizuki. "Secure Computation of Any Boolean Function Based on Any Deck of Cards". In: *Frontiers in Algorithmics. 13th International Workshop*. Proceedings. FAW 2019 (Sanya, China, Apr. 29–May 3, 2019). Ed. by Y. Chen, X. Deng, and M. Lu. LNCS 11458. Springer, 2019, pp. 63–75. DOI: 10.1007/978-3-030-18126-0_6.

[SMS+15a] K. Shinagawa, T. Mizuki, J. C. N. Schuldt, K. Nuida, N. Kanayama, T. Nishide, G. Hanaoka, and E. Okamoto. "Multi-party Computation with Small Shuffle Complexity Using Regular Polygon Cards". In: *Provable Security. 9th International Conference*. Proceedings: ProvSec 2015 (Kanazawa, Japan, Nov. 24–26, 2015). Ed. by M. H. Au and A. Miyaji. LNCS 9451. Springer, 2015, pp. 127–146. DOI: 10.1007/978-3-319-26059-4_7.

[SMS+15b] K. Shinagawa, T. Mizuki, J. C. N. Schuldt, K. Nuida, N. Kanayama, T. Nishide, G. Hanaoka, and E. Okamoto. "Secure Multi-Party Computation Using Polarizing Cards". In: *Advances in Information and Computer Security. 10th International Workshop on Security*. Proceedings: IWSEC 2015 (Nara, Japan, Aug. 26–28, 2015). Ed. by K. Tanaka and Y. Suga. LNCS 9241. Springer, 2015, pp. 281–297. DOI: 10.1007/978-3-319-22425-1_17.

[SNN+16] K. Shinagawa, K. Nuida, T. Nishide, G. Hanaoka, and E. Okamoto. "Committed AND protocol using three cards with more handy shuffle". In: *2016 International Symposium on Information Theory and Its Applications*. Proceedings: ISITA 2016 (Monterey, CA, USA, Oct. 30–Nov. 2, 2016). IEEE, 2016, pp. 700–702. URL: http://ieeexplore.ieee.org/document/7840515.

[T18] A. Toponce. *Playing Card Ciphers*. 2018. URL: https://aarontoponce.org/wiki/crypto/card-ciphers (visited on Aug. 23, 2019).

[TZL+17] F. Tramèr, F. Zhang, H. Lin, J. Hubaux, A. Juels, and E. Shi. "Sealed-Glass Proofs: Using Transparent Enclaves to Prove and Sell Knowledge". In: *2017 IEEE European Symposium on Security and Privacy*. Proceedings: EuroS&P 2017 (Paris, France, Apr. 26–28, 2017). IEEE, 2017, pp. 19–34. DOI: 10.1109/EuroSP.2017.28.

[UNH+16] I. Ueda, A. Nishimura, Y. Hayashi, T. Mizuki, and H. Sone. "How to Implement a Random Bisection Cut". In: *Theory and Practice of Natural Computing. 5th International Conference*. Proceedings: TPNC 2016 (Sendai, Japan, Dec. 12–13, 2016). Ed. by C. Martín-Vide, T. Mizuki, and M. A. Vega-Rodríguez. LNCS 10071. 2016, pp. 58–69. DOI: 10.1007/978-3-319-49001-4_5.

[Va] Visa. *Visa Token Service*. URL: https://usa.visa.com/partner-with-us/payment-technology/visa-token-service.html (visited on Jan. 10, 2019).

[Vb] Volksbank Mittelhessen. *VR-mobileCash: Bargeldauszahlung ohne Karte*. URL: https://www.vb-mittelhessen.de/privatkunden/girokonto-kreditkarten/infos-banking/geld-abheben-ohne-karte.html (visited on Aug. 23, 2019).

[V14] T. Verhoeff. *The Zero-Knowledge Match Maker*. 2014. URL: https://www.win.tue.nl/~wstomv/publications/liber-AMiCorum-arjeh-bijdrage-van-tom-verhoeff.pdf (visited on Aug. 23, 2019).

[V76] L. G. Valiant. "Universal Circuits (Preliminary Report)". In: *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*. STOC 1976 (Hershey, Pennsylvania, USA, May 3–5, 1976). Ed. by A. K. Chandra, D. Wotschke, E. P. Friedman, and M. A. Harrison. ACM, 1976, pp. 196–203. DOI: 10.1145/800113.803649.

[WKS+18] Y. Watanabe, Y. Kuroki, S. Suzuki, Y. Koga, M. Iwamoto, and K. Ohta. "Card-Based Majority Voting Protocols with Three Inputs Using Three Cards". In: *2018 International Symposium on Information Theory and Its Applications*. Proceedings: ISITA 2018 (Singapore, Oct. 28–31, 2018). IEEE, 2018, pp. 218–222. DOI: 10.23919/ISITA.2018.8664324.

[WZ17] D. Wichs and G. Zirdelis. "Obfuscating Compute-and-Compare Programs under LWE". In: *58th IEEE Annual Symposium on Foundations of Computer Science*. FOCS 2017 (Berkeley, CA, USA, Oct. 15–17, 2017). Ed. by C. Umans. IEEE Computer Society, 2017, pp. 600–611. DOI: 10.1109/FOCS.2017.61.

List of Publications

[KWH15] A. Koch, S. Walzer, and K. Härtel. "Card-Based Cryptographic Protocols Using a Minimal Number of Cards". In: *Advances in Cryptology – ASIACRYPT 2015*. Proceedings, Part I: 21st International Conference on the Theory and Application of Cryptology and Information Security (Auckland, New Zealand, Nov. 29–Dec. 3, 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.

[HKK+16] G. Hartung, B. Kaidel, A. Koch, J. Koch, and A. Rupp. "Fault-Tolerant Aggregate Signatures". In: *Public-Key Cryptography – PKC 2016*. Proceedings, Part I: 19th IACR International Conference on Practice and Theory in Public-Key Cryptography (Taipei, Taiwan, Mar. 6–9, 2016). Ed. by C. Cheng, K. Chung, G. Persiano, and B. Yang. LNCS 9614. Springer, 2016, pp. 331–356. DOI: 10.1007/978-3-662-49384-7_13.

[HKK+17] G. Hartung, B. Kaidel, A. Koch, J. Koch, and D. Hartmann. "Practical and Robust Secure Logging from Fault-Tolerant Sequential Aggregate Signatures". In: *Provable Security. 11th International Conference*. Proceedings: ProvSec 2017 (Xi'an, China, Oct. 23–25, 2017). Ed. by T. Okamoto, Y. Yu, M. H. Au, and Y. Li. LNCS 10592. Springer, 2017, pp. 87–106. DOI: 10.1007/978-3-319-68637-0_6.

[KKW+17] J. Kastner, A. Koch, S. Walzer, D. Miyahara, Y. Hayashi, T. Mizuki, and H. Sone. "The Minimum Number of Cards in Practical Card-Based Protocols". In: *Advances in Cryptology – ASIACRYPT 2017*. Proceedings, Part III: 23rd International Conference on the Theory and Applications of Cryptology and Information Security (Hong Kong, China, Dec. 3–7, 2017). Ed. by T. Takagi and T. Peyrin. LNCS 10626. Springer, 2017, pp. 126–155. DOI: 10.1007/978-3-319-70700-6_5.

[AGH+19b] D. Achenbach, R. Gröll, T. Hackenjos, A. Koch, B. Löwe, J. Mechler, J. Müller-Quade, and J. Rill. "Your Money or Your Life—Modeling and Analyzing the Security of Electronic Payment in the UC Framework". In: *Financial Cryptography and Data Security. 23rd International Conference*. Revised Selected Papers: FC 2019 (Frigate Bay, Saint Kitts and Nevis, Feb. 18–22, 2019). Ed. by I. Goldberg and T. Moore. LNCS. Springer, 2019. Cryptology ePrint Archive, Report 2019/924. In press.

[KSK19] A. Koch, M. Schrempp, and M. Kirsten. "Card-based Cryptography Meets Formal Verification". In: *Advances in Cryptology – ASIACRYPT 2019*. Proceedings: 25th Annual International Conference on the Theory and Application of Cryptology and Information Security (Kobe, Japan, Dec. 8–12, 2019). Ed. by S. Galbraith and S. Moriai. LNCS. Springer, 2019. In press.

In Submission:

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

[BKM+18] B. Broadnax, A. Koch, J. Mechler, T. Müller, J. Müller-Quade, and M. Nagel. *Fortified Universal Composability: Taking Advantage of Simple Secure Hardware Modules*. 2018. Cryptology ePrint Archive, Report 2018/519. In submission.

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

[KW18] A. Koch and S. Walzer. *Private Function Evaluation with Cards*. 2018. Cryptology ePrint Archive, Report 2018/1113. In submission.