On the Foundations of Key Exchange

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doktor rerum naturalium (Dr. rer. nat.)

von

Christina Brzuska, Master of Science

geboren in Dortmund.

Referenten: Marc Fischlin
Bogdan Warinschi

Tag der Einreichung: 2. August 2012
Tag der mündlichen Prüfung: 1. Oktober 2012

Darmstadt, 2013
Hochschulkennziffer: D 17

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den ausdrücklich in ihr genannten Hilfen – selbständig verfasst habe.

Christina Brzuska

Wissenschaftlicher Werdegang

10/2011–03/2012	Forschungsaufenthalt am Institute for Advanced Study, Princeton
05/2010–10/2012	Doktorandin der Informatik, TU Darmstadt
04/2010	Master of Science, "sehr gut", TU Darmstadt
10/2007–04/2010	Masterstudium in Mathematik, TU Darmstadt
08/2007	Bachelor of Science, "mention très bien", Université Bordeaux I
04/2007–09/2007	Erasmussemester in Mathematik, TU Darmstadt
09/2005–01/2007	Bachelorstudium in Mathematik, Université Bordeaux I
04/2005–09/2005	Mathematikstudium, Universiät Duisburg-Essen

Veröffentlichungen

[BBD+10] Christina Brzuska, Heike Busch, Özgür Dagdelen, Marc Fischlin, Martin Franz, Stefan Katzenbeisser, Mark Manulis, Cristina Onete, Andreas Peter, Bertram Poettering, Dominique Schröder. Redactable Signatures for Tree-Structured Data: Definitions and Constructions. In *ACNS 10*, Volume 6123 of LNCS, pages 87–104. Springer, Berlin, Germany.

[BDF12] Christina Brzuska, Özgür Dagdelen, Marc Fischlin. TLS, PACE, and EAC: A Cryptographic View on Modern Key Exchange Protocols. In *Sicherheit 2012*, Volume 195 of *LNI*, pages 71–82. GI, 2012.

[BFF+09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, Florian Volk. Security of Sanitizable Signatures Revisited. In *PKC 2009*, Volume 5443 of LNCS, pages 317–336. Springer, Berlin, Germany, 2009.

[BFLS09] Christina Brzuska, Marc Fischlin, Anja Lehmann, Dominique Schröder. Sanitizable Signatures: How to Partially Delegate Control for Authenticated Data. In *BIOSIG*, Volume 155 of *LNI*, pages 117–128. GI, 2009.

[BFLS10] Christina Brzuska, Marc Fischlin, Anja Lehmann, Dominique Schröder. Unlinkability of Sanitizable Signatures. In *PKC 2010*, Volume 6056 of LNCS, pages 444–461. Springer, Berlin, Germany, 2010.

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, Stefan Katzenbeisser. Physically Uncloneable Functions in the Universal Composition Framework. In *CRYPTO 2011*, Volume 6841 of *LNCS*, pages 51–70. Springer, 2011.

[BFS+13] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: Relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, ??:??–??, 2013.

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway Key Exchange Protocols. In *ACM CCS*, pages 51–62. ACM Press, 2011.

[BPS12] Christina Brzuska, Henrich Pöhls, and Kai Samelin. Non-interactive Public Accountability for Sanitizable Signatures. In *EuroPKI 2012*. To appear.

Acknowledgements

Marc Fischlin showed me a world that I did not even suspect to exist when I started my PhD. Starting with my first undergraduate course in cryptography, Marc laid the foundations for me to become a theoretic computer scientist. He introduced me to cryptography and research, he challenged me with interesting problems in new areas, he encouraged me to study whatever I was interested in. Marc introduced me to the quirky community of cryptographers and sent me to conferences all over the world. I was very fortunate to have had Marc as a research advisor, collaborator, teacher, example for good scientific practice, friend, and advisor also on non-scientific matters.

Research with Bogdan Warinschi was incredibly intense. We spent uncountably many hours in coffee places scribbling formulae and drawing pictures in notebooks. Bogdan always took a logician's point of view which helped me a lot to understand the formal foundations of cryptography.

In particular, Marc, Nigel Smart, Bogdan, Steven Williams and I had many intense all-day research discussions on key exchange, many of which are reflected in this thesis. I would like to thank all of them for the great online/offline/coffee place/eMail/telephone communication, collaboration, and discussions. I am grateful to the DAAD for supporting this fruitful exchange with Bristol University.

I am also grateful to the DAAD and CASED for supporting my six months visit to the Institute for Advanced Study. Working there with Russell Impagliazzo was an overwhelming experience. He spoilt me with amazing theory problems, he shared his incredible intuition with me, he immersed me into what is known as theory. He made me recall the importance of persistence when fighting with a dragon. Russell is a patient teacher and an enthusiastic advisor; he often compresses an entire book chapter into a five minute presentation. I am still amazed by how much he tought me in these six months. Our work on key agreement is reflected in this thesis.

I would like to thank my co-authors Paul Baecher, Özgür Dagdelen, Martin Franz, Marc Fischlin, Tobias Freudenreich, Stefan Katzenbeisser, Russell Impagliazzo, Anja Lehmann, Mark Manulis, Arno Mittelbach, Cristina Onete, Marcus Page, Andreas Peter, Henrich Pöhls, Bertram Poettering, Kai Samelin, Jakob Schelbert, Dominique Schröder, Heike Schröder, Nigel Smart, Florian Volk, Bogdan Warinschi and Steven Williams.

A lot of people shared their time with me in Darmstadt. I had research discussions, philosophical debates or simply a great time with them. In particular, I would like to thank Paul Baecher, Özgür Dagdelen, Pooya Farshim, Tommaso Gagliardoni, Anja Lehmann, Giorgia A. Marson, Arno Mittelbach, Cristina Onete, Andreas Peter, Bertram Poettering, Heike and Dominique Schröder. Thanks to Giorgia and Arno for proofreading Chapter 7.

Last, but not least, I am grateful to Reiner Hähnle, Stefan Katzenbeisser and Stefan Roth for joining my committee.

My Contribution

Exchange between fellow researchers and mutual contributions are the very nature of science, and this thesis is no exception in this regard. I would like to thank Marc Fischlin, Russell Impagliazzo, Nigel Smart, Bogdan Warinschi and Stephen Williams for the fruitful collaboration and inspiring discussions, and in particular for their contribution to our joint results. Stephen Williams graduated from Bristol University in 2011, and some of the jointly accomplished results also appear in his thesis [Wil11b]. In the following, I will give a detailed account of the joint results that appear in our respective theses.

As a part of our theses, Stephen Williams and I both include the jointly developed composition framework for key exchange that underlies [BFWW11] and [BFS+13] (Chapter 3 of this thesis) as well as two composition theorems, Theorem **1** in [BFWW11] (Theorem 1 in Chapter 4) and Theorem **1** in [BFS+13] (Theorem 3 in Chapter 5) which are the basis for our work on key exchange. In the spirit of this framework, Stephen then analysed SSH [Wil11a] and included the analysis in his thesis, while I focused on TLS [BFS+13] and included the analysis in my thesis (Chapter 6). Moreover, in his thesis, Stephen developed a formal syntax for our framework that facilitates the formal verification of our results and is deployed in [BFWW11]. In turn, I use the minimal syntax of [BFS+13], that is less suitable for formal verification, but leads to a more compact presentation. Moreover, I particularly contributed to the proof that for generally composable BR key exchange protocols, a session matching algorithm is necessary in [BFWW11] (See Section 4.5 in Chapter 4), that Stephen did not include in his thesis.

Moreover, I added Theorem 2 that deals with composition of non-forward secure Bellare-Rogaway key exchange protocols and does not appear anywhere else. Finally, the results in Chapter 7 were developed in discussions with Russell Impagliazzo and have not yet appeared anywere else either.

Note that the full version of [BFS+13] appeared here [BFS+12] and that the full version of [BFWW11] is available on the academic webpages of the authors. We also published a survey [BDF12] that mentions results from [BFS+13, BFWW11].

Abstract

Key exchange protocols allow two parties to agree on a shared secret over an untrusted channel. A huge number of scientific works on key exchange have been published since the discovery of the first key exchange protocol, designed in 1976 by Diffie and Hellman [DH76]. The most prominent of these works is the game-based security model by Bellare and Rogaway [BR93], published in 1993 and, today, known as the "golden standard" for the security of key exchange protocols.

The main purpose of key exchange protocols is to establish keys for a symmetric-key protocol such as a secure channel. Ideally, if both, the key exchange protocol and the channel protocol, are secure, then we would hope that they can be securely composed. While this is true for simulation-based security models [Can01, KT11, BPW03], game-based models usually lack composition theorems altogether. That is unfortunate, as they are often more suitable for real-life protocols such as TLS.

Our first two composition theorems address the composition of BR-secure key exchange protocols with arbitrary symmetric-key protocols. We formally establish that an additional condition is necessary, namely, the key exchange requires a public session matching algorithm.

Maybe surprisingly, many important key agreement protocols are not BR-secure due to the popular technique of explicit key confirmation. We devise a more flexible yet sufficiently strong model for this class of protocols and prove that it is closed under reductions. Overall, we devellop a tool for the modular analysis of real-life protocols and exemplify the use of our framwork on a profile of the TLS protocol.

As in the case of most practical cryptography, information-theoretic security for key exchange protocols is out of reach, i.e., impossible in a formal sense. Therefore, protocols such as TLS rely on computational assumptions, e.g., the Diffie-Hellmann assumption or the hardness of factoring numbers. As the security of protocols is tightly related to the underlying complexity assumptions, researchers have been striving for simpler and weaker assumptions. The holy grail in this area is to base key agreement, one-way functions or simply any type of cryptography on the mere assumptions that $\mathcal{NP}$ does not equal $\mathcal{P}$ [FF93, BT03, AGGM06]. While positive results in this area remain elusive, there has been some recent progress on negative results [HMX10, PTV11], showing that cryptographic primitives such as hash functions cannot be based on $\mathcal{NP}$-hardness unless $\mathbf{co}\mathcal{AM} \subseteq \mathcal{NP}$. In this thesis, we provide two oracle results that show that, via relativizing techniques, these negative results do not carry over to key agreement and regular one-way functions. In particular, we give an oracle relative to which $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is easy while infinitely many often secure key agreement exists; and we give an oracle relative to which $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ is easy while regular function families exists that are infinitely many often one-way.

Zusammenfassung

Ein Schlüsselaustausch ermöglicht zwei Parteien, sich über einen unsicheren Kanal auf einen gemeinsamen, geheimen Schlüssel zu einigen. Diffie und Hellman [DH76] entwickelten 1976 das erste Schlüsselaustauschprotokoll und legten damit den Grundstein für ein fruchtbares Forschungsgebiet. Eine der wichtigsten Arbeiten auf diesem Gebiet ist das spielebasierte Sicherheitsmodell von Bellare und Rogaway [BR93] aus dem Jahre 1993, welches heutzutage als "goldener Standard" für die Sicherheit von Schlüsselaustauschprotokollen gilt.

Ein Schlüsselaustausch wird gemeinsam mit einem anderen Protokoll ausgeführt, welches die abgeleiteten Schlüssel verwendet. Idealerweise hoffte man, daß ein BR-sicherer Schlüsselaustausch und ein sicheres Anwendungsprotokoll auch bei gemeinsamer Ausführung sicher bleiben. Diese Aussage ist für simulationsbasierte Sicherheitsmodelle bekannt [Can01, KT11, BPW03], für spielebasierte Modelle verfügen wir allerdings oft über kein einziges Kompositionstheorem. Das ist insofern ungünstig, als daß spielebasierte Modelle für praktische Protokolle wie TLS besser geeignet sind.

Unsere ersten zwei Kompositionstheoreme befassen sich mit der Komposition von BR-sicheren Protokollen mit Anwendungsprotokollen, die auf symmetrischen Schlüsseln basieren. Wir stellen dabei durch einen formalen Beweis fest, daß es notwendig ist, daß der Schlüsselaustausch einen *Public Session Matching* Algorithmus besitzt.

Überraschenderweise sind einige bedeutende Schlüsselaustauschverfahren nicht BR-sicher, da sie sogenannte *Key Confirmation* verwenden. Wir entwickeln daher ein zugleich flexibleres und starkes Modell für diese Klasse von Protokollen und zeigen, daß das Modell unter Reduktionen abgeschlossen ist. Insgesamt bieten unsere Kompositionsresultate eine Technik zur modularen Analyse praktischer Protokolle, was wir anhand einer abstrahierten Version von TLS aufzeigen.

Da praktische Kryptographie und Schlüsselaustausch insbesondere keine informationstheoretische Sicherheit bieten kann, basieren Protokolle wie TLS auf komplexitätstheoretischen Annahmen wie z.B. der Diffie-Hellman Annahme. Da die Sicherheit eines Protokolls auf dem Schwierigkeitsgrad des zugrundeliegenden Problems beruht, ist die Studie schwächerer, einfacher Annahmen eine wichtige Forschungsrichtung. Das größte offene Problem auf diesem Gebiet ist es, Schlüsselaustausch, One-Way-Funktionen oder andere Primitive auf $\mathcal{NP} \neq \mathcal{P}$ zu reduzieren [FF93, BT03, AGGM06]. Kürzlich konnten Unmöglichkeitsresultate zeigen, dass z.B. Primitive wie Hash-Funktionen nicht auf $\mathcal{NP}$-schwierige Probleme reduziert werden können, sofern $\mathbf{co}\mathcal{AM}$ nicht in $\mathcal{NP}$ liegt [HMX10, PTV11].

In dieser Arbeit zeigen wir mittels zweier Orakelseparierungen, dass es nicht möglich ist, die vorgenannten Unmöglichkeitsresultate mithilfe von relativierenden Reduktionen auf Schlüsselaustausch und reguläre One-Way-Funktionen zu übertragen. Relativ zu unserem ersten Orakel ist $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ einfach, während unendlich häufig sicherer Schlüsselaustausch existiert. Unser zweites Orakel erreicht, dass $\mathcal{AM} \cap \mathbf{co}\mathcal{AM} \subseteq \mathcal{P}$, während zugleich eine Familie regulärer Funktionen unendlich oft one-way ist.

Contents

Chapter 1

Introduction

Key exchange protocols are a widely deployed mechanism that allows two parties to agree on a shared secret over an untrusted channel. Since the discovery of the first key exchange protocol almost 40 years ago, a huge number of key agreement schemes has been proposed with varying security guarantees based on different assumptions (See [BM03] for a survey). Since Bellare and Rogaway [BR93] presented their security definition for key exchange in the early 90th, this game-based notion of key indistinguishability has become a widely accepted standard for the security of key agreement.

In this thesis, we complement the foundations that justify the popularity of the Bellare-Rogaway (BR) model. Moreover, we analyze a profile of the TLS protocol, a popular protocol that escapes the Bellare-Rogaway model while not suffering from obvious attacks. Towards this goal we will introduce a weaker, yet, as we argue, sufficient notion of security to prove the security of TLS. Finally, we will discuss the complexity-theoretic assumptions that underly the security of key exchange protocols, and we will argue that, indeed, there is reason to strive for weaker assumptions that the ones we rely on right now.

1.1 Composition of BR Secure Key Exchange Protocols

Typically, keys derived through a key exchange protocol are used for secure channels, e.g., TLS [DA06], the security protocol used HTTPS connections, in PACE/EAC [BDFK12, DF10], the protocol used with the New German Identity Card or EMV, the protocol for payment with EC-cards, Master Card or Visa Card. If the key exchange protocol is secure on its own and if the subsequent channel protocol is secure on its own, then we would like to conclude that their composition is secure, too.

The Bellare-Rogaway model stipulates that a key exchange protocol is secure if the resulting keys are indinstinguishable from random. Intuitively, one might expect that, indeed, a Bellare-Rogaway secure key exchange protocol can be securely composed with any secure symmetric-key protocol. However, as we exhibit, this theorem holds if and only if the protocol admits a so-called session-matching algorithm, i.e., one should be able to match two sessions just given the information that was transmitted over the

network and is thus considered *public* information. Note that, while it is a sufficient condition, we only prove a weak form of the converse. A more detailed explanation can be found in Chapter 4, Section 4.2. Note that [GR13] show that it is equivalent of whether one adds a session matching algorithm into the *security model* or not. However, they consider a weaker model for key exchange, so their results might not be immediately applicable to our setting. Indeed, it would be interesting to explore the exact relation between their result and ours.

Our first composition theorem considers the setting of forward-secure key exchange protocols showing that if they are BR-secure, then they compose with arbitrary secure symmetric-key protocols. We then show a similar composition theorem for key exchange protocols that are not forward secure. Indeed, they are generally composable with any *single session reducible* protocol, a large class of symmetric-key protocols that we introduce and that includes most primitives such as message authentication schemes, signatures and encryption.

1.2 Compositional Framework for Games

Game-based models are pre-destined for the analysis of practical protocols, because they seem to capture a notion of security that is strong enough, yet as weak as possible. In turn, game-based models usually do not enjoy powerful theoretical properties such as composability. And thus, composition results in the game-based setting are rather scarce. In comparison, simulation-based security models capture a much stronger notion of security; and indeed, almost all composition results have been achieved in simulation-based security models only.

Thus, our results unite the best from both worlds, as we can prove composition results in the setting of game-based security, thus allowing for a *modular* security analysis in a model that is as weak as possible and yet as strong as necessary (at least, that is, what research suggests so far).

It is interesting to note that, although there is a long plethora of works dealing with game-based security of key exchange as a standalone application, e.g. [BR95, BWJM97, BPR00, CK01, LLM06], their composability has never been formally proven or even been defined. This is striking, as key exchange per se is executed in order to use the derived keys in a subsequent task. This thesis intiates the study of composability of game-based key exchange protocols.

One important contribution towards this goal is a framework for general game-based composition that we put forth in Chapter 3. As standard in cryptographic games, we provide an interface to the adversary that allows him to ask queries to the game and thus retrieve information about the cryptographic algorithms in use. The goal of the adversary is to achieve a certain winning condition that is defined by the game. Moreover, the network model allows the adversary to entirely control the network, i.e., enabling active attacks such as message dropping, re-ordering or replacement. The basic framework is applicable to any type of network protocol composition and is not tailored towards key exchange.

We then specialise our model to recover the original definition of Bellare and Rogaway [BR93]. Note that we can strengthen the mode of corruption to capture, for example, the eCK [LLM06] version of the BR model. We implement one modification to the Bellare Rogaway model in order to cover a more general class of protocols. Namely, we do not pair sessions via matching conversations but instead use session identifiers, as put forward in the work by Bellare, Pointcheval and Rogaway [BPR00]. Unlike for matching conversations, one then has to prove that at most two sessions agree on the same session identifier and that two sessions with the same session identifier derive the same key. A third important requirement is regarding authentication, i.e., if two sessions accept with the same session identifier, then they really communicated with their intended partner.

1.3 Composability of General Key Exchange Protocols

Implementing key agreement protocols with key-indistinguishability is highly efficient—additionally, even before our work, key indistinguishability was believed to entail composability. Therefore, one might presume that key indistinguishability is one of the most important criteria in the design of key exchange protocols.

However, as it turns out, designers of real-life protocols such as TLS often desire to implement explicit key confirmation, i.e., they use the derived key to encrypt and authenticate messages already during the key exchange phase. Now, an adversary that shall distinguish whether it holds a real key or a random string can use its key for decryption and/or verification of the message authentication code and thus notice that his string is (not) the real key. Hence, using the session key in the key exchange phase destroys the property of key indistinguishability so that protocols that exhibit this popular type of key confirmation method cannot be proven secure under the Bellare-Rogaway paradigm.

Theoretically, one could implement both, key-indistinguishability and explicit key confirmation simultaneously by using a so-called key refresh step where one hashes the key to derive one or several "fresh" keys [BR93, BPR00], and it turns our that, if we add a key refresh step to TLS, then this modified version of TLS is BR-secure in the random oracle model [MSW10]. However, the key refreshment step looks like an artifact of theory. Indeed, there is little hope to successfully integrate a key refreshment step into a standard merely based on the inability to prove security. On the other hand, without including the key refreshment into the actual protocol, [MSW10] only analyze a modified version of TLS which is unsatisfactory.

Another approach to make practical key exchange protocols fit into the theoretical framework is to hope that the protocol would be secure without the key confirmation messages, so that one can prove a truncated protocol secure. Unfortunately, this is often not the case.

Now, intutively, if a key is intended to be used for encryption, what harm is done if it is used for encryption (even if this happened in the key exchange phase)? We would hope that the protocol is still secure. Our approach aims at giving a foundation

to this intuition and follows definitional ideas by [DDMW06, Sho99a, BBP04]. More concretely, our security model does not demand for the indistinguishability of the session keys anymore. Instead, we define what it means for a key exchange protocol to be "good" for an application primitive/protocol. Namely, a key exchange protocol is suitable for a primtive/protocol, if an adversary cannot break the primitive/protocol that later uses the session keys derived in the key exchange phase.

Interestingly, our model still exhibits composability. Namely, we prove that our notion is closed under reductions, i.e., if the key exchange is suitable for a building primitive, and if a protocol can be reduced to this simpler primitive, then the protocol can be safely used together with the key exchange protocol.

However, there ain't no such thing as a free lunch, so we slightly strengthen the notion of reduction to what we call a *key-independent reduction*. Here, the reduction from the protocol, e.g., an authenticated channel, to the primitive, e.g., a message authentication code, has to be *independent* of the key distribution that is used by the message authentication code. Maybe, at a first look, this sounds impossible to achieve, as the key distribution could be always the all zero string. In this case, it is trivial to break the authenticated channel—but in this case, it is also trivial to break the message authentication code. Recall that a reduction transforms an adversary against the protocol into an adversary against the primitive. A key-independent reduction sais that similarly, if the key distribution makes the authenticated channel insecure then it should also make the message authentication code secure.

Using the notion of key-independent reduction, our composition theorem shows that if a key exchange protocol is *suitable for a primitive* and if the protocol reduces to this primitive via a *key-independent reduction*, then the key exchange protocol is *suitable for the protocol*.

1.4 On the Security of TLS

Just as EMV and PACE/EAC, the TLS protocol consists of two phases, a key exchange phase, known as the TLS handshake, and a secure channel phase, called the TLS record layer protocol.

Our definitional composition framework captures, in many ways, a sufficient and neccessary condition for the security of a key agreement protocol whose keys are used in an application protocol. Indeed, we can show that a profile of the TLS protocol satisfies our notion thus confirming the intuition that key confirmation does not harm the security of the overall protocol.

The TLS record layer implements a length-hiding authenticated encryption scheme (LHAE), and Paterson, Risternpart and Shrimpton [PRS11] prove that the encryption scheme in the TLS record layer is indeed LHAE-secure. Their analysis was preceeded by a number of works that analyze abstracted versions of the channel implementation [BN00, Kra01, MT10].

The analysis of the channel will be a modular ingredient for our analysis of TLS, namely, whenever the encryption scheme is LHAE-secure, then our analysis applies—

even if at a later time, the current encryption scheme might be replaced by another LHAE-secure encryption scheme.

Recently, Jager et al. [JKSS12] also performed a monolithic analysis of TLS-DH, i.e., TLS using a Diffie-Hellman key exchange as the underlying key agreement method. Besides our work, they are the first to analyze an unmodified version of TLS in contrast to [MSW10], for example. They analyse TLS in a new model for so-called authenticated and confidential channel establishment (ACCE) protocols that they put forward. Besides small technical differences, their model for ACCE is indeed an instance of our framework. Namely, their ACCE model considers the combined security of a key agreement protocol and (a stateful version of) lenth-hiding authenticated encryption LHAE, which in our words, means that they prove that the TLS key agreement protocol is *suitable* for LHAE.

We use TLS to exemplify the use of our composition theorem, while Jager et al. focused on the analysis of TLS. In particular, we developed a composition framework allowing for a modular approach while their model only allows for a monolithical analysis. In turn, the presentation of their model is much shorter. However, devising a general framework allowed us to avoid certain caveats. In particular, they used matching conversations as a partnering mechanism so that their model was initially too strong for TLS and indeed, their was a small gap in the first version of their proof; they then restricted the reveal queries of the adversary and obtained a proof in that model. However, now, some real-life attacks are not reflected in their model anymore. Moving to session-identifier-based partnering, as in our model, should be able to overcome this problem.

They are the first to provide an analysis of TLS in the standard model. As a drawback, their analysis is based on a non-standard assumption. It should be interesting to see whether the latter can be ultimatively eliminated. In turn, their standard model analysis could also be carried out in our framwork.

Hence, in a nutshell, each approach has its advantages, in particular, we analyse TLS in the random oracle model, while they present a standard-model analysis. In turn, we provide a framework that allows for the modular analysis of key exchange protocols, while their model is tailored towards TLS.

1.5 Complexity-Theoretic Assumptions

The TLS protocol is designed based on several building blocks. One building block is a *passively* secure key exchange protocol such as the Diffie-Hellman key exchange protocol or an RSA-based key transport protocol. Other building blocks are secure signature schemes, symmetric authenticated encryption and secure hash functions. To assess the security of protocols such as TLS, we assume that these building blocks are secure and then prove that—under this assumption—the way in which TLS combines them, is secure, too.

Thus, the security of TLS need both, an analysis like ours as well as a proof of security for its underlying building blocks. Indeed, the security of these building blocks usually reduces to a complexity theoretic assumption, i.e., the hardnessness of a complexity-

theoretic problem. In this thesis, we elaborate on the weakest possible assumptions to build passively secure key agreement, a building block for TLS and other TLS-like protocols.

A weak assumption for key agreement would be, for example, that the set of problems solvable in non-deterministic polynomial-time $\mathcal{NP}$ is not contained in the set of problems that are solvable in polynomial-time $\mathcal{P}$. However, usually, we make much stronger, probabilistic assumptions, as, for example, the existence of one-way functions; functions that are easy to compute, but hard to invert on the average. It turns out that this is inherent as the existence of passively secure key agreement already implies the existence of one-way functions [IL89].

When considering specific assumption for key agreement such as the decisional Diffie-Hellman assumption, it turns out that not only they imply one-way functions, but also, they imply hard problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$. Whether this is inherent, is not known, and we prove that there is no relativizing reduction that could establish such as result, namely, we prove that there is an oracle Π such that relative to Π:

(i) secure key agreement exists.

(ii) $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is contained in $\mathcal{P}$.

Our result can be seen as a strengthening of the result by Chang et al. [CHL02] who prove that there exists and oracle Π relative to which

(i) secure key agreement exists.

(ii) one-way permutations do not exist.

Note that our result implies their result as a special case, as $\mathcal{NP} \cap \mathbf{co}\mathcal{NP} \subseteq \mathcal{P}$ implies that one-way permutations do not exist. Note that our result also implies a family of oracle separations, namely, whenever a cryptographic primitive P implies hard problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ (through a relativizing reduction), then using our result, one yields an oracle relative to which secure key agreement exists while the cryptographic primitive P does not. Note that Haitner et al. [HMX10] recently showed that a family of cryptographic primitives such as collision-resistent hash functions implies hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$, a slightly bigger class than $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$. If our results can be extended to $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$, then automatically, we yield an oracle separation between key agreement and all of these primitives, e.g., that the existence of passively secure key agreement does not imply collision-resistent hash functions.

One-way functions are a necessary condition for all of modern cryptography, and they turn out to be a sufficient condition for many cryptographic primitives such as message authentication codes and symmetric encryption schemes. The latter two imply symmetric authenticated encryption and are thus an important building block for the TLS protocol. A slightly stronger assumption are so-called regular one-way functions. It was suspected that regular one-way functions are actually a much stronger assumption than one-way functions alone [AGGM06], namely, that one-way functions already imply hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ (which we do not believe to be true for standard one-way

functions). However, our second oracle separation shows that a relativizing reduction cannot establish that regular one-way functions imply hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. Namely, we give an oracle Π such that relative to Π,

(i) regular one-way functions exist.

(ii) $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ is containted in $\mathcal{P}$.

This might explain the difficulty that Akavia et al. [AGGM06] had in establishing a reduction that turns regular one-way functions into hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. Their goal was to prove that any regular one-way function can be broken by an adversary that has complexity less than $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. They argue that, under the assumption that $\mathcal{NP}$ is not contained in $\mathbf{co}\mathcal{AM}$, this means that regular one-way functions cannot be based on the mere assumption that that $\mathcal{NP}$ is not contained in $\mathcal{P}$: assume that given access to a good inverter for a one-way function, a polynomial-time reduction R can decide any problem in $\mathcal{NP}$. And assume that a good inverter for the one-way function can be implemented in complexity $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. Then the reduction together with the inverter has complexity $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ and decides an $\mathcal{NP}$-hard problem, implying $\mathcal{NP} \subseteq \mathbf{co}\mathcal{AM}$.

Akavia et al. [AGGM10] exhibit a gap in their proof that relates regular one-way functions to hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$, and our oracle result can be seen as an explanation of why the gap is hard to bridge. Besides presenting the results, Chapter 7 will also introduce the two tools that we borrow from complexity-theory, namely, generic oracles which have previously been used by Blum and Impagliazzo [BI87] to separate classical complexity classes, and the relation between certificate complexity and block sensitivity that was discovered by Nisan [Nis91]. We will now elaborate on existing works and known techniques that are relevant to this thesis, and we will explain how our results relate to those.

1.6 Related Work

Definitions of Key Exchange. There has been a plethora of work on the security of network protocols, and this line of research is particularly long in the area of key agreement. We now first give a high-level overview and then consider in detail the works that are most relevant to us. Security of network protocols or protocols and primitives in general usually either follows the game-based approach or is based on a simulation-based notion such as the universal composition (UC) framework and similar frameworks [Can01, BPW07, Küs06].

Simulation-based security models are endowed with powerful theoretical properties such as composability. Unfortunately, this additional structure comes at the price of strengthening the model to the point that implementing primitives is less efficient, or, in some cases even impossible [CF01].

It is thus an important goal in the area of modelling to devise models that are as weak as possible (to allow for efficient implementation) but still as strong as necessary (to capture all real-life attacks). For key exchange, whenever the simulation-based

model [CK02] is too strong for the protocol in question, one might instead opt for a version of the Bellare-Rogaway model [BR93]. All models allow for active attacks on the network, some form of corruption, and they consider leakage of session keys. The Bellare-Rogaway model then demands that the session keys of uncorrupted, unrevealed sessions remain indistinguishable from random. We will present their definition in Chapter 3 and use it throughout Chapter 4. Note that instead of matching sessions via matching conversations as [BR93], we match sessions through session identifiers as introduced in [BPR00].

Composition theorems in the game-based setting are rare, and interestingly, one of the few results in this area indeed also deals with key exchange and has a goal that is quite similar to what we prove in Chapter 4. Namely, Canetti and Krawczyk [CK01, CK02] show that if a protocol is secure according to the game-based notion of SK-security, a BR-variant, then it is securely composable with arbitrary secure protocols. Albeit pursuing a similar goal, their result is not entirely in the game-based setting. They proceed by showing that their (game-based notion) of SK-security is equivalent to some version of the (simulation-based) UC-security and then apply the UC composition theorem, i.e., their composition theorem can combine a game-based notion of key exchange with a simulation-based secure application protocol.

Besides broader applicability (as game-based notions tend to be weaker), we can also avoid subtleties in the UC-framework such as pre-established session identifiers. It may be noted that this frequently criticized artifact of the UC framework can be easily circumvented by establishing session-identifiers beforehand, as done in [BLR]. However, UC-security requires these identifiers to be used in the run of the protocol, and that is not the case for practical protocols. As we target practical protocols in their due form, we thus prefer to avoid the restrictive syntax of UC.

Recently, Küsters and Thurgenthal [KT11] develop a UC-like framework that is able to reflect matching via session identifiers as common in practical protocols. An interesting question for future work is to try to analyze TLS in their framework.

Another note on the composition theorem by Canetti and Krawczyk [CK01, CK02] is that SK-security cannot be proven to imply UC-security straight away. Either one needs to weaken UC-security by adding non-standard oracles, or one boosts the security of the protocol via secure erasure, usually considered a strong assumption. It would be interesting to see to what extend this also applies to [KT11].

We thus work with a game-based model and show in Chapter 4 that key indistinguishability is a sufficient condition for general composability. In a specific context, however, it might not be a necessary condition, which is precisely the topic of Chapter 5. While we consider general applications, specific use cases have been explored by others before. For example, Bellare et al. [BBP04] consider an encapsulated key as secure if it can be safely used for a data encapsulation mechanism (DEM). Also, Datta et al. [DDM+05] explore the idea of context-specific security in the are of *Protocol Compositional Logic*. Similarly to our notion, they call a key "good" if information learned about it before do not harm the security of the encryption scheme that the key is used for. In the same context, the work by Datta et al. [DDMW06] targets to translate prop-

erties of game-based security into their logical framework. As common in the area of formal methods, their theorems only apply to protocols that are specified in a specific syntax, so that incorporating an additional primitive entails proving a new theorem for the the extended language. In turn, our theorem holds generally for all primitives and protocols.

A notion of security that might sound similar to ours is the security definition by Shoup [Sho99b]. He generalizes key indistinguishability to key indistinguishability when additionally, other protocols make use of the session key. This allows him to model information loss through key confirmation messages. However, this is different from our setting where we desire to argue that a composed protocol is secure (as it is not sufficient that the key is indistinguishable).

Assumptions for Key Agreement. In the area of lattice-based cryptography, there is a long line of research that deals with sufficient assumptions for key agreement. Note that most of these papers discuss public-key encryption, which is existentially equivalent to two-move key agreement. Potentially, key agreement protocols with more messages could be based on strictly weaker assumptions [Rud92]. Yet, to date, the power of interaction has never been exploited towards weaker assumptions.

The most promising lattice-based constructions for public-key encryption/two-move key agreement are those that admit a worst-case to average-case reduction with an $\mathcal{NP}$-hard worst-case problem [Ajt96, Reg03, MR04, Reg05]. Unfortunately, parameter choice is a crucial issue in the area of lattice problems. For example, the parameters γ, for which γ-GapSVP (Gap version of the **S**hortest **V**ector **P**roblem in lattices) is $\mathcal{NP}$-hard, are not known to yield crypto-systems. In turn, known cryptosystems based on γ-GapSVP use parameters γ such that the corresponding problem γ-GapSVP is already in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ and thus unlikely to be $\mathcal{NP}$-hard [GG98, AR05].

Our oracle result might indicate which features of the schemes could be exploited to yield, indeed, schemes from $\mathcal{NP}$-hardness.

Impossibility Results. Two main techniques to establish separations in cryptography are oracle separations and meta-reductions. To prove an oracle separation between two primitives Q and P, one constructs an oracle relative to which the primitive P exists, while Q does not exist. In this way, one rules out a relativizing construction from Q out of P. The most famous oracle separation is the seminal paper by Impagliazzo and Rudich [IR90] who prove that, relative to a **PSPACE** oracle and a random oracle, key agreement does not exist, while one-way functions do. Impagliazzo and Rudich introduced oracle separations into the area of cryptography and thereby initiated a long line of research dealing with oracle separations between cryptographic primitives, e.g., [DOP05, HHRS07, HH09, KP09, BCFW09, FLR$^+$10, MP12, LOZ12, BH13, HOZ13]. Concurrently to the result that separates key agreement from one-way functions, Rudich proved in his thesis [Rud88] that there is an oracle relative to which one-way functions exist, while one-way permutations do not. We will study this result in more detail in Chapter 7.

The second important technique for separation results are so-called meta-reductions. These results are usually conditional, i.e., they rely on some complexity assumption

such as $\mathcal{NP} \neq \mathbf{co}\mathcal{NP}$ or the existence of one-way functions, and they show that, if we can build Q out of P, then the assumption does not hold. For example, several meta-reductions rule out certain types of reductions from one-way function to $\mathcal{NP}$-hardness [FF93, BT03, AGGM06]. All three works assume that $\mathcal{NP}$ is not contained in $\mathbf{co}\mathcal{NP}$ (or in $\mathbf{co}\mathcal{AM}$) and then prove that if there is a reduction of a certain type from a one-way function to an $\mathcal{NP}$-hard problem, then $\mathcal{NP}$ is contained in $\mathbf{co}\mathcal{NP}$ (or $\mathbf{co}\mathcal{AM}$). Meta-reductions were first used by by Boneh and Venkatenesan [BV98] and, as for oracle separations, by now, there is a long line of works that prove separation results via meta-reductions, e.g., [Cor02, PV06, HRS09, FS10, Pas11, GW11, DHT12, KK12, Seu12, FF13].

We often call a separation result an *impossibility* result, namely, because a separation result shows that it is impossible to establish a reduction of a certain type. In particular, that means that separation results can possibly be circumvented using black-box techniques, as for example in the ingenious work by Barak [Bar01]. It is thus crucial to identify the class of techniques/reductions that are ruled out in order to point out the leverages to bypass them. [RTV04] and [BBF13] both give a framework to classify reductions. In both taxonomies, our results rule out relativizing reductions, which, according to [BBF13], is a strong separation that even rules out constructions that treat the primitive in a non-black-box and also allows the reduction to treat the primitive and the adversary in a non-black-box way, which are called NNN-reductions in the taxonomy of [BBF13]. Moreover, as key exchange with non-perfect correctness as well as regular one-way functions allow for embedding according to [RTV04], our results actually also rule out semi-black-box reductions, that is, reductions that only need to work for efficient adversaries, NNNa-reduction in the notation of [BBF13].

Interestingly, the results in Chapter 7 can also be interpreted as negative results for negative results—or as oracle separations that show that a certain meta-reduction is unlikely to exist. Namely, Akavia et al. [AGGM06] aim at proving a meta-reduction; our oracle result indicates that the gap in their proof might be difficult to bridge, at least via this form of meta-reduction.

Chapter 2

Preliminaries

We introduce the security definitions for basic cryptographic primitives such as symmetric encryption and message authentication codes as well as standard complexity classes. For an integral introduction to cryptography, we recommend [Gol04, Gol01], and for an integral introduction to complexity theory, we recommend [Gol08, AB09].

Throughout this thesis, the notion of an efficient algorithm denotes an interactive Turing machine that runs in probabilistic polynomial-time, unless explicitly stated otherwise.

2.1 Cryptography

A cryptographic primitive is a building block that can be used to build more complex cryptographic protocols or different/more complex cryptographic primitives. While there is no formal distinction between cryptgraphic primitives and cryptographic protocols, we usually think of primitives as implementing some non-interactive task such as encrypting or signing a message. In turn, protocols are an exchange of messages between two or more parties and usually interactive, such as, for example, in a key exchange protocol, or a secure channel.

Security of both, cryptographic primitives and protocols, is usually modelled through a probabilistic experiment. As discussed in the introduction, the main two approaches are game-based and simulation-based. In this thesis, we are exclusively concerned with game-based security. Here, security is described through an adversary playing a game. The game gives challenges to the adversary that the adversary has to "break", and the adversary can usually request some additional information from the game through queries, and he tries to "win" the game. The game is modelled in such a way such that, if the adversary breaks the challenges, then the he also wins the game. In the next chapter, we will study typical properties of cryptographic games. We preceed these rather abstract considerations with more concrete examples in this chapter that are standard in cryptography.

One-Way Functions and Pseudo-Random-Generators. The most basic version of game-based security are non-interactive games. Here, the adversary gets a challenge

and has to solve it. He does not get access to any queries. For example, a one-way function f is a function that is easy to compute, but hard to invert, on the average. Here, the game draws a random input x, computes $y := f(x)$ and gives y to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ then tries to find a pre-image x' of y and "wins" if $f(x') = y$. A function f is called one-way, if all efficient adversary $\mathcal{A}$ fail to compute a pre-image almost all the time, formally:

Definition 1 (One-Way Functions and Negligible Functions)**.** *An efficiently computable function f is one-way, if for all efficient adversaries $\mathcal{A}$, one has that*

$$\mathrm{Prob}_{x \leftarrow \{0,1\}^n} \left[\mathcal{A}(1^n, f(x)) \rightarrow x' \in f^{-1}(y) \right]$$

is negligible. A function $\epsilon(n)$ is negligible if it tends to 0 faster than the inverse of any positive polynomial when n tends to infinity.

One-way functions are the most prominent representatives of the class of search problems in cryptography. Roughly, a search problem is a problem where the adversary tries to find a long string that satisfies some property such as $f(x') = y$.

The second important class of problems in cryptography are decision problems where the adversary wins the game, if he successfully guesses a secret bit. A prominent example of a (non-interactive) decision problem are pseudo-random generators. A pseudo-random generator G is an efficiently computable function that turns a small number of truly random bits into a bigger number of pseudo-random bits. Pseudo-random means that for any efficient adversary $\mathcal{A}$, it is is hard to distinguish the output $G(x)$ of the pseudo-random generator from a random string of the same length. Here, the game flips a random bit b. If the bit is 0, the game gives $G(x)$ to the adversary. If the bit is 1, the game gives a random string of the same length to the adversary. The adversary then returns a bit d and wins if $d = b$. The pseudo-random generator is considered secure, if all efficient adversary fail to determine b better than with guessing probability, that is, the probability that $\mathcal{A}$ returns 1 on input $G(x)$ is the same as the probability hat $\mathcal{A}$ returns 1 on input a random string of length $|G(x)|$, up to negligible differences, formally:

Definition 2 (Pseudo-Random Generator)**.** *An efficiently computable function G with stretch $0 < s : \mathbb{N} \rightarrow \mathbb{N}$ is a pseudo-random-generator, if $G(\{0,1\}^n) \subseteq \{0,1\}^{n+s(n)}$ for all $n \in \mathbb{N}$ and for all efficient adversaries $\mathcal{A}$, one has that*

$$\left| \mathrm{Prob}_{x \leftarrow \{0,1\}^n} \left[\mathcal{A}(1^n, G(x) = 1) \right] - \mathrm{Prob}_{y \leftarrow \{0,1\}^{n+s(n)}} \left[\mathcal{A}(1^n, y) = 1 \right] \right|$$

is negligible.

Note that without the stretch condition, construction a pseudo-random generator is trivial, as the identity function would be a valid candidate.

Symmetric encryption. An encryption of a message should hide all possible information of the encrypted message, even if the attacker has partial information about the message. The security game that captures this goes back to Goldwasser and Micali [GM82]. Security models for encryption ask that, even if the adversary knows

(chooses) two messages m_0 and m_1, then their respective ciphertexts should be indistinguishable from each other. At a first sight, this security requirement might sound too strong, but imagining that we only encrypt a single bit, then, if the encryptions of 0 and 1 are not indistinguishable, then basically, the ciphertext leaks all the information about the encrypted message. Thus, for most applications, it is not recommended to use encryption schemes that are insecure according to the well-established notion of IND-CPA security that we define next—in some cases, it is even advisable to opt for a stronger notion, called IND-CCA security.

The security game for encryption is a decision game with a secret bit b, that allows the adversary to ask for encryptions of messages and also to ask challenge queries, where the adversary sends two messages m_0 and m_1 and gets back an encryption of the message m_b, depending on the secret bit b. Encryption queries are called chosen message attacks, and the corresponding security game is known as IND-CPA, indistinguishability under chosen plaintext attacks. If additionally, the adversary can make decryption queries, then the game is called IND-CCA (or IND-CCA2), namely indistinguishable under chosen-ciphertext attacks.

Formally, we define an encryption scheme $\mathcal{E}$ as a triple of algorithms $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where KeyGen is the key generation algorithm. To generate a key, we execute $k \leftarrow \mathsf{KeyGen}(1^\eta)$. We have $c \leftarrow \mathsf{Enc}_k(m)$, for a message m taken from the message space (M) of $\mathcal{E}$ and $m' \leftarrow \mathsf{Dec}_k(c')$, with $m' \in M \cup \{\perp\}$. The correctness requirement of encryption is that for all messages m, it holds with overwhelming probability over key generation and encryption that if $k \leftarrow \mathsf{KeyGen}(1^\eta)$ and $c \leftarrow \mathsf{Enc}_k(m)$, then $m = \mathsf{Dec}_k(c)$.

An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *indistinguishable under chosen ciphertext attacks* (IND-CCA2), if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{CCA2}_{\mathcal{A}}^{\mathsf{Enc}}$ evaluates to 1 is negligibly close to $\frac{1}{2}$ (as a function of n), where

Experiment $\mathsf{CCA2}_{\mathcal{A}}^{\mathsf{Enc}}(n)$
$k \leftarrow \mathsf{KeyGen}(1^n)$.
$b \leftarrow \{0,1\}$.
$d \leftarrow \mathcal{A}^{\mathsf{Enc}_k(\cdot,\cdot),\mathsf{Dec}_k(\cdot)}(1^n)$,
Where on input of m_0, m_1, the Enc oracle returns the output of $\mathsf{Enc}_k(m_b)$
The oracle Dec on input of c returns $\perp$, if c has been an output of oracle Enc,
Otherwise it returns $\mathsf{Dec}_k(c)$.
Return 1 iff $b = d$.

A symmetric encryption scheme is secure according to the weaker notion of indistinguishability under the chosen-plaintext attacks (IND-CPA), if the adversary is not given decryption oracle. An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *indistinguishable under chosen plaintext attacks* (IND-CPA), if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{CPA}_{\mathcal{A}}^{\mathsf{Enc}}$ evaluates to 1 is negligibly close to $\frac{1}{2}$ (as a function of n), where

Experiment $\mathsf{CPA}_{\mathcal{A}}^{\mathsf{Enc}}(n)$
$k \leftarrow \mathsf{KeyGen}(1^n)$.
$b \leftarrow \{0,1\}$.

$d \leftarrow \mathcal{A}^{\mathsf{Enc}_k(\cdot,\cdot)}(1^n)$,
Where on input of m_0, m_1, the Enc oracle returns the output of $\mathsf{Enc}_k(m_b)$
Return 1 iff $b = d$.

Asymmetric Encryption. We define an encryption scheme $\mathcal{E}$ as a triple of algorithms $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where KeyGen is the key generation algorithm. To generate a key pair we execute the randomized algorithm $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\eta)$. Encryption works via the randomized algorithm $c \leftarrow \mathsf{Enc}(pk, m)$, for a message m taken from the message space (M) of $\mathcal{E}$. To decryption, we run $m' \leftarrow \mathsf{Dec}(sk, c')$, with $m' \in M \cup \{\perp\}$. The correctness requirement of encryption is that with overwhelming probability over key generation and encryption that if $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\eta)$ and $c \leftarrow \mathsf{Enc}(pk, m)$, then $m = \mathsf{Dec}(sk, c)$.

An encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be *indistinguishable under chosen ciphertext attacks* (IND-CCA2), if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{CCA2}^{\mathsf{Enc}}_{\mathcal{A}}$ evaluates to 1 is negligibly close to $\frac{1}{2}$ (as a function of n), where

Experiment $\mathsf{CCA2}^{\mathsf{Enc}}_{\mathcal{A}}(n)$
$(sk, pk) \leftarrow \mathsf{KeyGen}(1^n)$.
$b \leftarrow \{0, 1\}$.
$d \leftarrow \mathcal{A}^{\mathsf{Enc}(pk,\cdot,\cdot),\mathsf{Dec}(sk,\cdot)}(pk, 1^n)$,
where the Enc oracle on input of m_0, m_1 returns the output of $\mathsf{Enc}(pk, m_b)$.
The oracle Dec on input of c returns $\perp$, if c has been an output of oracle Enc.
Otherwise it returns $\mathsf{Dec}_k(c)$.
Return 1 iff $b = d$.

We say that an encryption scheme satisfies the weaker notion of indistinguishability under chosen-plaintext attacks (IND-CPA), if for all efficient adversary the probability that the experiment $\mathsf{CPA}^{\mathsf{Enc}}_{\mathcal{A}}$ evaluates to 1 is negligibly close to $\frac{1}{2}$. Note that the difference between $\mathsf{CPA}^{\mathsf{Enc}}_{\mathcal{A}}$ and $\mathsf{CCA2}^{\mathsf{Enc}}_{\mathcal{A}}$ is the presence of a decryption oracle.

Experiment $\mathsf{CPA}^{\mathsf{Enc}}_{\mathcal{A}}(n)$
$(sk, pk) \leftarrow \mathsf{KeyGen}(1^n)$.
$b \leftarrow \{0, 1\}$.
$d \leftarrow \mathcal{A}^{\mathsf{Enc}(pk,\cdot,\cdot)}(pk, 1^n)$,
where the Enc oracle on input of m_0, m_1 returns the output of $\mathsf{Enc}(pk, m_b)$.
Return 1 iff $b = d$.

Authentication. Authentication is typically a search problem, namely, the adversary tries to fake (a string of) information that makes the receiver believe that a certain message was sent by another sender than himself. Usually, we also allow the adversary to see messages that have been authenticated by the sender. The symmetric-key authentication primitive is called a message-authentication codes (MAC), the asymmetric-key authentication primitive is called a signature scheme, and authenticated protocols, both symmetric-key and asymmetric-key, are called authenticated channels.

A MAC scheme MAC is a triple of algorithms $\mathsf{MAC} = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$, where keys for the scheme a generated by $k \leftarrow \mathsf{KeyGen}(1^\eta)$. We let $\sigma \leftarrow \mathsf{Mac}_k(m)$ with $m \in \{0,1\}^*$ and define $v \leftarrow \mathsf{Verify}_k(m', \sigma')$ where $v \in \{\mathsf{true}, \mathsf{false}\}$. Further, we require that $\mathsf{Verify}_k(m, \mathsf{Mac}_k(m)) = \mathsf{true}$ for all m with overwhelming probability over key generation and Mac generation.

A message authentication code $(\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ is called *unforgeable under chosen message attacks* if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{Mac}}$ evaluates to 1 is negligible (as a function of n), where

Experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{Mac}}(n)$
$k \leftarrow \mathsf{KeyGen}(1^n)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Mac}_k(\cdot), \mathsf{Verify}_k(\cdot,\cdot)}(1^n)$
Return 1 iff $\mathsf{Verify}_k(m^*, \sigma^*) = 1$ and $\mathcal{A}$ has never queried $\mathsf{Mac}_k(\cdot)$ about m^*.

A signature scheme SIG is a triple of algorithms $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sig}, \mathsf{Verify})$, where keys for the scheme a generated by $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\eta)$. We let $\sigma \leftarrow \mathsf{Sig}(sk, m)$ with $m \in \{0,1\}^*$ and define $v \leftarrow \mathsf{Verify}(m', \sigma', pk)$ where $v \in \{\mathsf{true}, \mathsf{false}\}$. Further, we require that genuinely generates signature are accepted with overwhelming probability.

As signature scheme $(\mathsf{KeyGen}, \mathsf{Sig}, \mathsf{Verify})$ is called *unforgeable under chosen message attacks* if for any efficient algorithm $\mathcal{A}$ the probability that the experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{SIG}}$ evaluates to 1 is negligible (as a function of n), where

Experiment $\mathsf{Forge}_{\mathcal{A}}^{\mathsf{Sig}}(n)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^n)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sig}(sk,\cdot)}(pk, 1^n)$
Return 1 iff $\mathsf{Verify}(m^*, \sigma^*, pk) = 1$ and $\mathcal{A}$ has never queried $\mathsf{Sig}(sk, \cdot)$ about m^*.

In both, the symmetric and the asymmetric setting, we sometimes need the notion of *strong* unforgeability. A signature scheme/message authentication code is strongly unforgeable under chosen message attacks, if no efficient adversary can generate a valid fresh *pair* of a message and a tag/signature. Note that in the standard notion of unforgeability, the message has to be fresh, while for strong unforgeability, only the pair needs to be new. Thereby, we assure that an adversary cannot generate a second tag/signature for a message, for which it already received a tag/signature from the $\mathsf{Mac}/\mathsf{Sig}$ oracle.

Computational Diffie–Hellman Assumption. A common assumption in cryptography are the computational Diffie-Hellman assumption (CDH) that is a search problem and its decisional variant, the decisional Diffie-Hellman assumption (DDH).

We here state the computational Diffie-Hellman Assumption (CDH), which we will use in the discussion of TLS in Chapter 5. CDH is widely believed to hold in various groups, e.g., in certain elliptic curves. Here, we consider the assumption over prime order groups $\mathbb{Z}_p$. The algorithm Params returns a prime p, a generating group element g and a range parameter q for the exponents.

Definition 3 (CDH)**.** *The CDH problem is hard with respect to an instance generation*

algorithm Params*, if for all efficient algorithms* $\mathcal{A}$ *the following probability is negligible:*

$$\Pr\left[(p,q,g) \leftarrow \mathsf{Params}(1^n); a,b \leftarrow \mathbb{Z}_q : \mathcal{A}(p,q,g,g^a,g^b,1^n) = g^{ab}\right],$$

where all group elements g, g^a, g^b, g^{ab} *are reduced modulo* p*.*

Decisional Diffie–Hellman Assumption. A stronger version of the computational Diffie-Hellman assumption is its decisional counterpart whose properties we will discuss in detail in Chapter 7. In particular, we will show, that both, the computational and the decisional Diffie-Hellman assumption imply hard problems in the complexity class $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ that we define in the following section.

Definition 4 (DDH)**.** *The DDH problem is hard with respect to an instance generation algorithm* Params*, if for all efficient algorithms* $\mathcal{A}$ *the following difference of probabilities is negligible:*

$$\Pr\left[(p,q,g) \leftarrow \mathsf{Params}(1^n); a,b,c \leftarrow \mathbb{Z}_q : \mathcal{A}(p,q,g,g^a,g^b,g^{ab},1^n) = 1\right]$$
$$-\Pr\left[(p,q,g) \leftarrow \mathsf{Params}(1^n); a,b,c \leftarrow \mathbb{Z}_q : \mathcal{A}(p,q,g,g^a,g^b,g^c,1^n) = 1\right],$$

where all group elements g, g^a, g^b, g^c, g^{ab} *are reduced modulo* p*.*

2.2 Complexity Theory

While crpytography inherently requires probabilistic average-case definitions, this is not so for classical complexity theory, often called worst-case complexity theory to differentiate it from average-case complexity theory. A problem in complexity theory is usually a language $\mathcal{L}$, a set of strings; and an algorithm solves the problem, if it decides the language, i.e., it computes the characteristic function $\chi_{\mathcal{L}}$ that returns 1 on inputs from the language $\mathcal{L}$ and 0 on all other strings. A relaxation of this notion is called a promise problem and splits the set of all strings into three sets, **Yes**-instances, **No**-instances and a third set that we call **Nocare**. An algorithm solves a promise problem, if it returns 1 on **Yes**-instances and 0 on **No**-instances. We do not care about the algorithm's behavior on **Nocare**-instances.

Definition 5. *A language* $\mathcal{L}$ *is a set of strings* $\mathcal{L} \subseteq \{0,1\}^*$*. An algorithm* M *is said to decide* $\mathcal{L}$*, if it computes* $\chi_{\mathcal{L}}$ *correctly. A promise problem consists of a partition of* $\{0,1\}$ *into three sets* ***Yes****,* ***No*** *and* ***Nocare****, where an algorithm decides this promise problem, if it returns* 1 *on inputs* $x \in$ ***Yes****,* 0 *on inputs* $x \in$ ***No*** *and has arbitrary behaviour on inputs from* ***Nocare****.*

Although debatable from a real-life implementation perspective, cryptographers and complexity theorists usually consider algorithms as efficient, if they run in time that is

polynomial in the length of their input. Cryptographers here usually consider probabilistic algorithms whose time is strictly bounded by a polynomial on all input instances, while complexity theorists often consider deterministic algorithms with this property.

Definition 6. *A language $\mathcal{L}$ is decidable in deterministic polynomial-time, denoted $\mathcal{L} \in \mathcal{P}$, if there is a deterministic polynomial-time Turing-Machine M such that $M(x) = 1$ if and only if $x \in \mathcal{L}$.*

While classical complexity theory often deals with the aforementioned deterministic algorithms, probabilism is a crucial element of cryptography. For example, it is well-known that secure encryption requires randomness. For most of this thesis, we thus use probabilistic-polynomial-time algorithms that run in strict polynomial-time.

Definition 7. *An (interactive) randomized algorithm M is a probabilistic polynomial-time algorithm (PPT), if for all its random strings and all possible interactions, its overall running time is strictly bounded by a polynomial in its (first) input. The running time of an interactive algorithm is defined as the sum over the running times the algorithm has each time it is invoked.*

Non-deterministic polynomial-time is the set of languages that can be easily decided by an algorithm that, for each input, may get an additional string that helps certifying that a string in the language. Consider the language of theorem statements that are provable (by a proof of fixed polynomial length). Given a theorem statement, a certificate for this theorem statement to be in the language would be a proof that the theorem is true. Also the image of a pseudorandom generator G forms an $\mathcal{NP}$-languages, as one can prove that a string y is in the image of G by giving a pre-image x as a witness, i.e., $G(x) = y$.

Definition 8. *We say that $\mathcal{L}$ is decidable in non-deterministic polynomial-time or $\mathcal{L} \in \mathcal{NP}$, if there is a deterministic polynomial-time machine M and a polynomial q such that*

$$\mathcal{L} = \{x | \exists z : |z| \leq q(|x|) \text{ and } M(x; z) = 1\}.$$

There are inherent relations between cryptography and complexity that we will analyse in detail in Chapter 7. One example is, that the aforementioned language defined by the image of a pseudo-random generator has to be a hard language in $\mathcal{NP}$, i.e., is not solvable in polynomial-time, if the pseudo-random generator is secure.

We considered deterministic algorithms, non-deterministic algorithms and probabilistic algorithms. An interesting class of problems emerges when we combine probabilism and non-determinism. For example, we can consider what type of problems a PPT algorithm can solve when given additional witnesses. This type of problems is known als Arthur-Merlin $\mathcal{AM}$ and is a generalization of $\mathcal{NP}$, as each deterministic polynomial-time algorithm is also a PPT algorithm.

Definition 9. *We say that a language $\mathcal{L}$ is in Arthur-Merlin $\mathcal{AM}$, if there is a deterministic polynomial-time machine M and fixed polynomials q and R such that for all*

$x \in \mathcal{L}$, it holds that

$$\text{Prob}_{r \leftarrow \{0,1\}^{R(|x|)}} [\exists z_r, |z_r| \leq q(|x|), M(x, r; z_r) = 1] \geq 1 - 2^{-|x|}$$

and for all $x \notin \mathcal{L}$, one has

$$\text{Prob}_{r \leftarrow \{0,1\}^{R(|x|)}} [\exists z_r, |z_r| \leq q(|x|), M(x, r; z_r) = 1] \leq 2^{-|x|}.$$

We say that a function is *co-range-verifiable* if the complement of its range is in $\mathcal{AM}$.

Polynomial Hierarchy and Relativized Complexity Classes. A relativized complexity class is defined as its unrelativized version, except that the machines are enhanced via an oracle. Querying this oracle is considered as only one step of computation.

Definition 10. *An oracle (Turing) machine M is said to run in time t, if for all inputs x and all oracles O, its running time is upper bounded by $t(|x|)$.*

Relative to an oracle O, the complexity class of polynomial-time algorithms $\mathcal{P}$, written as $\mathcal{P}^O$, is defined as follows.

Definition 11. *We say that $\mathcal{L}$ is decidable in deterministic polynomial-time relative to O, or that $\mathcal{L} \in \mathcal{P}^O$, if there is a deterministic polynomial-time oracle machine M and a polynomial q such that*

$$\mathcal{L} = \{M^O(x) = 1\}.$$

We write non-deterministic polynomial-time relative to O as $\mathcal{NP}^O$ and use the following definition.

Definition 12. *We say that $\mathcal{L}$ is decidable in non-deterministic polynomial-time relative to O, or that $\mathcal{L} \in \mathcal{NP}^O$, if there is a deterministic polynomial-time oracle machine M and a polynomial p such that*

$$\mathcal{L} = \{x | \exists z : |z| \leq q(|x|) \text{ and } M^O(x; z) = 1\}.$$

Likewise, the relativied complexity class $\mathbf{co}\mathcal{NP}^O$ is defined as the set of all languages whose complement is in $\mathcal{NP}^O$.

Definition 13. *We say that $\mathcal{L}$ is decidable in non-deterministic polynomial-time relative to O, or that $\mathcal{L} \in \mathcal{NP}^O$, if there is a deterministic polynomial-time oracle machine M and a polynomial q such that*

$$\mathcal{L} = \{x | \exists z : |z| \leq q(|x|) \text{ and } M^O(x; z) = 1\}.$$

Similarly, we can define relativied Arthur-Merlin.

Definition 14. *We say that a language $\mathcal{L}$ is in Arthur-Merlin relative to O, written $\mathcal{AM}^O$, if there is a deterministic polynomial-time oracle machine M and fixed polynomials q and R such that for all $x \in \mathcal{L}$, it holds that*

$$\text{Prob}_{r \leftarrow \{0,1\}^{R(|x|)}} \left[\exists z_r, |z_r| \leq q(|x|), M^O(x, r; z_r) = 1\right] \geq 1 - 2^{-|x|}$$

and for all $x \notin \mathcal{L}$, one has

$$\text{Prob}_{r \leftarrow \{0,1\}^{R(|x|)}} \left[\exists z_r, |z_r| \leq q(|x|), M^O(x, r; z_r) = 1\right] \leq 2^{-|x|}.$$

We can now define the polynomial hierarchy. The polynomial hierarchy is defined inductively by $\Sigma_0 := \Pi_0 := \Delta_0 := \mathcal{P}$ and then, for all $i \geq 0$, Σ_{i+1} is defined as $\mathcal{NP}$ where the oracle is defined by the level below, that is

$$\Sigma_{i+1} := \mathcal{NP}^{\Sigma_i}.$$

Similarly, Π_{i+1} is relativized $\mathbf{co}\mathcal{NP}$, that is

$$\Pi_{i+1} := \mathbf{co}\mathcal{NP}^{\Sigma_i}.$$

And Δ_{i+1} is relativized $\mathcal{P}$:

$$\Delta_{i+1} := \mathcal{P}^{\Sigma_i}.$$

Note that for all definitions, it is equivalent whether we choose Σ_i or Π_i as the oracle. It also holds that for all $i \geq 0$, Δ_i is contained in Π_i and Σ_i, and that Π_i and Σ_i are contained in Δ_{i+1}.

We believe that the polynomial hierarchy does not collapse, i.e., that all levels of the polynomial hierarchy are different, in particular, that $\mathcal{NP}$ is not contained in $\mathcal{P}$. If, at any level, it holds that $\Delta_i = \Sigma_i$, then the polynomial hierarchy collapses to that level, i.e., for all $j > i$, it holds that $\Sigma_j = \Pi_j = \Delta_j = \Delta_i$. Note that we know that Δ_i is contained in $\Pi_i \cap \Sigma_i$, but we do not know whether this containment is strict. If it turns out that $\Delta_i = \Pi_i \cap \Sigma_i$, then it is unknown wether this would cause the polynomial hierarchy to collapse. In particular, at the lowest level of the hierarchy, if $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ were contained in $\mathcal{P}$, then we do not know whether this causes the polynomial hierarchy to collapse or not. However, as well-studied problems such as factoring are contained in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ and underly the security of the famous RSA encryption scheme, we all hope that indeed, $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is not contained in $\mathcal{P}$.

Chapter 3

Compositional Framework for Games

Games are a standard modelling approach for security of schemes. In such a game, an arbitrary adversary interacts with the algorithms that define the protocol, via a set of queries that capture the use of the protocol in a real system. The adversary sends queries to the game, which computes responses using the algorithms under attack. The adversary tries to trigger an event which the game deems bad. In the previous chapter, we have seen examples of well-known games that capture security of message authentication codes (MAC), encryption as well as the hardness of the Diffie-Hellman assumption. In this chapter, we give a general abstract definition for cryptographic games and specialise it in two ways, namely for *protocols* and for *primitives*. Both formalisms reflect the same basic idea but differ in some aspects (*e.g.* protocols need explicit notions of users and sessions).

We then show how to cast the standard security definitions for primitives such as MACs and encryption in our framework as well as a security definition for a protocol implementing authenticated channels. Finally, we explain how the well-established Bellare-Rogaway (BR) security game for key exchange is an instance of our abstract framework.

This framework has been developed in joint work with Marc Fischlin, Nigel Smart, Bogdan Warinschi and Stephen Williams. Its ideas have been reflected in [BFWW11, BFS+13, Wil11b], although sometimes in a different syntax. We here give a unified presentation with minimal syntax.

Before going into the details of the definition, let us have a look at the big picture. Ultimatively, we desire to define a game that reflects that first, a key exchange protocol ke is run to derive symmetric keys and then, a symmetric key protocol π is run that uses the symmetric keys. Now, given a game G_{ke} that defines the attack model of the key exchange protocol, ke, and given the game G_π that captures the security of the symmetric key protocol, π, we generically define the execution of the composed protocol, and a game $G_{\mathsf{ke};\pi}$, modelling its security. In the composed protocol, each session first runs an instance of the key exchange protocol and then uses the derived key to execute

the symmetric key protocol while not using any other information from the key exchange stage—in turn, the adversary can access information from both stages. Note that, as common for key exchange, the game $G_{\mathsf{ke};\pi}$ allows the adversary to interact with ke and π concurrently and simultaneously: at any given point some sessions may be in the key exchange stage, while others are in the symmetric key protocol stage. The security requirement on the composition is inherited from G_π: the adversary wins against the composition if it breaks the symmetric key protocol. The game $G_{\mathsf{ke};\pi}$ does not place any explicit security requirement on the key exchange protocol as would be the case in the Bellare-Rogaway model.

Hence, the ultimate goal of this section is to define a model for composed protocols of a key exchange and an application protocol. However, along the way, we will introduce a useful framework for games that might be of independent interest hence the more general sytax that we introduce next.

3.1 Cryptographic Games

We do not enforce a particular syntax for primitives/protocols and take a general approach where we only explicitly identify a key-generation algorithm. A primitive ζ / protocol π is then given by a pair of algorithms $(\mathsf{kg}_\zeta, \mathsf{P}_\zeta)/(\mathsf{kg}_\pi, \mathsf{P}_\pi)$, where $\mathsf{kg}_{\zeta/\pi}$ is a randomized key generation algorithm taking as input the security parameter and outputting keys from some key space. The algorithm P_ζ is the algorithm that defines the primitive such as a particular encryption and decryption algorithm. Similarly, the algorithm P_π is executed locally by a party that executes the protocol, for example to compute the next message. Note that defininining primitives/protocols via a single algorithm is without loss of generality since several algorithms can be emulated by a single one, as long as the inputs are tagged to indicate for which of the underlying algorithms the input is intended.

We now first introduce those aspects of games that are identical for primitives and protocols, and then specialize our notions for primitives and protocols. For the remainder of this section, the term primitive refers to both, primitives and protocols.

The security of a primitive is captured by one (or more) games; where a game G_ζ for the primitive ζ is an interactive probabilistic Turing machine. The machine has input tapes G_ζ^{in} and $G_\zeta^{\mathsf{k\text{-}in}}$ to receive queries and keys, respectively, and one output tape G_ζ^{out}. The game takes as input a security parameter 1^η and allows the adversary access to various queries. The adversary drives the execution by writing queries (from some finite set) on G_ζ's standard input tape G_ζ^{in}. The game calculates a response and updates its internal state; the response is returned to the adversary. Notice that these calculations may involve the algorithm P_ζ, but we do not explicitly say how this is done. The execution is randomized as both the adversary and game may use random coins. Keys for the game are written on the input tape $G_\zeta^{\mathsf{k\text{-}in}}$. Keys may come from the key generation of the primitive, but can also come from somewhere else, such as a key exchange protocol. See later for how this tape is used.

We require that when the execution of the adversary terminates, there is a single bit

written on G_ζ^{out}, which is the outcome of the game, where 1 indicates that the adversary has succeeded in his goals. We write $\mathsf{Exec}(G_\zeta, \mathcal{A})(1^\eta)$ for the random variable that describes the output of the game when interacting with adversary $\mathcal{A}$ for security parameter 1^η. Naturally, $\mathsf{Exec}(G_\zeta, \mathcal{A})(1^\eta)$ also depends on the distribution of keys provided to $G_\zeta^{\mathsf{k\text{-}in}}$. If not specified, we assume that $\mathcal{A}$ provides those. We go into detail later.

3.2 Games for Cryptographic Primitives

We now refine the above general definition of games. First, we add a mechanism to explicitly maintain keys and related information (*e.g.* whether a key is corrupt). The game G_ζ maintains an internal list $\mathcal{L}_G$ consisting of tuples $(\mathsf{kid}, k, \mathrm{st}_{\mathsf{kid}})$; where kid is an administrative key identifier, k is the key corresponding to this identifier and $\mathrm{st}_{\mathsf{kid}} \in \{\mathsf{honest}, \mathsf{corrupted}\}$. We describe two queries that make use of the list $\mathcal{L}_G$; the NewKey and $\mathsf{Corrupt}$ queries which the game answers as follows:

- $\mathsf{NewKey}()$: Prior to making this query, $\mathcal{A}$ writes some value on the $G_\zeta^{\mathsf{k\text{-}in}}$ input tape of G_ζ; possibly the output of the primitive's key generation algorithm kg_ζ. The call $\mathsf{NewKey}()$ makes the game G_ζ obtain a new key k by reading its $G_\zeta^{\mathsf{k\text{-}in}}$ input tape. The game checks whether k has been seen previously by searching for an existing tuple $(\mathsf{kid}', k, \mathrm{st}_{\mathsf{kid}})$ containing the key k. If such a tuple exists then the value kid' is returned to the adversary. Otherwise it instantiates a new "session" of the primitive, keyed with k, by generating a new key identifier kid and adding the tuple $(\mathsf{kid}, k, \mathsf{honest})$ to the list $\mathcal{L}_G$. The value kid is returned to the adversary.
- $\mathsf{Corrupt}(\mathsf{kid})$: If there is a tuple $(\mathsf{kid}, k, \mathrm{st}_{\mathsf{kid}})$ on the list $\mathcal{L}_G$ then $\mathrm{st}_{\mathsf{kid}}$ is set to $\mathsf{corrupted}$ and k is returned to the adversary. The adversary may not interact with a session once it is corrupt. If no such tuple exists then the query is ignored.

Definition 15 (Primitive Game)**.** *A primitive game G_ζ for the primitive ζ is a cryptographic game with a set of queries that includes the two special queries* NewKey *and* $\mathsf{Corrupt}$*, and maintains a list $\mathcal{L}_G$ as defined above.*

See Section 3.4 for examples of games related to IND-CCA encryption and MAC security.

In the above definition the adversary is allowed to set keys for the game so security is impossible to guarantee (and indeed, we do not attempt to do so). We recover standard notions of security by restricting the adversary in certain ways. We present three (increasingly) stronger restrictions, the last yielding standard notions of security. We explicitly delineate the two intermediate classes since they are useful for technical reasons.

Definition 16 (Split Adversary)**.** *An adversary $\mathcal{A}$ against a cryptographic game G is a* split adversary *if it consists of two subadversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, such that $\mathcal{A}_1$ makes only certain types of queries to G, and $\mathcal{A}_2$ makes other types of queries of queries to G. The algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ may communicate as they wish. By convention we assume that A_2 is in charge of scheduling the execution.*

Since there are no restrictions on how the two subadversaries communicate splitting an adversary does not change its overall functionality. Next, we restrict the flow of information between the two subadversaries and the queries that each adversary makes to obtain standard security requirements. A *query-respecting* adversary and a *key-benign* adversary. The query-respecting adversary is a split adversary where only the first part of the adversary is allowed to create keys.

Definition 17 (Query-Respecting Adversary for Primitives)**.** *A split adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *against a primitive game* G_ζ *is* query-respecting *if it satisfies the following restrictions:*

- *The query* $\mathsf{NewKey}()$ *is only made by* $\mathcal{A}_1$*.*
- *Only* $\mathcal{A}_1$ *writes keys to the key input tape of* G_ζ*.*
- *Both parts* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are allowed to make* $\mathsf{Corrupt}$ *queries.*
- $\mathcal{A}_2$ *makes all other queries.*

Finally, a key-benign adversary is additionally restricted to only initialize instances of primitives with keys honestly produced via their associated key generation algorithm. In addition the adversary "forgets" the values of these keys (but not maintaining state across invocations). Specifically, we consider the class of split adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where we restrict the information passed from $\mathcal{A}_1$ to $\mathcal{A}_2$.

Definition 18 (Key-Benign Adversary for Primitives)**.** *For a game* G_ζ *of a primitive* $\zeta = (\mathsf{kg}_\zeta, \mathsf{P}_\zeta)$ *and a split adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, we say that* $\mathcal{A}$ *is key-benign with respect to* kg_ζ *if it behaves as follows:*

- *Adversary* $\mathcal{A}$ *is query-respecting.*
- *Subadversary* $\mathcal{A}_2$ *only sends the message* $\mathsf{NewKey}()$ *to* $\mathcal{A}_1$*.*
- *Each time* $\mathcal{A}_1$ *is activated by receiving a message* $\mathsf{NewKey}()$ *from* $\mathcal{A}_2$*, it runs the key generation algorithm* kg_ζ *once, writes the output of this algorithm on the input tape* $G^{\mathsf{k\text{-}in}}$ *of* G *and makes a* $\mathsf{NewKey}()$ *query to the primitive game. The game then returns a key identifier* kid *that* $\mathcal{A}_1$ *passes to* $\mathcal{A}_2$*.*
- *No other information is passed from* $\mathcal{A}_1$ *to* $\mathcal{A}_2$*.*

We stress that per our convention in Defintion 16 adversary $\mathcal{A}_2$ drives the execution: either it queries the game directly and is given the answer, or creates a new key via $\mathcal{A}_1$. The control is always returned to $\mathcal{A}_2$. The behaviour of $\mathcal{A}_1$ is fully specified and thus does not allow for constructing covert channels between $\mathcal{A}_1$ and $\mathcal{A}_2$. Standard security notions are obtained by restricting to adversaries which are key-benign. A security notion for a protocol is a pair (G_ζ, δ) with G_ζ a game as described above and δ an error probability (a probability with which the adversary can certainly win the game). A protocol is secure if no key-benign adversary can win the game with probability significantly better than δ (typically $\delta = \frac{1}{2}$ or $\delta = 0$).

Definition 19 ((G_ζ, δ)-Secure Primitive)**.** *We say that a primitive* $\zeta = (\mathsf{kg}_\zeta, \mathsf{P}_\zeta)$ *is* (G_ζ, δ)-secure*, or equivalently that it satisfies* (G_ζ, δ)*, if for any probabilistic polynomial-time algorithm* $\mathcal{A}$ *that is key-benign with respect to the key generation algorithm* kg_ζ *it*

holds that

$$\Pr\left[\mathsf{Exec}(G_\zeta, \mathcal{A})(1^\eta) = 1\right] - \delta$$

is a negligible function in the security parameter.

For clarity, sometimes we write $\mathsf{Exec}(G_\zeta, \mathcal{A} : \mathsf{kg}_\zeta)(1^\eta)$ for the execution of the game with a key-benign adversary with respect to kg_ζ.

3.3 Games for (Two-Party) Cryptographic Protocols

In this section we extend the above framework to games for two-party protocols. The main difference is that we introduce users and protocol sessions into the formalism. Recall, we make no assumptions on the syntax of protocols and assume that a protocol π is given by two algorithms, $\pi = (\mathsf{kg}_\pi, \mathsf{P}_\pi)$, where again the first algorithm is for generating keys, and the latter defines the execution of the protocol itself. We give general games for the security of protocols, but specialize them for the case when the protocols are based on long-term symmetric keys; in Subsection 3.5.1, we describe the alterations needed when dealing with long-term public keys.

IDENTITIES. We fix an integer n_i of size polynomial in the security parameter to restrict the size of the set of all users $\mathcal{U}$. Identities, used to model the users of a system, are then identified by an element U in set $\mathcal{U}$.

SESSIONS. Local sessions of a protocol are identified by *labels* label in a set LABELS. We can think of LABELS as $\mathcal{U} \times \mathcal{U} \times \mathbb{Z}$, where the label $\mathsf{label} = (U, V, k)$ refers to the k-th local session of the identity U, where the intended partner identity is V. In this way, identifiers are assured to be globally unique. These labels are only for bookkeeping in the communication between the game and the adversary. In particular, they allow the adversary to uniquely identify each session within the game and are not used by the protocol.

As part of its internal state, the game maintains a list $\mathcal{L}_G$ of tuples of the form $(\mathsf{label}, \mathsf{kid}, U, V)$, that can be augmented with additional variables when needed. Each such tuple corresponds to a local session of a user U with intended partner V. The entry kid is the key identifier for the key used by the owner of the session label. Both label and kid are only administrative identifiers which are not used within the protocol. The key corresponding to kid could be a shared password, a long-term key, or a key derived through a key exchange protocol. We use the notation $\mathsf{label.kid}$ to denote, for example, the key identifier stored in the list entry starting with label. As label is globally unique, $\mathsf{label.kid}$ is well-defined. We also use the notation $\mathsf{label} \in \mathcal{L}_G$ to denote that there is a list entry in $\mathcal{L}_G$ with label.

The game keeps track of the actual values for the keys, as well as the identities associated to these keys; recall that we work here in the symmetric key setting where such keys are shared by parties. This is done via a list $\mathcal{L}_G^{\mathsf{keys}}$ whose entries are of the form $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$, where kid is a (globally unique) key identifier, k is the actual value for the key, U and V are the identities associated to this key and $\mathrm{st}_{\mathsf{kid}} \in \{\mathsf{honest}, \mathsf{corrupted}\}$. As before, keys are passed to the game by the adversary via the input tape $G_\pi^{\mathsf{k\text{-}in}}$.

The behaviour of the game G_π is determined by the function that defines the protocol P_π and, as for primitives, we do not fully specify this dependency. It is worth noting however that a typical game maintains the state of the various local sessions, directs the queries to the appropriate sessions, updates the local state, and returns an answer to the adversary.

We now detail the particular mechanism that our games use to start sessions and provide keys to such sessions. We informally discuss the queries that implement the mechanism and their use, and then give a more formal description.

A query $\mathsf{NewKey}(U, V)$ allows the adversary to "register" the key written on the input key tape with the game (a key identifier kid is returned). As this is a local process, via query $\mathsf{SameKey}(V, U, \mathsf{kid})$, the same key can be registered for user V. A new session of the protocol run by U, with intended partner V, is started via a query $\mathsf{NewSession}(U, V, \mathsf{kid})$: the key used by U in this session is the one indexed by kid. We may then start a session of V with the same key. Note that keys tie two sessions of two users together; this is a security property that we will require from any key exchange protocol used to derive keys for π.

Formally, we require protocol games to allow the following special queries:

- $\mathsf{NewKey}(U, V)$: The game G_π reads a new key k off the $G_\pi^{\mathsf{k\text{-}in}}$ tape, generates a new identifier kid and creates a new tuple $(\mathsf{kid}, U, V, k, \mathsf{honest})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$. The key identifier kid is returned to the adversary.
- $\mathsf{SameKey}(U, V, \mathsf{kid})$: If there is a tuple $(\mathsf{kid}, V, U, k, \mathrm{st}_{\mathsf{kid}})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$, the tuple $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ is added to $\mathcal{L}_G^{\mathsf{keys}}$ and kid is returned to the adversary. Else, the game returns $\perp$.
- $\mathsf{NewSession}(U, V, \mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for a tuple $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ and aborts if no such tuple exists. Else, it generates a new identifier label, creates the tuple $(\mathsf{label}, \mathsf{kid}, U, V)$ on the list $\mathcal{L}_G$ and returns label to the adversary.
- $\mathsf{Corrupt}(\mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for all entries of the form $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ and does nothing if no such entry exists. Otherwise, for all such entries, it sets $\mathrm{st}_{\mathsf{kid}} = \mathsf{corrupted}$ and returns k to the adversary. No further queries are allowed to a corrupted session.

By slightly modifying the above definitions one can easily model public-key protocols, this is detailed in Section 3.5.1.

Definition 20 (Protocol Game)**.** *A protocol game G_π for $\pi = (\mathsf{kg}_\pi, \mathsf{P}_\pi)$ is a cryptographic game with a set of queries that includes the special queries $\mathsf{NewKey}(U, V)$, $\mathsf{SameKey}(U, V, \mathsf{kid})$, $\mathsf{NewSession}(U, V, \mathsf{kid})$ and $\mathsf{Corrupt}(\mathsf{kid})$. The game G_π maintains a list $\mathcal{L}_G$ and a list $\mathcal{L}_G^{\mathsf{keys}}$ as defined above.*

As before, we write $\mathsf{Exec}(G_\pi, \mathcal{B})(1^\eta)$ for the random variable that describes the output of the game when interacting with adversary $\mathcal{B}$ for security parameter 1^η. We adapt the notions of query-respecting and key-benign adversaries from primitives to the case of protocols.

Definition 21 (Query-Respecting Adversary for Protocols)**.** *A split adversary $\mathcal{B} =$*

$(\mathcal{B}_1, \mathcal{B}_2)$ *against a protocol game* G_π *is* query-respecting *if it satisfies the following restrictions:*

- *The query* NewKey *is only made by* $\mathcal{B}_1$ *who has written some value to the tape* $G_\pi^{\mathsf{k\text{-}in}}$.
- *The query* $\mathsf{SameKey}(U, V, \mathsf{kid})$ *is only made by* $\mathcal{B}_1$. *Moreover if a query* $\mathsf{NewKey}(U, V)$ *previously returned some* kid *then* $\mathcal{B}_1$ *is allowed at most one* $\mathsf{SameKey}(V, U, \mathsf{kid})$ *query and no* $\mathsf{SameKey}(U, V, \mathsf{kid})$ *query.*
- *The query* $\mathsf{NewSession}(U, V, \mathsf{kid})$ *is only made by* $\mathcal{B}_2$.
- *Both,* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *are allowed the query* $\mathsf{Corrupt}(\mathsf{kid})$.
- *All other queries are made by* $\mathcal{B}_2$.

The second requirement in the above definition ensures that for adversaries that write different key values on the key input tape of G_π at most two protocols sessions of any pair of users U, V use the same key.

Definition 22 (Key-Benign Adversary for Protocols)**.** *For a game* G_π *of a protocol* $\pi = (\mathsf{kg}_\pi, \mathsf{P}_\pi)$ *and a split adversary* $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$, *we say that* $\mathcal{B}$ *is key-benign with respect to* kg_π *if it behaves as follows.*

- *Adversary* $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ *is query-respecting.*
- *The message sent from* $\mathcal{B}_2$ *to* $\mathcal{B}_1$ *is of the form* $\mathsf{NewKey}(U, V)$ *or of the form* $\mathsf{SameKey}(U, V, \mathsf{kid})$.
- *Each time,* $\mathcal{B}_1$, *receives a message* $\mathsf{NewKey}(U, V)$ *from* $\mathcal{B}_2$, *it runs the key generation algorithm* kg *once, writes the output of this algorithm on the input tape* $G^{\mathsf{k\text{-}in}}$ *of* G *and makes a* $\mathsf{NewKey}(U, V)$ *query to the protocol game. The game then returns a key identifier* kid *that* $\mathcal{B}_1$ *passes to* $\mathcal{B}_2$.
- *Each time,* $\mathcal{B}_1$, *receives a message* $\mathsf{SameKey}(U, V, \mathsf{kid})$ *from* $\mathcal{B}_2$, *it makes a* $\mathsf{SameKey}(U, V, \mathsf{kid})$ *query to the protocol game. The game then returns a key identifier* kid *that* $\mathcal{B}_1$ *passes to* $\mathcal{B}_2$.
- *No other information is passed from* $\mathcal{B}_1$ *to* $\mathcal{B}_2$.

A security notion for a protocol is then a pair (G_π, δ) with G_π a game as described above and δ an error probability (a probability with which the adversary can certainly win the game). A protocol is secure if no key-benign adversary can win the game with probability significantly better than δ.

Definition 23 ((G_π, δ)-Secure Protocol)**.** *We say that a protocol* $\pi = (\mathsf{kg}_\pi, \mathsf{P}_\pi)$ *is* (G_π, δ)-secure, *if for any probabilistic polynomial-time algorithm* $\mathcal{B}$ *key-benign with respect to the key generation algorithm* kg_π, *it holds that*

$$\Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B})(1^\eta) = 1\right] - \delta$$

is a negligible function in the security parameter.

When the game for the protocol is clear from the context, we may simply say that the protocol is δ-secure instead of (G_π, δ)-secure. The same simplification applies for primitives. Below in Section 3.4 we provide an example of a game for security of authenticated channel protocols.

3.4 Examples for Primitive and Protocol Games

In this section we examine how our previous formalism for primitive and protocol games applies to some well known examples.

3.4.1 Primitive Games

The following examples show how the standard games for defining multi-user IND-CCA security of encryption schemes and multi-user UF-CMA for message authentication codes can be easily cast as instances of the our framework for primitive games. The only difference is the addition of the Corrupt query, which is a simple extension, but needed to be able to cope with security of protocols using this primitive which allow adaptive session-state corrupts.

EXAMPLE: IND-CCA ENCRYPTION. We now describe the game for symmetric key based IND-CCA encryption in the multi-user setting [BBM00] using this language. We assume the scheme is given by the algorithms $\mathsf{KeyGen}(1^\eta)$, $\mathsf{Enc}_k(m)$, $\mathsf{Dec}_k(c)$), so that $\mathsf{kg}_\zeta = \mathsf{KeyGen}(1^\eta)$. Execution begins with the game selecting a random bit $b \leftarrow \{0,1\}$. The key-benign adversary can now make the following queries, in addition to the NewKey and Corrupt queries, as follows:

- $\mathsf{Enc}(\mathsf{kid}, m)$ – The game computes the encryption $c \leftarrow \mathsf{Enc}_k(m)$, where k is the key in the tuple in $\mathcal{L}_G$ corresponding to kid. The ciphertext c is then passed back to the adversary. If no such tuple exists then this operation does nothing.
- $\mathsf{Challenge}(m_0, m_1, \mathsf{kid})$ – The game computes the challenge ciphertext $c^\dagger \leftarrow \mathsf{Enc}_k(m_b)$ as above and returns $c^\dagger$ back to the adversary. Note that it is required $|m_0| = |m_1|$.
- $\mathsf{Dec}(\mathsf{kid}, c)$ – The game computes the decryption $m \leftarrow \mathsf{Dec}_k(c)$, where again k is the key in the tuple in $\mathcal{L}_G$ corresponding to kid. The value of m is passed back to the adversary.
- $\mathsf{Guess}(b')$ – The game outputs 1 if $b' = b$, otherwise 0 is output. Execution of the game terminates.

We make the following two restrictions on the queries, so as to make sure that the game cannot be trivially won:

- On calling $\mathsf{Dec}(\mathsf{kid}, c)$ if c was the output of some call to Challenge for *this value* of kid then the game aborts outputting zero.
- The adversary may not make a $\mathsf{Corrupt}(\mathsf{kid})$ query if it has made a $\mathsf{Challenge}(\cdot, \cdot, \mathsf{kid})$ query for the same value of kid, and vice-versa.

Note that the query $\mathsf{Enc}(\mathsf{kid}, m)$ can be simulated via a call to $\mathsf{Challenge}(m, m, \mathsf{kid})$, however we keep a separate query of $\mathsf{Enc}(\mathsf{kid}, m)$ so as to make the above restrictions simpler to define.

Since the adversary can guess the value of b with probability $1/2$, we require that this game is key benign secure for $\delta = 1/2$. Notice that security clearly depends on what key generation algorithm is allowed to write to the $G_\zeta^{\mathsf{k\text{-}in}}$ tape. For example if kg_ζ consisted of sampling from the set of l-bit strings uniformly at random then we would obtain the standard security notion for IND-CCA encryption with a cipher of l-bit keys. On the other hand kg_ζ could consist of simply outputting the same l-bit string, since

the adversary is assumed to know the code of kg_ζ, such an algorithm would always be insecure. We see therefore that the definition of a kg_ζ is always implicit in any security notion, we have simply brought it more to the fore.

EXAMPLE: EF-CMA MAC. Now suppose that the primitive we wish to model is a MAC, given by a triple of algorithms $(\mathsf{KeyGen}(1^\eta), \mathsf{Mac}_k(m), \mathsf{Verify}_k(m,\sigma))$. We want to test whether this primitive is secure against a chosen message attack, where the adversary is trying to create a forgery for any message (ie. existential forgeability). We now detail how this execution proceeds within our model. When execution of the game begins, the key benign adversary makes a number of queries to the game. In addition to NewKey and Corrupt queries he can make the following queries:

- $\mathsf{Mac}(m, \mathsf{kid})$ – The game computes $\sigma = \mathsf{Mac}_k(m)$, where k is the key in the tuple in $\mathcal{L}_G$ containing kid. The game returns σ to the adversary.
- $\mathsf{Verify}(\mathsf{kid}, m, \sigma)$ – Here the game computes the boolean value $\tau = \mathsf{Verify}_k(m,\sigma)$ for k corresponding to kid in $\mathcal{L}_G$. If $\tau = \mathsf{true}$ *and* m has never been queried to $\mathsf{Mac}(m, \mathsf{kid})$ *and* $\mathsf{Corrupt}(\mathsf{kid})$ has not been called then the game outputs 1 and terminates. Otherwise the value τ is returned to the adversary.

If the game does not terminate because of the result of a Verify query, eventually the adversary terminates and the game writes 0 to its output tape. We say the scheme is key-benign EF-CMA secure if the above game is δ-secure for $\delta = 0$.

3.4.2 Protocol Games

Here we look at the specific example of a protocol which provides authenticated channels. In order to model an authenticated channels scheme we must first decide upon a game based definition to capture the requirements of an authenticated channel. An authenticated channel has a number of desired properties. The first is that one must be able to verify messages are sent by someone who possesses the shared secret key. The second property is that messages are only accepted if they are received in order and where duplicates are rejected. We now describe a game to formally capture these notions.

The adversary is able to make call to NewKey, SameKey, NewSession and Corrupt as previously described, the only difference here is, that for each kid, at most one call to $\mathsf{NewSession}(U, V, \mathsf{kid})$ and $\mathsf{NewSession}(V, U, \mathsf{kid})$ is allowed, as authenticated channels shall preserve communication between two sessions. Thus, the property is trivially broken, if several sessions may use the same key.

The game maintains two sets of "append only" lists, $\iota = \{\iota_{\mathsf{label}}\}$ and $\theta = \{\theta_{\mathsf{label}}\}$, where each entry corresponds to a separate value of label. The adversary then interacts with the protocol via the following queries made available via the game:

- $\mathfrak{m} \leftarrow \mathsf{Init}(m_{\mathsf{plain}}, \mathsf{label})$ – The message m_{plain} is appended to the list ι_{label} corresponding to the entry $(\mathsf{label}, \mathsf{kid}, U, V)$. The oracle responds with the (authenticated) protocol message, $\mathfrak{m}$, which is intended to be sent to party V with session identifier sid. It is assumed that $\mathfrak{m}$ contains the message m_{plain} as a subsequence. The value of $\mathfrak{m}$ is appended to the list θ_{label}.

- $m_{\mathsf{plain}} \leftarrow \mathsf{Send}(\mathfrak{m}, \mathsf{label}')$ – The protocol message $\mathfrak{m}$ is passed to session label' as though it was a message received through the authenticated channel. The message $\mathfrak{m}$ is appended to the $\iota_{\mathsf{label}'}$ list. If the protocol message authenticates correctly then the message m_{plain} is appended to the $\theta_{\mathsf{label}'}$ list.

At some stage the adversary $\mathcal{B}$ will terminate its execution with the game. At any point during execution, the game G checks that, for all parties (U, V) with $(\mathsf{label}_1, \mathsf{kid}, U, V)$, $(\mathsf{label}_2, \mathsf{kid}, V, U) \in \mathcal{L}_G$, that $\theta_{\mathsf{label}_2}$ is a subsequence of ι_{label_1} when $\mathsf{sid} \neq \perp$ and the entries $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathrm{key}_1})$, $(\mathsf{kid}, V, U, k, \mathrm{st}_{\mathrm{key}_2}) \in \mathcal{L}_G^{\mathsf{keys}}$ have $\mathrm{st}_{\mathrm{key}_1} = \mathrm{st}_{\mathrm{key}_2} \neq \mathsf{corrupted}$. If there exists any pair where this condition is not satisfied then the adversary $\mathcal{B}$ has won the game and so the game immediately writes 1 to the output tape of the Turing machine G. Otherwise, it outputs 0, when the adversary terminates. An instantiation of an authenticated channel is called secure, if it is δ-secure with $\delta = 0$.

3.5 Games for Key Exchange Protocols

A key exchange protocol allows two local sessions, which use long term keys of identities, to agree upon a short term session key. The game for key exchange protocols captures the typical execution model of a key exchange protocol, where an adversary can run multiple sessions, mediates all communication, and is allowed to corrupt various keys in the system.

To define the game for key exchange, we specialize the generic two-party protocol game definition given in Section 3.3. As the generic definition only applies to symmetric long-term keys, below in Section 3.5.1 we provide a minor extension to allow asymmetric long-term keys. In order to "partner" two sessions we use the notion of a session identifier value. This value is computed by the key exchange protocol. Using a session identifier still allows one to base partnering on notions such as matching conversation as done by Bellare et al. [BR93]; using the message transcript one can achieve a similar, while not equivalent notion. Partnered sessions are required to compute the same session key, and this session key must be indistinguishable from random. Further, as we consider two party protocols at most two sessions should ever share the same session identifier.

The session identifier is distinct to the local session identifier label. The former is computed by the key exchange algorithm to determine which sessions are partners, whilst the latter is simply a unique label for the adversary to address queries to a particular session.

We assume that when a key exchange protocol session agrees upon or rejects a key, the adversary knows this has taken place. We require this property explicitly, but one can see that in the models of [BR93, BWJM97], by sending a 'Reveal' query after every 'Send' query it is possible for an adversary to learn when sessions accept or reject a key.

To model the above requirements of a key exchange protocol we extend the above definition and introduce two security games. The first requirement, secrecy, is modelled using the methods of Bellare-Rogaway security: the adversary chooses a session of the game and receives either a random key or the real session key agreed. It wins the game if it determines correctly with which it was provided. We distinguish between protocols

that are forward secure and those which are not forward secure. Here, forward security means that if a long term key is corrupted by an adversary, any session keys (including those computed using the corrupted long term key) already agreed will still be considered secure, see Section 4.1 and Chapter 4.6 for more details on forward-secure key exchange protocols.

The second security game places restrictions on the partnered sessions: the adversary attempts to cause partnered sessions to agree upon different keys, or force at least three sessions to agree upon the same session identifier.

Both security games have the same execution model, and share many of the same characteristics in terms of game state. Therefore we first introduce the common elements and introduce game-specific properties later. In particular, additionally to the standard queries for protocol games, both games share Send, $\mathsf{Corrupt}$ and Reveal queries. The two games have different winning conditions, and the BR-secrecy game allows the adversary the additional Test and Guess queries.

Note that instead of π, we use the notation ke for a key exchange protocol to be able to distinguish it from the protocol that it will be composed with, i.e., a key exchange protocol is denoted $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \mathsf{P}_{\mathsf{ke}})$, where $\mathsf{kg}_{\mathsf{ke}}$ generates the symmetric/asymmetric long term keys for the key exchange protocol. As in Section 3.4, we now give sematics to this algorithm, namely, P_{ke} is the "next message" algorithm for the key exchange protocol, i.e., the state sinfo of P_{ke} is initiated to (U, V, k), where k is a long term key shared between U and V. The algorithm P_{ke} can then be run on sinfo and a message msg and returns a response message msg' and an updated state sinfo'. Running P_{ke} on an empty message makes the algorithm generate the first protocol message.

The game for key exchange is written as G_{ke}. As before, this game has input tapes $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ for receiving keys and $G_{\mathsf{ke}}^{\mathsf{in}}$ for receiving queries. In addition to its normal output tape, the game has an additional output tape, $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$, where the keys derived from sessions are written. The adversary does not have access to this tape which we only use for defining the security of the composition between a key exchange and a protocol/primitive. It will play a major role for the composition theorems in Chapter 4 and Chapter 5 and it is thus introduced here. For the definition of BR-security, is is yet not needed.

The internal state of G_{ke} augments the generic list $\mathcal{L}_G$ as defined in Section 3.3. The tuples $(\mathsf{label}, \mathsf{kid}, U, V)$ in the list $\mathcal{L}_G$ are extended to tuples of the form $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$, where the semantics of the additional entries is as follows. Entry sid is a (global) session identifier set by the protocol at some point during the execution. Note that sid can have a very different structure than being, for example, the entire conversations of a session. For example it may be a partial transcript or the result of a local computation, potentially involving secret information. To analyse a protocol, one needs to choose the appropriate form of sid. The value sid must be locally computable by a session and needs to satisfy security requirements specified later. The session identifier used in the analysis of a protocol does not necessarily need to coincide with values that are called "session identifiers" in the protocol specification. For instance, TLS uses administrative session identifiers for technical reasons that do not satisfy the necessary

security requirements. In contrast to sid, the value label is not locally computed but merely an administrative game-related value which the local session of a user has no access to. The value $\mathrm{st}_{\mathrm{exec}} \in \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$ indicates the status of the session, the entry κ is the key produced by the session, and $\mathrm{st}_{\mathrm{key}} \in \{\mathsf{fresh}, \mathsf{revealed}\}$ indicates if the session key has been revealed to the adversary. If the value of κ is $\perp$ then $\mathrm{st}_{\mathrm{exec}} \in \{\mathsf{running}, \mathsf{rejected}\}$. If $\mathrm{st}_{\mathrm{exec}}$ is set to accepted for any local session label this is always the result of some query to the game.

We require the key exchange protocol to set the value κ and the value sid, before setting $\mathrm{st}_{\mathrm{exec}}$ to accepted. Furthermore, as soon as $\mathrm{st}_{\mathrm{exec}}$ is set to accepted for the session identified by label, the session key κ and the session identifier sid are written onto the tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$ and the game signals to the adversary that a session has accepted by sending the message $(\mathsf{accepted}, \mathsf{label}, U, V)$, for U and V corresponding to identifier label. This message is in addition to the normal response of the query that caused a session to accept. We can also think of the session state sinfo as beeing part of the list $\mathcal{L}_G$, as the session state is always associated with a label label. Therefore, we sometimes use the notation label.sinfo.

The adversary can interact with the game via queries for setting long-term keys (NewKey and SameKey), starting new sessions (NewSession), corrupting the long-term key of parties (Corrupt), sending messages to the different sessions (Send), and revealing the locally output keys (Reveal).

Note that the NewKey query here refers to the setting of *long-term* keys for the key exchange protocol, while the NewSession query starts key exchange protocol sessions. For instance, the (asymmetric) key set via a NewKey query correspond to TLS certificates while the NewSession query corresponds to a single TLS session. We first detail the queries appropriate to symmetric long-term keys; these are the specializations of the queries outlined in Section 3.3. Next we detail the adaptations required to model long-term asymmetric keys.

Queries for long-term symmetric keys.

- $\mathsf{NewKey}(U, V)$: The game G_{ke} checks whether there is a tuple $(*, U, V, *, *)$ or a tuple $(*, V, U, *, *)$ on list $\mathcal{L}_G^{\mathsf{keys}}$. If so, there is already a long-term key for the pair (U, V), so it returns $\perp$. Else, it reads a new key k off the $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ tape, generates a new identifier kid and creates a new tuple $(\mathsf{kid}, U, V, k, \mathsf{honest})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$. The key identifier kid is returned to the adversary.
- $\mathsf{SameKey}(U, V, \mathsf{kid})$: The game G_{ke} checks if there is a tuple $(*, U, V, *, *)$ on list $\mathcal{L}_G^{\mathsf{keys}}$. If so it returns $\perp$. Else, it searches list $\mathcal{L}_G^{\mathsf{keys}}$ for a tuple $(\mathsf{kid}, V, U, k, \mathrm{st}_{\mathsf{kid}})$ and returns $\perp$ if no such tuple exists. Else, it creates a new tuple $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$. The key identifier kid is returned to the adversary.
- $\mathsf{NewSession}(U, V, \mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for a tuple $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ and aborts if no such tuple exists. Else, it creates a new identifier label. The tuple $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ is created on list $\mathcal{L}_G$, with sid and κ being undefined, $\mathrm{st}_{\mathrm{exec}} := \mathsf{running}$, and $\mathrm{st}_{\mathrm{key}} := \mathsf{fresh}$. If $\mathrm{st}_{\mathsf{kid}} = \mathsf{corrupted}$, then $\mathrm{st}_{\mathrm{key}}$ is immediately set to revealed. The game returns label to the adversary.

- $\mathsf{Corrupt}(\mathsf{kid})$: The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for all entries of the form $(\mathsf{kid}, U, V, k, \mathrm{st}_{\mathsf{kid}})$ and does nothing if no such entry exists. Otherwise, for all such entries, it sets $\mathrm{st}_{\mathsf{kid}} = \mathsf{corrupted}$ and returns k to the adversary. For all tuples $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ on the list $\mathcal{L}_G$, $\mathrm{st}_{\mathrm{key}}$ is set to $\mathsf{revealed}$.[1] No further queries are allowed to sessions of a corrupted party.
- $\mathsf{Send}(\mathsf{label}, \mathsf{msg})$: The game returns $\perp$, if label is not in $\mathcal{L}_G$ or if $\mathsf{label}.\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$. Else, the game delivers message msg to the session labelled label and runs $\mathsf{P_{ke}}$ on the state of this session to compute a response. The response of this algorithm is returned to the adversary.
  - Upon executing $\mathsf{P_{ke}}$, if $\mathrm{st}_{\mathrm{exec}} = \mathsf{rejected}$ then the message $\mathsf{rejected}$ is also sent to the adversary.
  - Upon executing $\mathsf{P_{ke}}$, if $\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$ then the message $\mathsf{accepted}$ is also sent to the adversary, and κ and sid are written to the output tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$ of the key exchange game. Furthermore, the game searches the list $\mathcal{L}_G$ for a tuple $(\mathsf{label}', \mathsf{kid}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathsf{revealed})$. If such a tuple exists, $\mathrm{st}_{\mathrm{key}}$ is set to $\mathsf{revealed}$. This corresponds to the case where the partner session of label accepted a session and became revealed before label accepted the session key.
- $\mathsf{Reveal}(\mathsf{label})$: The game searches the list $\mathcal{L}_G$ for the tuple $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ and does nothing if no such tuple exists. Else, if a tuple is found but $\mathrm{st}_{\mathrm{exec}} \neq \mathsf{accepted}$ then the game simply returns $\perp$ to the adversary. Otherwise the game sets $\mathrm{st}_{\mathrm{key}}$ to $\mathsf{revealed}$, and returns κ to the adversary. Furthermore, if there is a tuple $(\mathsf{label}', \mathsf{kid}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}}{}')$ with $\mathrm{st}_{\mathrm{key}}{}' = \mathsf{honest}$, then $\mathrm{st}_{\mathrm{key}}{}'$ is set to $\mathsf{revealed}$. No further queries are allowed to a revealed session.

Having defined game execution for a key exchange game, we can now introduce the winning conditions for the BR-security game. For convenience, we will split the winning condition for the BR-game into two and thereby actually create two different games. The reason is that the weak requirement of Match security will also be used in Chapter 5, where key exchange protocols are not compelled to satisfy the BR key-indstinguiushability requirement. Let us turn to Match-security first.

Match-SECURITY. The condition of Match-security assures that session identifiers are a meaningful concept, namely, the game checks whether two sessions with the same sid have the same key, whether partnered sessions are partnered with the identity they expected to talk to and whether there are not more than 2 sessions with the same sid. Note that this is a very weak requirement that does not imply any security of the keys. For example, several sessions may still return the same key. The output of $G_{\mathsf{ke}}^{\mathsf{Match}}$ is formally given in Figure 3.1.

Definition 24 (Match-secure Key Exchange)**.** *We say that a key exchange protocol* $\mathsf{ke} = (\mathsf{kg_{ke}}, \mathsf{P_{ke}})$ *is* Match*-secure* ke *if it is* $(G_{\mathsf{ke}}^{\mathsf{Match}}, 0)$*-secure.*

BR-SECURITY. To provide secrecy guarantees of the session key we ask an adversary to decide whether it received the real session key, or a random value, for a session of its

[1] In the forward-secure variant, for all tuples $(\mathsf{label}, \mathsf{kid}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ with $\mathrm{st}_{\mathrm{exec}} = \mathsf{running}$ in the list $\mathcal{L}_G$, the value $\mathrm{st}_{\mathrm{key}}$ is set to $\mathsf{revealed}$.

For each $\mathsf{label} \in \mathcal{L}_G$ do
  If $\mathsf{label.sid} \neq \perp$ then for all
  $\mathsf{label}' \in \mathcal{L}_G$ with $\mathsf{label}'.\mathsf{sid} = \mathsf{label.sid}$ do
    If $\mathsf{label}.U \neq \mathsf{label}'.V$ or $\mathsf{label}.V \neq \mathsf{label}.U$ then return 1.
     `//distinct intended partners`
    If $\mathsf{label}.U = \mathsf{label}'.V$ and $\mathsf{label}.V = \mathsf{label}.U$, $\mathsf{label}.\mathrm{st}_{\mathrm{exec}} = \mathsf{label}.\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$ but
    $\mathsf{label}.\kappa \neq \mathsf{label}'.\kappa$ then return 1. `//distinct keys`
  If the number of labels $\mathsf{label}' \in \mathcal{L}_G$ with
  $\mathsf{label}'.\mathsf{sid} = \mathsf{label.sid}$ is strictly larger than 2,
  then return 1. `//too many partners`
Return 0.

Figure 3.1: Output of the Match-game G_{Match}.

Test(label):
 If $b_{\mathsf{test}} = 0$ then $\kappa \xleftarrow{\$} \mathcal{D}$
 Else set $\kappa := \mathsf{label}.\kappa$,
 Return κ
Guess(b):
 $b_{\mathsf{guess}} := b$
 Return okay

Figure 3.2: Test and Guess queries for the BR-secrecy game.

choice. It is assumed any random value is drawn from some key distribution $\mathcal{D}$ (often the uniform distribution for bit strings of length $|\kappa|$). We write $\kappa \xleftarrow{\$} \mathcal{D}$ for the value of κ drawn randomly from the distribution $\mathcal{D}$. We call this game the BR-secrecy game and use the game execution model of key exchange protocols as described so far. We now set out the additional details of the model for the secrecy property.

The state of the BR-secrecy game contains a bit $b_{\mathsf{test}} \in \{0, 1\}$, drawn at random in the beginning of the game and a bit $b_{\mathsf{guess}} \in \{0, 1, \perp\}$ along with a session identifier $\mathsf{label}_{\mathsf{tested}} \in \mathsf{LABELS} \cup \{\perp\}$. The bit b_{test} determines whether the adversary is given the real session key, or random value in response to the Test query.[2] The bit b_{guess} stores the adversary's guess for the value of b_{test}. The session identifier $\mathsf{label}_{\mathsf{tested}}$ is the label of the session for which the adversary made the Test query.

There are two additional queries required to model the BR-security of a key exchange protocol, namely Test and Guess. The Test query provides the adversary with either the real session key for a given session, or a random value. The Guess query provides the game with the adversary's guess to the value b_{test}. The queries are given in Figure 3.2.

In order to prevent trivial attacks, we place restrictions on the admissibility of the

[2] We assume that the adversary only makes a single Test query. Security with respect to many Test queries then follows by a hybrid argument [BR93].

If $\mathsf{label}_{\mathsf{tested}} = \perp$ then return 0. `//No Test query made`
For each $\mathsf{label} \in \mathcal{L}_G$ s.t. $\mathsf{label.sid} = \mathsf{label}_{\mathsf{tested}}.\mathsf{sid}$ do
`//Test for exposure of partner key or key itself`
If $\mathsf{label}.\mathrm{st}_{\mathrm{key}} = \mathsf{revealed}$ then return 0.
If $b_{\mathsf{test}} \neq b_{\mathsf{guess}}$ then return 0. `//Wrong guess`
Else return 1.

Figure 3.3: Predicate for the BR-secrecy game.

Test query. An adversary is not allowed to test a session which is corrupt, has not accepted, or whose partner is corrupt, or to test more than a session. In these cases, the game returns the error message false. Note that these cases are publicly verifiable.

Moreover, the adversary should not Test a session which is revealed or where the partner session has been revealed. In these cases though, the game does not return false but instead lets the adversary continue. This is in order to prevent leakage of partnering information through the Test query. The adversary may not be aware this has occurred; however at the end of execution the game checks for this, and causes the experiment to be lost if such an action has occurred. We also forbid Reveal queries on the tested session or its partner. Again, the adversary is later declared to lose if such a Reveal query happens (but again without being informed immediately). Furthermore we only allow the adversary one Guess query.

To determine whether the adversary has won, the BR-secrecy game checks if the adversary's guess for the value of b_{test} is correct. Furthermore, the predicate causes the adversary to lose the game if the tested session (or its partner) have been revealed. Note that no checks relating to corruption are required, as the game automatically marks all session-keys as revealed when the long-term key of a party is corrupted. Interestingly, with our notation, the predicate for BR-security is identical for forward-secure and non-forward-secure protocols. The reason here is that the corruption model is entirely encoded into the corrupt query. The game's $G_{\mathsf{ke}}^{\mathsf{BR}}$ response is formally depicted in Figure 3.3.

We write the BR-secrecy game as $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$, where $\mathcal{D}$ is the key distribution from which random keys are chosen during the Test query. Furthermore we denote the game $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ with the secret bit b_{test} as $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D},b_{\mathsf{test}}}$.

Definition 25 (BR-secure Key Exchange)**.** *We say that a key exchange protocol* $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \mathsf{P}_{\mathsf{ke}})$ *is BR-secure, if it is* $(G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}, \frac{1}{2})$*-secure and* $(G_{\mathsf{ke}}^{\mathsf{Match}}, 0)$*-secure.*

3.5.1 Modifications for Asymmetric Long-Term Keys

To modify the symmetric key model into an asymmetric key model, the main task is to deal with the key identifiers and the NewKey query. Instead of having two inputs, the NewKey query has only one, as keys are now assigned to single users instead of pairs of users. For the same reason, the SameKey query becomes obsolete. The list of

sessions, $\mathcal{L}_G$, stores tuples $(\mathsf{label}, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$. The owner of the session is U. So, U uses its secret key corresponding to kid_U, and U uses the public key corresponding to kid_V. The Send and Reveal queries are the same as for symmetric long-term keys. We now formally define the NewKey, NewSession and Corrupt queries for asymmetric long-term keys.

- NewKey(U): The game G_π reads a new key pair (sk, pk) off the $G_\pi^{\mathsf{k\text{-}in}}$ tape, generates a new identifier kid and creates a new tuple $(\mathsf{kid}, U, (sk, pk), \mathsf{honest})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$. The key identifier kid is returned to the adversary together with pk.
- NewSession($U, V, \mathsf{kid}_U, \mathsf{kid}_V$): The game searches the tuples $(\mathsf{kid}_U, U, (sk, pk), \mathrm{st}_{\mathsf{kid}})$ and $(\mathsf{kid}_V, V, (sk', pk'), \mathrm{st}_{\mathsf{kid}})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$ and aborts if either of the tuples does not exist. Else, it generates a new identifier label and creates the tuple $(\mathsf{label}, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ on the list $\mathcal{L}_G$ and returns label to the adversary. Note that also, instead of initializing sinfo to (U, V, k), it is initialized to (U, V, pk_U, sk_U, pk_V).
- Corrupt(kid): The game searches the list $\mathcal{L}_G^{\mathsf{keys}}$ for an entry $(\mathsf{kid}, U, (sk, pk), \mathrm{st}_{\mathsf{kid}})$ and does nothing if no such entry exists. Otherwise it sets $\mathrm{st}_{\mathsf{kid}} = \mathsf{corrupted}$ and returns (sk, pk) to the adversary. No further queries are allowed to sessions of U that use the secret key corresponding to kid. Note that queries to sessions of V that use the public key corresponding to kid are still allowed. In the forward-secure case, for all sessions $(\mathsf{label}', \mathsf{kid}_V, \mathsf{kid}_U, V, U, \mathsf{sid}, \mathsf{running}, \kappa, \mathrm{st}_{\mathrm{key}})$, $\mathsf{label}'.\mathrm{st}_{\mathrm{key}}$ is set to revealed. In the non-forward-secure case, for all sessions $(\mathsf{label}', \mathsf{kid}_V, \mathsf{kid}_U, V, U, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$, $\mathsf{label}'.\mathrm{st}_{\mathrm{key}}$ is set to revealed.

3.6 Composition of Key Exchange with Protocols

Keys derived via key exchange protocols can be used in symmetric protocols and primitives, and we aim to determine when such uses are secure. In this section we define what "secure use" means by giving security games for the composition between key exchange and primitives and protocols.

The composed game runs the key exchange game and the primitive/protocol game as subgames. Whenever a session in the key exchange phase accepts a key, then the composed game passes this key to the protocol/primitive game as a new key. Thus, the adversary is not given access to the NewKey query, as new keys are passed directly from the key exchange protocol to the primitive/protocol. Otherwise, the adversary is given all key exchange queries (to model attacks in the key exchange phase) and all queries of the primitive/protocol game (to model attacks on the latter). The adversary is successful when satisfying the winning condition of the primitive/protocol game. The key exchange game does not have a separated winning condition. The key exchange protocol is considered suitable for the primitive/protocol if the adversary cannot break the primitive/protocol when the previously randomly chosen keys are replaced by keys derived via a key exchange protocol.

We now discuss the formalism in more detail: We have already formally defined the execution for key exchange protocols via the game G_{ke}. The game G_{ke} writes the keys output by the sessions of the protocol on its special output tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$. We have

also defined (generic) security notions for protocols and primitives G_π and G_ζ. Both these games expect to receive keys as input on the special input tapes $G_\pi^{\mathsf{k\text{-}in}}$ and $G_\zeta^{\mathsf{k\text{-}in}}$, respectively. We now define the game $G_{\mathsf{ke};\zeta}$, which allows an adversary to simultaneously interact with the key exchange protocol and with the instantiation of the primitive that uses the keys derived via the key exchange protocol. Roughly speaking, we "fuse" the game G_{ke} with G_ζ by simply passing the keys written on $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$ to $G_\zeta^{\mathsf{k\text{-}in}}$. The output tape of the resulting game is the output tape of G_ζ. Since the subgame G_ζ writes the bit onto the tape, this means that the goal of the adversary is to break ζ. The game $G_{\mathsf{ke};\pi}$ reflects the analogous idea for protocols. In Section 3.6.1 we present these ideas in greater detail, and show how the games internally maintain state, and pass information from the key exchange sub-game to the protocol/primitive sub-game.

An interesting issue arises when considering corruption. In the composed game, corruptions need to be treated consistently. For instance, the adversary might reveal keys in the key exchange phase while not corrupting the key in the primitive/protocol game. Then, the adversary could trivially win any game. Thus, whenever a key is revealed in the key exchange phase, the composed game issues a Corrupt query to the primtive/protocol subgame. For the long-term keys of the key exchange, we need to distinguish *forward security* and *non-forward security*. When a protocol is forward secure, then corruption of the long-term key used in the key exchange does not affect sessions which have already terminated. However, in non-forward secure protocols, corruption of the long-term secrets automatically renders insecure, all session keys which were established using this key. Hence, in the non-forward secure case, the composed game marks all these keys as corrupted in the primitive/protocol game via additional Corrupt queries. For forward secure protocols, no additional action needs to be undertaken. We thus distinguish between the forward secure composed game $(G_{\mathsf{ke};\rho}^{\mathsf{fs}}, \delta_\rho)$ and the non-forward secure composed game $(G_{\mathsf{ke};\rho}^{\mathsf{nfs}}, \delta_\rho)$. Again this is detailed more formally in the following Subsection.

In Chapter 4, we will see that forward-secure key exchange protocols can even be composed with protocols that do not support key corruption. We will then conclude that it is meaningful for those to omit the Reveal query from the composed game, because the Reveal query in the BR-game models leakage of information through the use of keys in a primitive/protocol, which is, indeed, already included in the composed game. And the security game for the primitive/protocol in the composed game considers entire key loss an unlikely event. For non-forward secure protocols or for protocols that support corruption, such considerations are not helpful, as in both cases, leakage of the entire key is considered to be a possible event in a real-life execution. Thus, for the remainder of this chapter, we assume that the primitive/protocol support key corruption and include the Reveal query.

3.6.1 Details of the Key-Exchange/Protocol Composition

We first detail how to compose a key exchange protocol with a protocol, then we discuss the corruption model, and finally we discuss the modifications to compose a key exchange

protocol with a primitive.

COMPOSITION OF KEY EXCHANGE WITH PROTOCOLS. The game $G_{\mathsf{ke};\pi}$ internally runs a copy of G_{ke} and a copy of G_π. The key input tape is $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ and the tape for the output bit is G_π^{out}. The tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$ and the input tape $G_\pi^{\mathsf{k\text{-}in}}$ are internalized by the composed game; we explain later how $G_{\mathsf{ke};\pi}$ uses these to pass keys from one game to the other. The query input tapes $G_{\mathsf{ke}}^{\mathsf{in}}$ and G_π^{in} of the two subgames are internalized as well. Instead the game has a new input tape, on which it accepts any of the following queries: $\mathsf{NewKey}_{\mathsf{ke}}$, $\mathsf{SameKey}_{\mathsf{ke}}$, $\mathsf{NewSession}_{\mathsf{ke}}$, $\mathsf{Corrupt}_{\mathsf{ke}}$, $\mathsf{Send}_{\mathsf{ke}}$ and $\mathsf{Reveal}_{\mathsf{ke}}$ which are intended for the subgame G_{ke} and also queries $\mathsf{NewSession}_\pi$, $\mathsf{Corrupt}_\pi$ and Name_π for the subgame G_π. Here Name_π is a generic query for the protocol game. The parameters of these queries are as before. Notice that the adversary is no longer allowed the queries NewKey_π, $\mathsf{SameKey}_\pi$, as keys for the protocol sessions are now obtained from the G_{ke} game. The composed game internally maintains a list $\mathcal{L}_{\text{Identifiers}}$ linking sessions of the key exchange game to key identifiers of the protocol game. The list $\mathcal{L}_{\text{Identifiers}}$ is a list of tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ of administrative session identifiers $\mathsf{label}_{\mathsf{ke}}$, session identifiers, sid, of the key exchange game and key identifiers, kid_π, for the underlying protocol game.

For most queries, the composed game simply forwards the queries of the adversary to the appropriate subgame, and forwards back the response. For example when the adversary makes a $\mathsf{NewKey}_{\mathsf{ke}}(U, V)$, the composed game makes a $\mathsf{NewKey}(U, V)$ query to G_{ke} and returns $\mathsf{kid}_{\mathsf{ke}}$ obtained from G_{ke} to the adversary. The trickier parts of the execution deal with passing the keys from one game to the other and with (long-term and session) key corruption. We explain these difficulties in turn.

Keys are passed from G_{ke} to G_π when some session in G_{ke} accepts, *i.e.* , when G_{ke} writes (κ, sid) on $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$. There are two possible situations: if the pair of session identifier and session key, (sid, κ) had not been output before, then κ is a new key established between the identities associated to sid. Thus, the game generates a new key identifier which is returned to the adversary. Otherwise, there already exists a session of the key exchange with the same values of sid and κ. This session is the partner of the newly finished key exchange session. Therefore, we initialise the newly finished session within the protocol subgame via a $\mathsf{SameKey}$ query, thus partnering this session with the previously established protocol session. We now formalize these two situations.

THE $\mathsf{Send}_{\mathsf{ke}}$ QUERY. When, a query $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ is made, the $G_{\mathsf{ke};\pi}$ performs the following operations:

- Check whether $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\text{exec}} = \mathsf{running}$.
- If not, return failure message false.
- Else, run $G_{\mathsf{ke}}(\mathsf{Send}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg}))$.
- If it returns msg', return msg'.
- If it returns $(\mathsf{msg}', \mathsf{accepted})$ or $\mathsf{accepted}$, then the key exchange game wrote $(\mathsf{sid}^*, \kappa^*)$ to its output tape.
- If there is $\mathsf{label}'_{\mathsf{ke}}$ with $\mathsf{label}'_{\mathsf{ke}}.\mathsf{sid} = \mathsf{sid}^*$ and $\mathsf{label}'_{\mathsf{ke}}.\mathrm{st}_{\text{exec}} = \mathsf{accepted}$, then issue the query $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ to the protocol game G_π.
- Else, write key κ on the input tape of the protocol game G_π and query $\mathsf{NewKey}_\pi(U, V)$ to the protocol game G_π which returns a key identifier kid_π that is returned to the

adversary, and the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ is stored in list $\mathcal{L}_{\text{Identifiers}}$. Further, if $\text{st}_{\text{key}} = \mathsf{revealed}$ the game sends the query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to G_π.

- Return all answers from the subgames to the adversary.

Corruption Model. We now explain how the composed game deals with corruption. The problem is that corrupting keys in one of the games influences which keys are corrupt in the other game. We start with the simpler case of $\mathsf{Reveal}_{\mathsf{ke}}$ queries. When such a query is issued for some session label, the composed game sends $\mathsf{Reveal}(\mathsf{label})$ to G_{ke}. If the answer is $\perp$ then nothing else happens. Otherwise (*i.e.* the answer is some session key k) then for each entry $(\mathsf{label}, \mathsf{sid}, \mathsf{kid}_\pi)$ on the list $\mathcal{L}_{\text{Identifiers}}$, the composed game issues a $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ query to G_π. These queries essentially mark that the key has been corrupted. The game then returns k to the adversary.

For corruption of symmetric long-term keys we distinguish two possibilities. In the forward-secure version of the game, corrupting a long-term key does not affect the security of sessions keys already established (using the long-term key). In the non forward-secure version, all of the sessions keys derived using the long-term key become corrupt. To distinguish between the two possibilities we often denote the corresponding games $G^{\mathsf{fs}}_{\mathsf{ke};\pi}$ (**F**orward **S**ecure) and $G^{\mathsf{nfs}}_{\mathsf{ke};\pi}$ (**N**on-**F**orward **S**ecure), respectively. We will omit these subperscripts when they are clear from context. For example, throughout Chapter 4, we will only treat forward-secure key exchange protocols.

In $G^{\mathsf{fs}}_{\mathsf{ke};\pi}$ the $\mathsf{Corrupt}_{\mathsf{ke}}$ queries are just forwarded to the subgame $G^{\mathsf{fs}}_{\mathsf{ke}}$ and its answer it relayed to the adversary. In the $G^{\mathsf{nfs}}_{\mathsf{ke};\pi}$ version, when a corruption query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid}_{\mathsf{ke}})$ is received, the composed game relays it to the subgame $G^{\mathsf{nfs}}_{\mathsf{ke}}$ which returns a key k, that is passed back to the adversary. Furthermore, for all sessions $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, *, *, \mathsf{sid}, \mathsf{accepted}, \kappa, \text{st}_{\text{key}})$ in $\mathcal{L}_{G_{\mathsf{ke}}}$, the composed game searches the list $\mathcal{L}_{\text{Identifiers}}$ for tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and sends the query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to the protocol game G_π. The answer of the subgame is also relayed to the adversary. This models the idea that if the key exchange is not forward secure, corruption of long term keys also compromises the derived session keys.

For asymmetric long-term keys, the forward-secure model is as described for symmetric long-term keys. In the non-forward-secure model (for asymmetric long-term keys), whenever the composed game receives the query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid}_U)$, for all tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_U, *, *, *, \mathsf{sid}, \mathsf{accepted}, \kappa, \text{st}_{\text{key}})$ and $(\mathsf{label}'_{\mathsf{ke}}, *, \mathsf{kid}_U, *, *, \mathsf{sid}', \mathsf{accepted}, \kappa', \text{st}_{\text{key}}{}')$ in $\mathcal{L}_{G_{\mathsf{ke}}}$, the game searches $\mathcal{L}_{\text{Identifiers}}$ for triples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{sid}', \mathsf{kid}'_\pi)$. It then queries $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ and $\mathsf{Corrupt}_\pi(\mathsf{kid}'_\pi)$ to the G_π subgame for all found values. The answers of the subgame are relayed to the adversary.

3.6.2 Composing Key Exchange with Primitives

Most of the discussion above also applies to the composition of key exchange protocols with primitives. We therefore give only relevant details of defining $G_{\mathsf{ke};\zeta}$; highlighting the differences. As above, we distinguish between the forward secure, $G^{\mathsf{fs}}_{\mathsf{ke};\zeta}$, and non forward secure $G^{\mathsf{nfs}}_{\mathsf{ke};\zeta}$ versions of the composed game. The input/output tape configuration is as above. In addition to the queries for the key exchange subgame, which are as

above, an adversary is allowed to make queries: $\mathsf{Corrupt}_\zeta$ and Name_ζ for the subgame G_ζ (where Name_ζ is any generic query). The query NewKey_ζ used to instantiate keys for the G_ζ subgame is only used internally by the composed game. The only conceptual difference between composition of key exchange with protocols and with primitives is that for primitives, the key agreed by two parties which have obtained the same sid is instantiated in the subgame for the primitive only once. Specifically, when some session of the key exchange outputs (sid, κ) on $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$ the composed game searches for a triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$ in the $\mathcal{L}_{\text{Identifiers}}$. If such a triple exists, then the game takes no further action. Otherwise, the composed game writes κ on $G_\zeta^{\mathsf{k\text{-}in}}$ and issues a NewKey_ζ query to its G_ζ subgame. As for protocols, if the session where the key has been obtained is revealed, then the composed game issues a $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\zeta)$ query to G_ζ.

3.6.3 Suitability for Primitives/Protocols

Above we defined how key exchange protocols can be composed with primitives and protocols. We now explain what it means for a key exchange protocol to be suitable for primitive ζ (and analogously for protocol π). Intuitively, this means that the security of the primitive does not break down when, instead of using keys generated with the key generation algorithm for the primitive, one uses the keys established by the key exchange protocol. Using the machinery developed in the previous section, the requirement simply means that the game $G_{\mathsf{ke};\zeta}$ for the composed protocol $(\mathsf{ke}; \zeta)$ cannot be won when the long-term keys of the parties are generated honestly. This intuition, which applies equally to the case of composing key exchange with protocols, is formalized next. The definition treats explicitly both the forward-secure and the non-forward secure settings.

Definition 26 (Suitability for Primitives/Protocols)**.** *Let* $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \mathsf{P}_{\mathsf{ke}})$ *be a key exchange protocol,* $\rho = (\mathsf{kg}_\rho, \mathsf{P}_\rho)$ *be an arbitrary primitive or protocol, and* (G_ρ, δ_ρ) *an arbitrary security notion for* ρ*. We say that* ke *is* (G_ρ, δ_ρ)*-suitable-for-*ρ *if* $(\mathsf{kg}_{\mathsf{ke}}, \mathsf{ke}; \rho)$ *is* $(G_{\mathsf{ke};\rho}^{\mathsf{nfs}}, \delta_\rho)$*-secure. We say that that* ke *is* (G_ρ, δ_ρ)*-suitable-for-*ρ *with forward security if* $(\mathsf{kg}_{\mathsf{ke}}, \mathsf{ke}; \rho)$ *is* $(G_{\mathsf{ke};\rho}^{\mathsf{fs}}, \delta_\rho)$*-secure.*

If the security notion for ρ is clear from the context, we may simple say that ke is suitable-for-ρ. One aspect we wish to stress is that (as per Definition 18 of a key-benign adversary) the key generation used to initialize the composed game is the key generation algorithm of the *key exchange protocol*. In turn this means that the main functionality of the adversary is covered by the second stage of the adversary that interacts simultaneously with the underlying games G_{ke} and G_ρ.

The main property desired from a key exchange protocol is to be suitable for the protocol where the keys derived are then used. In the next section we show that being suitable for the symmetric primitive on which the protocol relies together with the authentication property we define next, suffice to ensure this. The intuition of why we need an authentication property is the following. In the composition of key exchange with primitives, for every two partnered sessions (*i.e.* that have the same sid), the adversary is given access to a single instance of the primitive under the key derived in the session that

finishes first. When the partner session finishes, the key is ignored. We therefore ensure via the Match-property that partnered sessions do agree on the same key and that there exist at most two sessions which agree on the same sid. Notice that the requirement is very weak. In particular, it even allows for the same key to be output in multiple sessions. However, the notion of suitability for a specific primitive will usually disallow unrelated sessions to output the same key, as naturally, this leads to a security breach for most natural primitives.

Chapter 4

Composability of BR Key Exchange Protocols

In this chapter, we prove that forward-secure BR-secure key exchange protocols, for which a session matching algorithm exists, can be securely composed with arbitrary symmetric key protocols. In practice, assume you want to run a key exchange protocol to use the keys for a secret channel. To conclude security of the entire protocol, one would usually analyse the protocol as a whole. Instead, with our theorem, one can now analyse the two components seperately, and, more importantly, if one uses an existing provably secure key exchange protocol (i.e. BR-secure), one can simply re-use the existing security analysis without further investigation; and the same applies to secret channels.

We notice that many games such as the multi-user encryption game that we considered in Section 3.4 fall in the class of so-called *single session reducible* protocols, a notion we introduce in this chapter. For protocols in this class, it suffices to analyse a single session of the protocol and security for concurrent execution follows automatically.

Overall, in the case of a composed protocol consisting of a key exchange part and a secret channel part, the analysis boils down to a single session analysis of the secret channel protocol and a (possibly existing) BR-analysis of the key exchange part. For clarity, we emphasise that single session reducibility is not a requirement of our framework, but a useful tool to shorten a complex analysis if applicable.

We now take a closer look at the public session matching we assume to exist: one might think that this requirement is a necessary artifact for our proof to work. However, this is (provably) not the case. In Section 4.5, we show that if a key exchange protocol is composable with arbitrary symmetric key protocols and security is shown via a specific kind of black-box reduction, then (a weak form of) a session matching algorithm exists. We emphasize that the public session matching is only on the key exchange protocol, and *not* on the subsequent uses of the key; hence, it does not impact the protocol with which the key exchange is composed.

We finally note that it may look impossible to provide secure composition of key exchange protocols with *arbitrary* symmetric key protocols. The seemingly intuitive counter argument is that, if the symmetric key protocol "misbehaves" in the sense that

it duplicates some steps of the key exchange protocol in a bad way, then the composition would easily become insecure. As an example assume that the key exchange somehow involves (in a secure way) a step in which a nonce is encrypted under the new session key, and that the first step of the subsequent protocol is that a party, exceptionally receiving such an encrypted message, would immediately disclose the session key. Then replaying the previous message from the key exchange phase should violate the security of the overall protocol. This line of reasoning, however, is incorrect. Key indistinguishability of a key exchange protocol essentially says that one can replace the actual key by an independent random key, more or less decoupling the two phases. This is even true in presence of key leakage in the symmetric key protocol, as such leakage can be already captured in the Bellare-Rogaway model through special key reveals the adversary can enforce. This implies that the "misbehaving" symmetric key protocol either contradicts the indistinguishability of the key exchange protocol, or that the duplication of steps is harmless because the derived key is independent of the information flow in the key exchange phase. Carrying out this argument formally requires some care, especially with the session matching, but our theorem shows that general composition indeed holds.

This chapter deals with BR-key exchange protocols based on asymmetric long-term keys. The case of symmetric long-term keys follows analogeously—see also Chapter 5, where we prove a composition theorem in the setting of symmetric long-term keys. Forward-secure BR-key exchange protocols support any type of symmetric key application protocol, and noteworthy, even those that do not support key corruption. Clearly, the same theorem cannot hold for non-forward secure protocols—here, a loss of the long-term keys makes all session keys associated to the long-term key insecure. As a consequence, the symmetric key protocol has to be secure even in a setting where adaptive corruption is possible. In Section 4.6, we prove that also non-forward secure key exchange protocols can be securely composed with a large class of symmetric-key protocols that we specify, namely the class of all so-called *session-restricted* protocol, as notion that is slightly stronger than single-session reducibility. We will later explain how our second composition Theorem 3 in Chapter 5 can be combined to yield even more general results. We will now first turn to forward-secure protocols.

4.1 On Forward Secure Protocols

Forward security is a desirable security feature that many key exchange protocols enjoy. In a nutshell, a key exchange protocol is forward-secure, if corruption of a user does not affect those session keys where the session has already terminated. Moreover, the same applies when the partner of a terminated session gets corrupted. A prominent example is a signed Diffie-Hellman key exchange—upon corruption of the longterm signature keys, one does not learn anything about the session keys, as the session keys do not contain any information about the secret exponent. In contrast, if the adversary obtains the signing key while a session has not finished yet, he is able to insert his own value g^b and thus entirely learn the key of the receiver session.

A typical example for a non-forward secure protocol is key transport. Here, one

user generates a symmetric key and encrypts it under the other receiver's public key. Upon corruption of the receiver, the adversary learns the secret encryption key and thus learns all previously exchanged symmetric keys of the user. In this chapter, we prove a composition theorem for forward-secure BR-secure protocols and arbitrary application protocols. In Chapter 4.6, we will also treat the more involved case of non-forward secure key exchange protocols, for which we can prove composability with a large class of symmetric-key protocols.

Dealing with forward-secure protocols has the benefit that keys that were established between two users cannot be rendered insecure via a Corrupt query. Looking at the key exchange game G_{ke}, we observe that the Reveal queries equally allow the adversary to learn sessions keys. Yet, we work in the setting of composition. Reveal queries in the BR-secrecy game model possible key leakage through use of the keys in and application protocol. But as we explicitly model a composed game, the use of the keys is already modelled by the application protocol game G_π. The game for the application protocol might consider it unlikely that keys entirely leak, e.g., because they are only used once and are securely erased immediately afterwards. It is noteworthy that in the composed game, a Reveal query to the key exchange game corresponds to a key Corrupt query in the protocol state and vice versa. Thus, we can safely drop the Reveal queries from the key exchange part of composed game and leave it up to the protocol part to decide on admitting corruption or not. Our composition theorem then holds for both, application protocols that admit Corrupt queries and those which do not.

4.1.1 Modifications of Protocol Games

As explained, we drop the $\mathsf{Corrupt}_\pi$ queries as explicit requirement from the model for protocol games and do not make use of it any longer in the description of the composed game. We stress again that the protocol may allow this query in the set of "additional queries". Note that, in contrast to adaptive key corruption, the protocol game still has to support static corruption, respectively adversarially set keys. The reason is that, when composed with a key exchange, an honest session of the key exchange might accept while the partner is already corrupt. We have seen that, in the case of signed Diffie-Hellman, the adversary is able to influence and learn the key in this case. We thus require the protocol game to admit a $\mathsf{SetKey}(U, V, \kappa)$ query (instead of the Corrupt-query). Formally, $\mathsf{SetKey}(U, V, \kappa)$ in Game G_π works as follows:

- The game G_π reads a new key k off the $G_\pi^{\mathsf{k\text{-}in}}$ tape, generates a new identifier kid and creates a new tuple $(\mathsf{kid}, U, V, k, \mathsf{corrupted})$ on the list $\mathcal{L}_G^{\mathsf{keys}}$.
- The key identifier kid is returned to the adversary.

Except for this modification, all definitions of the protocol game remain identical to those given in Section 3.3.

4.1.2 Modifications of the Composition Model

We now have to adapt the definition of the composed game, as defined in Section 3.6. As explained, as it is up to the application protocol to admit Reveal-queries, we deny the

$\mathsf{Reveal}_{\mathsf{ke}}$ query to the adversary. THE $\mathsf{Send}_{\mathsf{ke}}$ QUERY. In the forward-secure setting, only the $\mathsf{Reveal}_{\mathsf{ke}}$ and the $\mathsf{Send}_{\mathsf{ke}}$ query might affect the protocol stage of the composed game. As there are no $\mathsf{Reveal}_{\mathsf{ke}}$ queries, it suffices to define the modified $\mathsf{Send}_{\mathsf{ke}}$ query in the game $G^{\mathsf{fs}}_{\mathsf{ke};\pi}$—we denote $G^{\mathsf{fs}}_{\mathsf{ke};\pi}$ by $G_{\mathsf{ke};\pi}$ throughout this chapter. On input $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$, the game $G_{\mathsf{ke};\pi}$ performs the following operations:

- Check whether $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{running}$.
- If not, return failure message false.
- Else, run $G_{\mathsf{ke}}(\mathsf{Send}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg}))$.
- If it returns msg', return msg'.
- If it returns $(\mathsf{msg}', \mathsf{accepted})$ or $\mathsf{accepted}$, then the key exchange game wrote $(\mathsf{sid}^*, \kappa^*)$ to its output tape.
- If there is $\mathsf{label}'_{\mathsf{ke}}$ with $\mathsf{label}'_{\mathsf{ke}}.\mathsf{sid} = \mathsf{sid}^*$ and $\mathsf{label}'_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$, then issue the query $\mathsf{SameKey}_{\pi}(V, U, \mathsf{kid}_{\pi})$ to the protocol game G_{π}.
- Else if $\mathsf{label}.\mathrm{st}_{\mathrm{key}} \neq \mathsf{revealed}$, then write key κ on the input tape of the protocol game G_{π} and query $\mathsf{NewKey}_{\pi}(U, V)$ to the protocol game G_{π} which returns a key identifier kid_{π} that is returned to the adversary, and the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_{\pi})$ is stored in list $\mathcal{L}_{\mathrm{Identifiers}}$.
- Else if $\mathsf{label}.\mathrm{st}_{\mathrm{key}} = \mathsf{revealed}$, then query $\mathsf{SetKey}(\mathsf{label}.U, \mathsf{label}.V, \kappa^*)$ to G_{π}.
- Return all answers from the subgames to the adversary.

The main difference between the $\mathsf{Send}_{\mathsf{ke}}$-query here and the one that we defined in Section 3.6 is that in the case where sid is new and $\mathsf{label}.\mathrm{st}_{\mathrm{key}} = \mathsf{revealed}$, we now use the SetKey_{π} query instead of a NewKey_{π} and a $\mathsf{Corrupt}_{\pi}$-query.

Note that in the case where there is a $\mathsf{label}'_{\mathsf{ke}} \in \mathcal{L}_{G_{\mathsf{ke}}}$ with $\mathsf{label}'_{\mathsf{ke}}.\mathsf{sid} = \mathsf{sid}^*$, we do not have to check whether $\mathsf{label}'.U$ is corrupted. If $\mathsf{label}'.U$ is corrupted and sid is identical, then the sessions derived the same session key (due to the Match-property). Moreover, the session key $\mathsf{label}'.U$ is secure due to forward security. Let us shortly conclude what this means for the session identifier sid in the signed Diffie-Hellman case: the session identifier cannot be a pairs of nonces that are exchanged in the beginning (as this would violate Match security). Quite on the contrary, the entire value g^b has to be taken into account by the session identifier. However, the signature might be modified by the adversary without harm and thus does not need to be a part of the session identifier.

4.2 Session Matching

For composability, we need an additional property, called *session matching*. Roughly, this means that an eavesdropper on the communication between the BR-adversary and the BR-secrecy game should be able to deduce which sessions are partnered, i.e., at any time, the eavesdropper should be able to produce a list of pairs of all partnered (accepted) sessions. Note that this is trivially satisfied when defining session identifiers through matching conversations. However, when using abstract session identifiers, sid, this need not be the case. For instance, consider a BR-secure key exchange that uses matching conversations as the session identifier. We now transform the protocol as follows: the participants encrypt all messages they send. The session identifiers are

$\mathsf{P}_{\mathsf{partner}}(\mathcal{L}_{G_{\mathsf{ke}}}, \mathsf{LABELS}_{\mathsf{partner}}, \mathsf{LABELS}_{\mathsf{single}})$:
For each $\mathsf{label} \in \mathsf{LABELS}_{\mathsf{single}}$ do
`//Alleged single parties don't have partners`
If $\mathsf{label}.\mathrm{st}_{\mathrm{exec}} \neq \mathsf{accepted}$ then return 0.
Else if there exists $\mathsf{label}' \in \mathcal{L}_{G_{\mathsf{ke}}}$, $\mathsf{label}' \neq \mathsf{label}$ with $\mathsf{label}'.\mathsf{sid} = \mathsf{label}.\mathsf{sid}$ then return 0.
For each $(\mathsf{label}, \mathsf{label}') \in \mathsf{LABELS}_{\mathsf{partner}}$ do
`//Alleged partners have accepted and are really partnered`
If $\mathsf{label}.\mathrm{st}_{\mathrm{exec}}, \mathsf{label}'.\mathrm{st}_{\mathrm{exec}}) \neq (\mathsf{accepted}, \mathsf{accepted})$ then return 0.
If $\mathsf{label}.\mathsf{sid} \neq \mathsf{label}'.\mathsf{sid}'$ then return 0.
For each $\mathsf{label} \in \mathcal{L}_{G_{\mathsf{ke}}}$ do
`//Each accepted session is assigned as single or partnered`
If $\mathsf{label}.\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$, $\mathsf{label} \notin \mathsf{LABELS}_{\mathsf{single}}$ and for all
$(\mathsf{label}_0, \mathsf{label}_1) \in \mathsf{LABELS}_{\mathsf{partner}}$, one has $\mathsf{label}_0 \neq \mathsf{label}$ and $\mathsf{label}_1 \neq \mathsf{label}$ then return 0.
Return 1.

Figure 4.1: Session matching predicate.

now defined as matching conversations *on the plaintexts*. First note that the resulting key exchange protocol is as secure as the original, assuming secure encryption. But the protocol has an interesting property: assume the encryption scheme is re-randomizable. Then, an eavesdropper on the communication is unable to deduce which sessions between two parties are partnered, as the BR-adversary may re-randomize all messages sent.

We therefore define an efficient session matching algorithm $\mathcal{M}$ which can deduce from the communication between the BR-secrecy adversary and BR-secrecy game which sessions are partnered. Algorithm $\mathcal{M}$ is allowed to see all queries and answers exchanged between a key exchange (game) and an adversary $\mathcal{A}$; this includes all public parameters of the system. The requirement on $\mathcal{M}$ is independent of the winning condition of $\mathcal{A}$ in the game; algorithm $\mathcal{M}$ needs to always provide correct matchings.

More formally, a *session matching algorithm* $\mathcal{M}$ for the key exchange protocol is defined as an efficient algorithm that receives all information exchanged between a key exchange game G_{BR} and an adversary $\mathcal{A}$ against G_{BR}.

We require that each time the key exchange game sends a response to the adversary $\mathcal{A}$, algorithm $\mathcal{M}$ is able to output two sets $\mathsf{LABELS}_{\mathsf{partner}}$ and $\mathsf{LABELS}_{\mathsf{single}}$, where $\mathsf{LABELS}_{\mathsf{partner}}$ contains pairs $(\mathsf{label}_0, \mathsf{label}_1)$, and $\mathsf{LABELS}_{\mathsf{single}}$ consists of session identifiers label. We define the predicate $\mathsf{P}_{\mathsf{partner}}$ to specify correctness of these sets by checking all pairs $(\mathsf{label}_0, \mathsf{label}_1)$ are sessions which share the same session identifier, and all identifiers in the set $\mathsf{LABELS}_{\mathsf{single}}$ are sessions which are currently unpartnered. This is formally described in Figure 4.1.

Definition 27 (Session matching algorithm)**.** *A session matching algorithm $\mathcal{M} : \{0,1\}^* \to \{0,1\}^*$ for a key exchange protocol* ke *is an efficient algorithm such that the following holds for any adversary $\mathcal{A}$ playing against G_{BR}: after each response of the key exchange game, the algorithm $\mathcal{M}$ is given an ordered list of all queries and responses sent between $\mathcal{A}$ and $G_{\mathsf{BR},\mathcal{D}}$, along with the public parameters*

of the system. Algorithm $\mathcal{M}$ then outputs sets $\mathsf{LABELS}_{\mathsf{partner}}$ *and* $\mathsf{LABELS}_{\mathsf{single}}$ *such that, for the current list $\mathcal{L}_G$ of the game $G_{\mathsf{BR},\mathcal{D}}$, the sets* $\mathsf{LABELS}_{\mathsf{partner}}$, $\mathsf{LABELS}_{\mathsf{single}}$ *always satisfy the predicate* $\mathsf{P}_{\mathsf{partner}}(\mathcal{L}_{G_{\mathsf{ke}}}, \mathsf{LABELS}_{\mathsf{partner}}, \mathsf{LABELS}_{\mathsf{single}})$ *given in Figure 4.1.*

We remark that the idea of a session matching algorithm has already appeared in different forms in the literature. As mentioned above, in the original paper [BR93] the notion of matching conversations via the communication transcripts (and their order) supports a straightforward session matching algorithm. In [BR95] Bellare and Rogaway introduce a partner function which resembles our notion of a session matching algorithm, but their function does not need to be efficiently computable. Finally, in [BPR00] the authors require the session identifiers, defining essentially the partners, to be given to the adversary upon acceptance of a session, again yielding a session matching algorithm straightforwardly. As we show in Section 4.5, a weak form of session matching algorithm is in fact necessary to ensure secure composition.

We remark that, at a superficial glance, the session matching algorithm for the key exchange protocol seems to impose some restrictions on the communication privacy or anonymity for the symmetric key protocol. This, however, is not true, as session matching for key exchange does not refer to the actual usage of the derived keys in the subsequent protocol. In particular, the symmetric key protocol and its security game may well cover anonymity-related properties such as the key-hiding property [Fis99, AR00], i.e., which of two keys has been used to encrypt messages.

4.3 Single-Session Reducible Games

Usually, game based notions of protocol security require one to consider multiple sessions executed concurrently in order to draw conclusions about the security of the scheme. Notice that when different sessions of the protocol depend only on independent, efficiently samplable states, then it may be possible to reduce the security of the many session scenario to that of a single session. This greatly simplifies the analysis of the protocol and thus allows one to conclude security of the composed protocol more easily.

In symmetric key protocol games, all unknown keys are independent. Thus, in many cases one is able to analyse only the security of a single pair of sessions and, provided this is secure, may conclude the standard multi-session scenario is secure. For example, consider an authenticated channel. An adversary wins if any one session accepts some invalid (non-authenticated) message. It is clear that any adversary who is able to do this when there are multiple, concurrently executing sessions, will be able to achieve the same goal when there is only a single run of the protocol being executed. We note that for key exchange protocols, individual runs are not independent due to the session keys depending upon the shared long term asymmetric keys in some way.

The *single session game* is a symmetric key game where the adversary is allowed to query at most one NewKey query and one $\mathsf{SameKey}$ query, i.e., the adversary is given access to at most one pair of "honest" sessions. We denote this game by $G_{\pi-1}$. Note that any (multi-session) symmetric key game G_π has a single session version $G_{\pi-1}$. A

symmetric key game is called *single session reducible* if its (multi-session) security can be reduced to the security of the corresponding single session game.

Definition 28 (Single-Session Reducibility)**.** *A security game G_π is* single session reducible *if for any PPT adversary $\mathcal{A}$ against G_π where $\mathrm{Adv}^{G_\pi}_{\pi,\mathcal{A}}(1^\eta)$ is non-negligible, then there exists a PPT adversary $\mathcal{B}$ against $G_{\pi-1}$ such that $\mathrm{Adv}^{G_{\pi-1}}_{\pi,\mathcal{B}}(1^\eta)$ is non-negligible.*

We stress again that single session reducibility is not a prerequisite for our composition theorem to work. This class of protocols only supports a simpler analysis. In Chapter 4.6, we will see a necessary condition for single-session reducibility.

In Section 3.4, we show that the game of authenticated channels satisfies this condition. Hence, a single session secure authenticated channel remains secure when putting the protocol into a multi-session setting where the symmetric key generation of the protocol is replaced by a BR-secure key exchange protocol.

4.4 Composition Result

We now present our main results. In Theorem 1 we show that a BR-secure key exchange, with the additional property of having an efficient session matching algorithm, securely composes with a symmetric key protocol.

Theorem 1. *Let* ke *be a BR-forward-secure key exchange protocol w.r.t. $\mathcal{D}$, where an efficient session matching algorithm exists. Let π be a secure protocol w.r.t. G_π. If the key generation algorithm of π outputs keys with distribution $\mathcal{D}$ then the composition* $\mathsf{ke};\pi$ *is secure w.r.t. $G_{\mathsf{ke};\pi}$ and for any efficient $\mathcal{A}$ we have*

$$\mathrm{Adv}^{G_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) \leq {n_i}^2 \cdot n_s \cdot \mathrm{Adv}^{G_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta) + \mathrm{Adv}^{G_\pi}_{\pi,\mathcal{C}}(1^\eta)$$

for some efficient algorithms $\mathcal{B}$ and $\mathcal{C}$, where n_i is the maximum number of participants and n_s is the maximum number of sessions, and thus ${n_i}^2 \cdot n_s$ is the maximal size of the set LABELS*.*

The proof proceeds in two stages. First, we show that we can replace all the session keys one-by-one with random keys, where partner sessions are keyed with the same random value. This results in a composed game, where keys used by the symmetric protocol are independent of the key exchange. Next, we show this is then equivalent to the symmetric key protocol game G_π. Intuitively this means a break against this composition is a break against the symmetric key protocol, where keys are generated randomly.

We provide the formal proof of Theorem 1 after stating an corollary for single session reducible protocols which is an immediate application. Essentially, if a symmetric key protocol is single session reducible and secure for a single session, then it securely composes with a BR-secure key exchange protocol.

Corollary 1. *Let* ke *be a BR-secure key exchange protocol w.r.t.* $\mathcal{D}$*, where an efficient session matching algorithm exists. Let* G_π *be a single session reducible security game, and let* π *be a secure protocol w.r.t.* $G_{\pi-1}$*. If the key generation algorithm of* π *outputs keys with distribution* $\mathcal{D}$ *then the composition* $\mathsf{ke};\pi$ *is secure w.r.t.* $G_{\mathsf{ke};\pi}$*.*

Proof. Since π is secure w.r.t $G_{\pi-1}$, and G_π is single session reducible we have that π is secure w.r.t. G_π by definition. Therefore we can now apply Theorem 1 and the result holds. □

Proof of Theorem 1. To prove Theorem 1, a hybrid argument is involved, where one by one, the real keys used by the protocol are replaced by random keys. Each replacement is reduced to a BR-secrecy game to show that it only triggers a negligible loss in success probability. The session matching ensures that we can assign matching keys to partners in these hybrid games for valid simulations.

We now turn to the definitions needed to prove the hybrid argument. Let the game $G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}$ be the game $G_{\mathsf{ke};\pi}$ where for the first λ sessions to accept a key (where the partner session has not yet accepted), the key from the key exchange session is replaced by a random value for the π stage of the composition. The random value is drawn according to distribution $\mathcal{D}$ which corresponds to the output distribution of the key generation algorithm of π.

The game $G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}$ runs as for the game $G_{\mathsf{ke};\pi}$ with the following modifications. The game maintains the variable λ^*, which is set to 0 initially. The behaviour of $G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}$, is defined to act as game $G_{\mathsf{ke};\pi}$, on all queries except the Send query. When a Send query is made the game performs as described in Figure 4.2.

$\mathsf{Send}(\mathsf{label}, \mathsf{msg})$:
- If $\lambda \leq \lambda^*$, then act as $G_{\mathsf{ke};\pi}$ and return its output, else:
- Check whether $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{running}$.
- If not, return failure message false.
- Else, run $G_{\mathsf{ke}}(\mathsf{Send}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg}))$.
- If it returns msg', return msg'.
- If it returns $(\mathsf{msg}', \mathsf{accepted})$ or $\mathsf{accepted}$, then the key exchange game wrote $(\mathsf{sid}^*, \kappa^*)$ to its output tape.
- If there is $\mathsf{label}'_{\mathsf{ke}}$ with $\mathsf{label}'_{\mathsf{ke}}.\mathsf{sid} = \mathsf{sid}^*$ and $\mathsf{label}'_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$, then issue the query $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ to the protocol game G_π.
- Else if there is no such $\mathsf{label}'_{\mathsf{ke}}$ and if $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{fresh}$, then draw $\kappa_\pi \leftarrow \mathcal{D}$, increment λ^* by 1, write κ on the input tape of the protocol game G_π and query $\mathsf{NewKey}_\pi(U, V)$ to the protocol game G_π which returns a key identifier kid_π that is returned to the adversary, and the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ is stored in list $\mathcal{L}_{\mathrm{Identifiers}}$.
- Else if $\mathsf{label}.\mathrm{st}_{\mathrm{key}} = \mathsf{revealed}$, then query $\mathsf{SetKey}(\mathsf{label}.U, \mathsf{label}.V, \kappa^*)$ to G_π which returns a key identifier kid_π that is returned to the adversary, and the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_\pi)$ is stored in list $\mathcal{L}_{\mathrm{Identifiers}}$.
- Return all answers from the subgames to the adversary.

Figure 4.2: Send query for the game $G^{\lambda,\mathcal{D}}_{\mathsf{ke},\pi}$.

By using Lemma 1, one transforms the game $G_{\mathsf{ke};\pi} = G^{0,\mathcal{D}}_{\mathsf{ke};\pi}$ into the game $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$ for

$n = {n_i}^2 \cdot n_s$, where these two games are indistinguishable to the adversary due to the BR-security of the key exchange. As an immediate consequence of Lemma 1, we have

$$\left|\mathrm{Adv}^{G^{0,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) - \mathrm{Adv}^{G^{n,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta)\right| \leq n \cdot \mathrm{Adv}^{G_{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta).$$

The game $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$ now uses random keys which are independent from the keys derived in the key exchange protocol, and Lemma 2 then tells us that the advantage of an adversary against $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$ is equal to the advantage of an adversary against the G_π game for the symmetric key protocol. Since the protocol π is secure w.r.t. G_π, we therefore conclude the game $G_{\mathsf{ke};\pi}$ is secure. □

Lemma 1. *Let* ke *be a BR-secure key exchange protocol w.r.t. $\mathcal{D}$, where an efficient session matching algorithm exists. Let π be a symmetric key protocol whose key generation algorithm outputs keys with distribution $\mathcal{D}$. For all $\lambda = 1, \ldots, {n_i}^2 \cdot n_s$, for any efficient $\mathcal{A}$ we have*

$$\mathrm{Adv}^{G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) \leq \mathrm{Adv}^{G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) + \mathrm{Adv}^{G^{\mathsf{BR},\mathcal{D}}}_{\mathsf{ke},\mathcal{B}}(1^\eta)$$

for some efficient algorithm $\mathcal{B} = \mathcal{B}(\lambda)$.

We note that we give λ as auxiliary input to $\mathcal{B}$ for simplicity. For the full hybrid argument picking λ at random in the corresponding range actually suffices.

Proof. Given an adversary $\mathcal{A}$ against the game $G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}$, we construct an algorithm $\mathcal{B}$ against the BR-security of ke. If $\mathcal{A}$ has a non-negligible difference in advantage between the games $G^{\lambda-1,\mathcal{D}}_{\mathsf{ke};\pi}$ and $G^{\lambda,\mathcal{D}}_{\mathsf{ke};\pi}$, then algorithm $\mathcal{B}$ will have non-negligible advantage in the BR-secrecy game of ke.

Algorithm $\mathcal{B}$ honestly simulates the π stage of the composition, using the keys from the BR-secrecy game, $G^{\mathsf{BR},\mathcal{D}}_{\mathsf{ke}}$, and all key exchange queries are forwarded to the $G^{\mathsf{BR},\mathcal{D}}_{\mathsf{ke}}$ game. To allow $\mathcal{B}$ to simulate the π stage, $\mathcal{B}$ simulates the lists $\mathcal{L}_{G_\pi}$ and $\mathcal{L}^{\mathsf{keys}}_{G_\pi}$ and the variable λ^*. It keeps track of whether sessions have accepted or not using the partial function $\mathsf{ACC} : \mathsf{LABELS} \to \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$. Algorithm $\mathcal{B}$ maintains a list of session keys (for accepted sessions where the key is obtained through a Reveal or Test query) using the partial function $\mathsf{KST} : \mathsf{LABELS} \to \mathcal{D}$. Algorithm $\mathcal{B}$ also keeps track of all corrupt sessions within $G^{\mathsf{BR},\mathcal{D}}_{\mathsf{ke}}$, so locally constructs and updates a restricted copy of $\mathcal{L}^{\mathsf{keys}}_{G_{\mathsf{ke}}}$, which only contains the entries $(\mathsf{kid}, U, \mathrm{st}_{\mathrm{key}})$ and a restricted copy of $\mathcal{L}_{G_{\mathsf{ke}}}$ which only contains $(\mathsf{label}, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathrm{st}_{\mathrm{exec}}, \mathrm{st}_{\mathrm{key}})$. Moreover, $\mathcal{B}$ maintains a restricted list to match sessions from the key exchange game with keys from the protocol game via a restricted list $\mathcal{L}^{\mathrm{Identifiers}}$ which contains pair $(\mathsf{label}, \mathsf{kid})$ In descriptions of the algorithms $\mathcal{B}$, we denote these lists by $\mathcal{L}_{G_{\mathsf{ke}}}(\mathcal{B})$, $\mathcal{L}_{G^{\mathsf{keys}}_{\mathsf{ke}}}(\mathcal{B})$ and $\mathcal{L}^{\mathrm{Identifiers}}(\mathcal{B})$.

Algorithm $\mathcal{B}$ may also run the session matching algorithm $\mathcal{M}$, which outputs lists $\mathsf{LABELS}_{\mathsf{single}}$ and $\mathsf{LABELS}_{\mathsf{partner}}$ as described in Section 4.2. For the session matching algorithm $\mathcal{M}$ run by the adversary $\mathcal{B}$, which is playing the game $G^{\mathsf{BR},\mathcal{D}}_{\mathsf{ke}}$, we write $(\mathsf{LABELS}_{\mathsf{single}}, \mathsf{LABELS}_{\mathsf{partner}}) \leftarrow \mathcal{M}^{G^{\mathsf{BR},\mathcal{D}}_{\mathsf{ke}}}_{\mathcal{A}}$, where the sets $\mathsf{LABELS}_{\mathsf{single}}$ and $\mathsf{LABELS}_{\mathsf{partner}}$ are the outputs of the session matching algorithm as previously described.

We now describe how $\mathcal{B}$ ansers $\mathcal{A}$'s queries. When $\mathcal{A}$ makes a $\mathsf{Send}_{\mathsf{ke}}$ query, $\mathcal{B}$ answers as given in Figure 4.3. All other queries made by $\mathcal{A}$ are for the π stage of the composition, and are honestly simulated by $\mathcal{B}$.

$\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}, \mathsf{msg})$:
- $\mathcal{B}$ checks whether $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{running}$.
- If not, $\mathcal{B}$ returns failure message false.
- Else, $\mathcal{B}$ queries $(\mathsf{Send}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg}))$ to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$.
- If $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ returns msg', $\mathcal{B}$ return msg'.
- If $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ also returns $\mathsf{rejected}$, then $\mathcal{B}$ sets $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} := \mathsf{rejected}$.
- Else if $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ also returns $\mathsf{accepted}$, then $\mathcal{B}$ sets $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} := \mathsf{accepted}$. In this case, the key exchange game set the session key of this session to some value. $\mathcal{B}$ thus performs the following operations:
  - $(\mathsf{LABELS}_{\mathsf{single}}, \mathsf{LABELS}_{\mathsf{partner}}) \leftarrow \mathcal{M}_{\mathcal{B}}^{G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}}$.
  - If there is a pair $(\mathsf{label}, \mathsf{label}')$ or a pair $(\mathsf{label}', \mathsf{label})$ in $\mathsf{LABELS}_{\mathsf{partner}}$, then $\mathcal{B}$ sets $\mathsf{KST}(\mathsf{label}) := \mathsf{KST}(\mathsf{label}')$, creates the tuple $(\mathsf{label}, \mathsf{label}'.\mathsf{kid})$ in list in $\mathcal{L}_{\mathrm{Identifiers}}(\mathcal{B})$ and queries $\mathsf{SameKey}(\mathsf{label}'.V, \mathsf{label}'.U, \mathsf{label}'.\mathsf{kid})$ to the simulated protocol game G_π that returns the key identifier $\mathsf{kid} = \mathsf{label}.\mathsf{kid}$, which $\mathcal{B}$ relays to $\mathcal{A}$.
  - Else if $\mathsf{label} \in \mathsf{LABELS}_{\mathsf{single}}$ and $\mathsf{label}.\mathrm{st}_{\mathrm{key}} = \mathsf{revealed}$, then $\mathcal{B}$ queries $\mathsf{Reveal}_{\mathsf{ke}}(\mathsf{label})$ to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ that returns a key κ. $\mathcal{B}$ sets $\mathsf{KST}(\mathsf{label}) := \kappa$ and queries $\mathsf{SetKey}(\mathsf{label}.U, \mathsf{label}.V, \kappa)$ to the simulated protocol stage, which returns a key identifier kid. $\mathcal{B}$ adds $(\mathsf{label}, \mathsf{kid})$ to $\mathcal{L}_{\mathrm{Identifiers}}(\mathcal{B})$ and returns kid to $\mathcal{A}$.
  - Else, $\mathsf{label} \in \mathsf{LABELS}_{\mathsf{single}}$ and $\mathsf{label}.\mathrm{st}_{\mathrm{key}} = \mathsf{fresh}$. Depending on λ, $\mathcal{B}$ operates as follows:
    - If $\lambda < \lambda^*$, then $\mathcal{B}$ draws a random key $\kappa \mathrm{exec}\mathcal{D}$, sets $\mathsf{KST}(\mathsf{label}) := \kappa$, writes κ on the key input tape $G_\pi^{\mathsf{k\text{-}in}}$ tape and queries $\mathsf{NewKey}(\mathsf{label}.U, \mathsf{label}.V)$ to the simulated game G_π, which returns a key identifier kid. $\mathcal{B}$ adds $(\mathsf{label}, \mathsf{kid})$ to $\mathcal{L}_{\mathrm{Identifiers}}(\mathcal{B})$ and returns kid to $\mathcal{A}$.
    - If $\lambda = \lambda^*$, then $\mathcal{B}$ queries $\mathsf{Test}(\mathsf{label})$ to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ to receive a key κ. Then, $\mathcal{B}$ sets $\mathsf{KST}(\mathsf{label}) := \kappa$, writes κ on the key input tape $G_\pi^{\mathsf{k\text{-}in}}$ tape and queries $\mathsf{NewKey}(\mathsf{label}.U, \mathsf{label}.V)$ to the simulated game G_π, which returns a key identifier kid. $\mathcal{B}$ adds $(\mathsf{label}, \mathsf{kid})$ to $\mathcal{L}_{\mathrm{Identifiers}}(\mathcal{B})$ and returns kid to $\mathcal{A}$.
    - If $\lambda > \lambda^*$, then $\mathcal{B}$ queries $\mathsf{Reveal}(\mathsf{label})$ to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ to receive a key κ. Then, $\mathcal{B}$ sets $\mathsf{KST}(\mathsf{label}) := \kappa$, writes κ on the key input tape $G_\pi^{\mathsf{k\text{-}in}}$ tape and queries $\mathsf{NewKey}(\mathsf{label}.U, \mathsf{label}.V)$ to the simulated game G_π, which returns a key identifier kid. $\mathcal{B}$ adds $(\mathsf{label}, \mathsf{kid})$ to $\mathcal{L}_{\mathrm{Identifiers}}(\mathcal{B})$ and returns kid to $\mathcal{A}$.

Figure 4.3: The response of algorithm $\mathcal{B}$ to a Send query made by $\mathcal{A}$ playing the $G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}$ game.

Remark: In Figure 4.3, when $\lambda^* > \lambda$ and there exists an entry for label in $\mathsf{LABELS}_{\mathsf{partner}}$, we initialise the session key of this session with the key of its partner session. Since ke is BR-secure, we have that the matching game $G_{\mathsf{ke}}^{\mathsf{Match}}$ is secure, and hence the two partnered completed sessions will share the same session key with overwhelming probability. Hence our initialisation of session keys is correct.

Since $\mathcal{B}$'s local copy of $\mathcal{L}_{G_{\mathsf{ke}}}$ contains identical information relating to the reveal state of sessions as the one maintained by $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$, it is able to either initialise a session at the π stage by a NewKey query, if the session is unrevealed or by a SetKey query, if the session is revealed. Note that $\mathcal{B}$ asking Reveal queries to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ does not affect whether

keys are considered known or not within its simulation of the π stage. Thus the quality of the simulation is not affected by $\mathcal{B}$'s additional Reveal queries. Notice that if the Test query made by $\mathcal{B}$ to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ returns the real key then $\mathcal{B}$ perfectly simulates the $G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}$ game, while if a random key is returned $\mathcal{B}$ perfectly simulates the $G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}$ game. The advantage of $\mathcal{B}$ in game $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ corresponds to the difference in success probability of $\mathcal{A}$ upon playing $G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}$ or $G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}$.

At some point algorithm $\mathcal{A}$ terminates. If $\mathcal{A}$ wins against the composed game, $\mathcal{B}$ sends the query $\mathsf{Guess}(1)$ to $G_{\mathsf{BR},\mathcal{D}}$ and otherwise $\mathcal{B}$ sends $\mathsf{Guess}(0)$. We have

$$\Pr[\mathrm{Exp}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D},0}}(1^\eta) = 0] = \mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}}(1^\eta)$$

and

$$\Pr[\mathrm{Exp}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D},1}}(1^\eta) = 1] = \mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}}(1^\eta).$$

This gives

$$\begin{aligned}\mathrm{Adv}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}}(1^\eta) &= \left| \Pr\left[\mathrm{Exp}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D},0}}(1^\eta) = 0\right] \right. \\ &\qquad \left. - \Pr\left[\mathrm{Exp}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D},1}}(1^\eta) = 1\right] \right| \\ &= \left| \mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}}(1^\eta) - \mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}}(1^\eta) \right| \leq \epsilon(1^\eta),\end{aligned}$$

where $\epsilon(1^\eta)$ is a negligible function in the security parameter, denoting the advantage against the BR-secrecy game. Thus

$$\mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{\lambda-1,\mathcal{D}}}(1^\eta) \leq \mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{\lambda,\mathcal{D}}}(1^\eta) + \mathrm{Adv}_{\mathsf{ke},\mathcal{B}}^{G^{\mathsf{BR},\mathcal{D}}}(1^\eta).$$

□

The final step of Theorem 1 is to show the game $G_{\mathsf{ke};\pi}^{n,\mathcal{D}}$, where all session keys of the key exchange are replaced by random keys, can be reduced to the security of the symmetric key protocol game G_π. We now show this in Lemma 2.

Lemma 2. *Let* ke *be a key exchange protocol, let* π *be a symmetric key protocol whose key generation algorithm produces keys w.r.t. distribution* $\mathcal{D}$*. Let* $n = {n_i}^2 \cdot n_s$*, where* n_i *is an upper bound on the number of users and* n_s *is an upper bound on the number of sessions. Then for any efficient adversary* $\mathcal{A}$ *we have*

$$\mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}^{n,\mathcal{D}}}(1^\eta) \leq \mathrm{Adv}_{\pi,\mathcal{B}}^{G_\pi}(1^\eta)$$

for some efficient adversary $\mathcal{B}$*.*

Proof. The outline of the proof is as follows: Algorithm $\mathcal{B}$ plays against a game G_π and internally simulates honestly the entire composed game $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$. As the keys used in the protocol stage are independent of the key exchange stage, $\mathcal{B}$ can answer $\mathcal{A}$'s queries to the key exchange stage by its simulated composed game, while forwarding $\mathcal{A}$'s queries to the protocol stage to G_π. The outputs to $\mathcal{A}$ are perfectly identical to the composed game $\mathcal{A}$ expects to play against.

Formally, given the adversary $\mathcal{A}$ against the game $G^{n,\mathcal{D}}_{\mathsf{ke};\pi}$ we construct algorithm $\mathcal{B}$ playing the G_π game as follows. Algorithm $\mathcal{B}$ internally simulates the entire key exhcnage game as is done in the composed game. We now describe how $\mathcal{B}$ answers $\mathcal{A}$'s queries. If the query is to the π stage of the composed game, $\mathcal{B}$ forwards this query to G_π and returns the response to $\mathcal{A}$. If the query is for the ke stage then $\mathcal{B}$ uses its internal data of G_{ke} to simulate the actions of the composed game and create a response to return to $\mathcal{A}$. Note that for all these queries the simulation is perfect. The Send query is formally given in Figure 4.4.

$\mathsf{Send}(\mathsf{label}, \mathsf{msg})$:
- $\mathcal{B}$ checks whether $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{running}$.
- If not, it returns failure message false.
- Else, $\mathcal{B}$ runs the simulated game $G_{\mathsf{ke}}(\mathsf{Send}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg}))$.
- If G_{ke} returns msg', $\mathcal{B}$ returns msg' to $\mathcal{A}$.
- If G_{ke} returns $(\mathsf{msg}', \mathsf{accepted})$ or $\mathsf{accepted}$, then the key exchange game wrote $(\mathsf{sid}^*, \kappa^*)$ to its output tape.
- If there is $\mathsf{label}'_{\mathsf{ke}}$ with $\mathsf{label}'_{\mathsf{ke}}.\mathsf{sid} = \mathsf{sid}^*$ and $\mathsf{label}'_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$, then $\mathcal{B}$ issues the query $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ to the protocol game G_π.
- Else if $\mathsf{label}.\mathrm{st}_{\mathrm{key}} \neq \mathsf{revealed}$, then $\mathcal{B}$ queries $\mathsf{NewKey}_\pi(U, V)$ to the protocol game G_π which returns a key identifier kid_π that is returned to the adversary, and the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ is stored in list $\mathcal{L}_{\mathrm{Identifiers}}$.
- Else if $\mathsf{label}.\mathrm{st}_{\mathrm{key}} = \mathsf{revealed}$, then query $\mathsf{SetKey}(\mathsf{label}.U, \mathsf{label}.V, \kappa^*)$ to G_π.
- Return all answers from the subgames to the adversary.

Figure 4.4: Simulation by algorithm $\mathcal{B}$ in response to a Send query of $\mathcal{A}$ playing $G^{n,\mathcal{D}}_{\mathsf{ke};\pi,\mathcal{A}}$.

Keys used by G_π and the keys used by the protocol stage of the composed game are identically distributed, as in the case of a $\mathsf{SetKey}(U, V, \kappa)$ query, the key is set to κ in both games. When a $\mathsf{NewKey}(U, V)$ query is sent, both games randomly draw a key from distribution $\mathcal{D}$. If the adversary queries $\mathsf{SameKey}(V, U, \mathsf{kid})$, in both games, there is a tuple $(\mathsf{kid}, V, U, \kappa, \mathrm{st}_{\mathsf{kid}})$ created on $\mathcal{L}^{\mathsf{keys}}_G$. Thus, the simulation is sound.

At some point $\mathcal{A}$ will terminate execution and at this point $\mathcal{B}$ also terminates. If $\mathcal{A}$ has won against the composed game, then $\mathcal{B}$ will have won against the G_π game. Hence we have,

$$\mathrm{Adv}^{G^{n,\mathcal{D}}_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) \leq \mathrm{Adv}^{G_\pi}_{\pi,\mathcal{B}}(1^\eta).$$

□

4.5 Observations on Session Matching

In Section 4.2 we defined a *session matching algorithm*, and gave an example of a BR-secure key exchange that does not support such an algorithm (based on re-randomizable encryption). Moreover, we showed that a BR-secure key exchange is composable if there exists a session matching algorithm $\mathcal{M}$ which, at any point in the game, outputs correct lists of partnered sessions. This section is devoted to prove that the converse also holds, i.e., if a key exchange protocol is composable in general, then a weak form of session matching algorithm exists. In other words, if for all secure protocols π, there is a black-box reduction from the composed game $G_{\mathsf{ke};\pi}$ to BR-security of the key exchange, then this black-box reduction can be used to build some session matching algorithm. This shows that a form of session matching algorithm is both necessary and sufficient to provide general composability for BR-secure key exchange protocols.

We first specify the notion of a straightline black-box reduction, which works for any protocol π, and any adversary $\mathcal{A}$. The reduction reduces the security of the composed protocol $(\mathsf{ke};\pi)$, to the BR-security of ke. The reduction is black-box and has oracle access to a single copy of $\mathcal{A}$, i.e., it may only query the oracle $\mathcal{A}$ via a certain interface, but may not set randomness for it, run several copies of $\mathcal{A}$ or re-set $\mathcal{A}$ to its initial state, as more powerful definitions of black-box reduction sometimes allow. We first show the reduction in our composition proof is of this type, and then move on to give the construction of a "weak" session matching algorithm.

We stress that the derived session matching algorithm is not as powerful as the one defined in Definition 27. Namely, one of the differences will be that the algorithm only provides a good session matching for those adversaries $\mathcal{A}$ that receive additional session key information from the key exchange game. Moreover, the weak session matching algorithm will produce the correct result, with some non-negligible probability better than a random guess, i.e., it is not required to always succeed. Note that the (non-weak) session matching algorithm must produce the correct result with probability 1. Finally, the weak session matching algorithm also makes additional Test and Reveal queries, but only in a strictly controlled manner. In particular, we assume that they do not affect the winning probability of an adversary that plays the BR-game $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$.

Definition 29 (Straightline Black-Box Reduction)**.** *Let $\mathcal{A}$ be an adversary against $G_{\mathsf{ke};\pi}$. The reduction accesses $\mathcal{A}$ via oracle queries. The $\mathcal{A}$ oracle is given the secret bit b of the BR-secrecy game, lists $\mathsf{LABELS}_{\mathsf{partner}}$, $\mathsf{LABELS}_{\mathsf{single}}$, and the correct session key, whenever a key exchange session accepts. This information flow is realized through a special tape between oracle $\mathcal{A}$ and $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ which the reduction is unable to read. Let π be secure with respect to G_{π}. We say that there exists a* straightline black-box reduction *from $G_{\mathsf{ke};\pi}$ to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ if there exists a PPT algorithm $\mathcal{B}$ against $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$, such that for all $\mathcal{A}$ the following conditions hold:*

1. *If $\mathrm{Adv}_{\mathsf{ke};\pi,\mathcal{A}}^{G_{\mathsf{ke};\pi}}(1^{\eta})$ is non-negligible, then $\mathrm{Adv}_{\mathsf{ke},\mathcal{B}}^{G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}}(1^{\eta})$ is non-negligible.*
2. *Algorithm $\mathcal{B}$ has oracle access to a single oracle $\mathcal{A}$.*

3. *Algorithm $\mathcal{B}$ honestly relays any queries $\mathcal{A}$ makes to the* ke *stage of* $(\mathsf{ke};\pi)$ *to the game* $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$,

4. *and the only other queries made by $\mathcal{B}$ to G_{BR} are* Test *and* Reveal *queries.*

The above notion may sound restrictive; adversary $\mathcal{A}$ receives information from $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ that is unknown to $\mathcal{B}$. Additionally, algorithm $\mathcal{B}$ is merely an evesdropper as far as queries to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ are concerned. However, we can observe that our reduction in Theorem 1 is of the above type. In particular, $\mathcal{B}$ does not tamper with queries to the key exchange game.

We now prove that the notion of a straightline black-box reduction implies the existence of a weak session matching algorithm.

To construct a weak session matching algorithm we now define a particular protocol π_0 with its game G_{π_0}. The protocol π_0 consists of two algorithms (kg,ξ) which act as follows: the key generation algorithm kg outputs a random key from $\{0,1\}^\eta$. The protocol algorithm ξ on any type of input always returns the empty message as a response.

Besides the standard NewKey, $\mathsf{SameKey}$ and SetKey, G_{π_0} provides one additional query, $\mathsf{Target}(\mathsf{kid},\kappa)$. When the adversary has asked $\mathsf{Target}(\mathsf{kid},\kappa)$, the game ignores all further queries. The predicate P_{π_0} of the game G_{π_0} checks whether kid corresponds to an honest session, i.e., whether list $\mathcal{L}_{G_{\pi_0}}^{\mathsf{keys}}$ stores $\mathsf{kid}.\mathrm{st}_{\mathsf{kid}} = \mathsf{honest}$, and whether κ is the key corresponding to this key identifier. If so, P_{π_0} outputs 1, else P_{π_0} outputs 0.

If, before the Target query, G_{π_0} receives any of the queries NewKey, $\mathsf{SameKey}$ or SetKey, G_{π_0} behaves as described in Section 3.3. Additionally, after every such query, the game returns two lists $\mathsf{LABELS}_{\mathsf{partner}}$ and $\mathsf{LABELS}_{\mathsf{single}}$ that contain pairs of keys that are used by two users and keys that are only used by one user so far, i.e., a different encoding of a shortened list $\mathcal{L}_{G_{\pi_0}}^{\mathsf{keys}}$, that only contains tuples (kid,U,V) instead of $(\mathsf{kid},U,V,\kappa,\mathrm{st}_{\mathsf{kid}})$. At some point, G_{π_0} receives a pair (kid,κ) and outputs 1 if and only if $\mathsf{kid}.\mathrm{st}_{\mathsf{kid}} = \mathsf{honest}$ and $\kappa = \mathsf{kid}.\kappa$. Note that for random keys κ, the corresponding protocol π_0 is secure (as a standalone protocol), since winning the game requires the adversary to predict an unknown key, while the key is information-theoretically hidden.

Algorithm $\mathcal{B}$ is required to work for all adversaries satisfying the description of Definition 29. By considering a particular subclass of those, we will be able to extract information on $\mathcal{B}$'s ability to provide matching sessions. Let $\mathcal{A}$ be an arbitrary adversary against $G_{\mathsf{ke};\pi_0}$ that does not receive the additional key information from $G_{\mathsf{BR},\mathcal{D}}$. We now describe the following universal wrapper algorithm $\mathcal{A}_0$ for such adversaries.

Algorithm $\mathcal{A}_0$ receives the additional information and runs $\mathcal{A}$ as a subroutine. Algorithm $\mathcal{A}_0$ plays against G_{ke,π_0}. and does not modify any of $\mathcal{A}$'s queries, except the Target query. Note that after the Target query is issued, no further queries need be made by the adversaries. When $\mathcal{A}$ issues its $\mathsf{Target}(\mathsf{kid},\kappa)$ query then let us assume that w.l.o.g. this is always a query for an honest session label. Now, throughout the game, $\mathcal{A}_0$ received lists $\mathsf{LABELS}_{\mathsf{partner}}$ and $\mathsf{LABELS}_{\mathsf{single}}$ from G_{π_0}. Algorithm $\mathcal{A}_0$ checks whether at any time in the game, these lists are different from those given in the additional information to $\mathcal{A}_0$. If a difference occurs, then $\mathcal{A}_0$ does not modify $\mathcal{A}$'s output $\mathsf{Target}(\mathsf{kid},\kappa)$ but simply

forwards it. Else, $\mathcal{A}_0$ searches in its lists for the correct key κ' and outputs $\mathsf{Target}(\mathsf{kid}, \kappa')$ as its final output.

Clearly, the algorithm $\mathcal{A}_0$ wins against the composed game with probability 1, as the lists always match and the key output in the Target session is always correct. However, when the reduction $\mathcal{B}$ performs a simulation and cannot provide suitable matchings, this may no longer hold. Nevertheless, as $\mathcal{A}_0$ has non-negligible winning advantage in the composed game, by definition, the reduction $\mathcal{B}$ with oracle access to $\mathcal{A}_0$ also has a non-negligible advantage in the BR-secrecy game. To assure that $\mathcal{A}_0$ produces useful output for $\mathcal{B}$, the adversary $\mathcal{B}$ needs to provide correct lists $\mathsf{LABELS}_{\mathsf{partner}}$ and $\mathsf{LABELS}_{\mathsf{single}}$ in each step. Else, $\mathcal{B}$ only observes an execution of a copy of $\mathcal{A}$, and $\mathcal{A}$ does not receive additional information about the keys. Thus, as $\mathcal{A}$ is just an arbitrary adversary without additional information, there is an algorithm $\mathcal{B}$, which is able to break the BR-secrecy of ke, contrary to the assumption (namely, the algorithm $\mathcal{B}$ which runs $\mathcal{A}$ as a subroutine). Therefore, $\mathcal{B}$ needs to provide an accurate matching at least in a significant number of cases.

The last step is to analyse how often $\mathcal{B}$ may fail to provide a good matching or admissible Reveal and Test queries. Analysis shows that, with high probability $\mathcal{B}$ provides admissible Reveal and Test queries and it achieves the correct session matching in a significant number of cases. Note that for the latter, the probability of a random guess for the matching being correct is negligible; therefore, with non-negligible probability, algorithm $\mathcal{B}$ produces a better result than a random guess. Thus, the constructed weak session matching algorithm indicates that composability is not achievable without some session matching properties of the key exchange protocol. We now turn to the analysis.

We now turn to the analysis. We say that $\mathcal{B}$ only makes admissible Reveal and Test queries, if $\mathcal{B}$ neither reveals the partner of a tested session, nor tests the partner of the revealed session. Recall that in either case, $\mathcal{B}$ loses in the BR-secrecy game. As $\mathcal{B}$ is only a successful algorithm when winning in the BR-secrecy game with probability significantly greater than $\frac{1}{2}$, it is obvious that $\mathcal{B}$ needs to provide admissible Test and Reveal queries with probability significantly greater than $\frac{1}{2}$. We now argue that this probability needs to be negligibly close to 1 by considering modified wrappers $\mathcal{A}_p$.

Let $p(\eta)$ be a positive, monotone function in the security parameter η, and let $p(\eta)$ be upper bounded by 1. Algorithm $\mathcal{A}_p$ flips a weighted coin. With probability $1 - p(\eta)$, algorithm $\mathcal{A}_p$ behaves as $\mathcal{A}_0$ and provides helpful information to $\mathcal{B}$. With probability $p(\eta)$, $\mathcal{A}_p$ only forwards $\mathcal{A}$'s output, so $\mathcal{B}$ does not receive any helpful information in this case. Note that $\mathcal{B}$ cannot distinguish these two cases due to the BR-security of the key exchange. Thus, $\mathcal{B}$'s probability of providing admissible queries is equal to some probability $q(\eta)$ in both cases.

The reduction $\mathcal{B}$'s success probability is lower bounded by $(1 - p(\eta)) \cdot (1 - q(\eta)) + p(\eta) \cdot \frac{1}{2}(1 - q(\eta)) = (1 - q(\eta))(\frac{1}{2} + \frac{1}{2}(1 - p(\eta)))$. Algorithm $\mathcal{A}_p$ has non-negligible winning probability, whenever $1 - p(\eta)$ is non-negligible in the security parameter η. Therefore, in order to exceed $\frac{1}{2}$ by a non-negligible amount, the term $(1 - q(\eta))$ must be negligibly close to 1. Hence, $\mathcal{B}$ provides admissible Test and Reveal queries in almost all cases.

We now argue that $\mathcal{B}$ provides matching sessions to its oracle $\mathcal{A}_0$ with non-negligible

probability. Recall that $\mathcal{B}$ needs to provide matching sessions to $\mathcal{A}_0$, as else, the oracle $\mathcal{A}_0$ does not pass any helpful information to $\mathcal{B}$. Thus, $\mathcal{B}$ provides matching sessions to $\mathcal{A}$ with non-negligible probability.

An analysis similar to the one for admissible queries fails, as providing non-matching sessions does not prevent $\mathcal{B}$ from winning the BR-secrecy game. In particular, when flipping a coin, $\mathcal{B}$ wins the game with probability $\frac{1}{2}$. If, for example, $\mathcal{B}$ can check whether it provides a good session matching to its oracle, no conclusions can be made. Thus, the matching property is only achieved in a weak flavor. Recall that $\mathcal{B}$ is significantly more successful in identifying partnered sessions than a purely random guess.

4.6 On Non-Forward-Secure Protocols

We prove that a symmetric-key protocol can be safely composed with a non-forward secure key exchange protocol as long as the symmetric-key protocol is defined via a *session-restricted* protocol. Jumping ahead to Chapter 5, we note that it is actually sufficient to be reducible to a session-restricted protocol in a key-independent way. We first define session-restricted games and then prove the composition theorem. Note that, throughout this section, we work with the notion of protocol games, as introduced in Chapter 3.

4.6.1 Session-Restricted Games

We here build on the definition of single-session games. Basically, a session-restricted game G_π for protocol π is a conglomeration of independent single- session games $G_{\pi-1}$, which, as we recall, only allow for a single NewKey and $\mathsf{SameKey}$ query. As $G_{\pi-1}$ only maintains a single key, there is no need for key identifiers kid. Thus, the list $\mathcal{L}_{G_{\pi-1}}$ only contains tuples $(U, V, k, \mathrm{st}_{\mathsf{kid}})$ instead of tuples $(U, V, k, \mathrm{st}_{\mathsf{kid}})$. We also drop the key identifier kid from all queries to $G_{\pi-1}$. In turn, G_π matches each independent copy of $G_{\pi-1}$ by a key identifier kid, we denote this copy by $G_{\pi-1}(\mathsf{kid})$. Moreover, we add a $\mathsf{Target}(\mathsf{kid})$ query to the game. This query determines which of the subgames defines the output.

We now describe the behaviour of G_π formally.

- On input a query $\mathsf{NewKey}(U, V)$, the game G_π initializes a new, independent copy of $G_{\pi-1}$ and a key identifier kid, that it returns to the adversary.
- On input a query $\mathsf{SameKey}(\mathsf{kid}, U, V)$, the game G_π relays the query $\mathsf{SameKey}(U, V)$ to $G_{\pi-1}$.
- On input any other query $Q(\mathsf{kid})$, G_π relays Q to $G_{\pi-1}$ and returns the answer. This also applies to $\mathsf{Corrupt}$ queries.
- If in the end, the adversary has not made any Target query, he looses. Else, the subgame $G_{\pi-1}(\mathsf{kid})$ whose key identifier kid was used on the Target query, returns its output b, and G_π returns b.

We will now prove that if a game is session-restricted, then it is also single-session reducible.

Lemma 3. *Let G_π be a session-restricted game, then G_π is also single-session reducible, in particular, G_π is secure if and only if $G_{\pi-1}$ is secure:*

$$\mathrm{Adv}^{G_\pi}_{\pi,\mathcal{A}}(1^\eta) \leq n \cdot \mathcal{A}^{G_{\pi-1}}_{\pi,\mathcal{B}}(1^\eta) \leq n \cdot \mathcal{A}^{G_\pi}_{\pi,\mathcal{B}}(1^\eta)$$

for some efficient adversary $\mathcal{B}$, where n is an upper bound on the NewKey *queries by $\mathcal{A}$.*

Proof. Let $G_{\pi-1}$ be the subgame that G_π runs independent copies of. It is clear that security in the multi-session case implies security in the single-session case with a tight security reduction. We thus only prove the converse direction. Let $\mathcal{A}$ be an adversary against G_π. We construct an adversary $\mathcal{B}$ against $G_{\pi-1}$ as follows. Let n be an upper bound on the number of NewKey queries that $\mathcal{A}$ makes. In the beginning of the game, $\mathcal{B}$ draws a random number k_0 between 1 and n. Then, $\mathcal{B}$ simulates a run of G_π. When $\mathcal{A}$ makes its k_0's NewKey query, instead of initializing a new copy of $G_{\pi-1}$ as G_π would do, $\mathcal{B}$ relays all queries to the game $G_{\pi-1}$, it is playing against. $\mathcal{B}$ then generates a key identifier kid for this session according to the distribution defined by $G_{\pi-1}$. Any query that comes prepended with kid, is relayed to the exterior game. All answers to other queries are faithfully simulated. When $\mathcal{A}$ terminates, the adversary $\mathcal{B}$ terminates too.

By construction, the simulation is perfect. The advantage of $\mathcal{B}$ is $\frac{1}{n}$ of the advantage of $\mathcal{A}$ and thus non-negligible. □

4.6.2 Composition Theorem

We now prove that key exchange protocols that are BR-secure, but not forward-secure can be safely composed with session-restricted games.

Theorem 2. *Let* ke *be a BR-secure key exchange protocol w.r.t. $\mathcal{D}$, where an efficient session matching algorithm exists. Let π be a secure protocol w.r.t. a session-restricted game G_π. If the key generation algorithm of π outputs keys with distribution $\mathcal{D}$ then the composition* ke; π *is secure w.r.t. $G_{\mathsf{ke};\pi}$ and for any efficient $\mathcal{A}$ we have*

$$\mathrm{Adv}^{G_{\mathsf{ke};\pi}}_{\mathsf{ke};\pi,\mathcal{A}}(1^\eta) \leq \mathrm{Adv}^{G^{\mathsf{BR},\mathcal{D}}_{\mathsf{ke}}}_{\mathsf{ke},\mathcal{B}}(1^\eta) + n \cdot \mathrm{Adv}^{G_{\pi-1}}_{\pi,\mathcal{C}}(1^\eta) \leq \mathrm{Adv}^{G^{\mathsf{BR},\mathcal{D}}_{\mathsf{ke}}}_{\mathsf{ke},\mathcal{B}}(1^\eta) + n \cdot \mathrm{Adv}^{G_\pi}_{\pi,\mathcal{C}}(1^\eta)$$

for some efficient algorithms $\mathcal{B}$ and $\mathcal{C}$, where n is an upper bound on the number of accepting sessions of the key exchange protocol.

Proof of Theorem 2. The second inequality follows from Lemma 3. We now prove the first inequality, namely, if there is a successful aversary $\mathcal{A}$ against $G_{\mathsf{ke},\pi}$, then there is either a successful adversary $\mathcal{B}$ against $G^{\mathsf{nfs},\mathsf{BR},\mathcal{D}}_{\mathsf{ke}}$ or a successful adversary $\mathcal{C}$ against $G_{\pi-1}$. Here, the proof is similar to the one of Theorem 1, except for the hybrid argument which was used to replace all keys one by one. Instead of defining a sequence of games, we only replace one key and then reduce to a single-session version $G_{\pi-1}$ of the protocol game instead of reducing to a multi-session version, as was done for Theorem 1.

We now define the modified game $G_{\mathsf{ke};\pi,n}$ where one of the keys is replaced by a random key. We will then show an adversary $\mathcal{B}$ that has non-negligible winning probability

against $G_{\mathsf{ke}}^{\mathsf{nfs},\mathsf{BR},\mathcal{D}}$, if $\mathcal{A}$'s winning probability changes significantly between $G_{\mathsf{ke};\pi,n}$ and $G_{\mathsf{ke};\pi}$.

$G_{\mathsf{ke};\pi,n}$ keeps the same state as $G_{\mathsf{ke};\pi}$ and an additional value n, where for the nth session, the key is replaced by a random value for the π stage of the composition. The random value is drawn according to distribution $\mathcal{D}$ which corresponds to the output distribution of the key generation algorithm of π.

The game $G_{\mathsf{ke};\pi,n}$ runs as for the game $G_{\mathsf{ke};\pi}$ with the following modifications. The game maintains the variable λ^*, which is set to 0 initially. The behaviour of $G_{\mathsf{ke};\pi,n}$, is defined to act as game $G_{\mathsf{ke};\pi}$, on all queries except the Send query. When a Send query is made the game performs as described in Figure 4.5.

$\mathsf{Send}(\mathsf{label}, \mathsf{msg})$:
- If $n \neq \lambda^*$, then act as $G_{\mathsf{ke};\pi}$ and return its output. If NewKey query is made, increment λ^* by 1, else:
- Check whether $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{running}$.
- If not, return failure message false.
- Else, run $G_{\mathsf{ke}}(\mathsf{Send}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg}))$.
- If it returns msg', return msg'.
- If it returns $(\mathsf{msg}', \mathsf{accepted})$ or $\mathsf{accepted}$, then the key exchange game wrote $(\mathsf{sid}^*, \kappa^*)$ to its output tape.
- If there is $\mathsf{label}'_{\mathsf{ke}}$ with $\mathsf{label}'_{\mathsf{ke}}.\mathsf{sid} = \mathsf{sid}^*$ and $\mathsf{label}'_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{accepted}$, then issue the query $\mathsf{SameKey}_{\pi}(V, U, \mathsf{kid}_{\pi})$ to the protocol game G_{π}.
- Else if there is no such $\mathsf{label}'_{\mathsf{ke}}$ and if $\mathsf{label}_{\mathsf{ke}}.\mathrm{st}_{\mathrm{exec}} = \mathsf{fresh}$, then draw $\kappa_{\pi} \leftarrow \mathcal{D}$, increment λ^* by 1, write κ on the input tape of the protocol game G_{π} and query $\mathsf{NewKey}_{\pi}(U, V)$ to the protocol game G_{π} which returns a key identifier kid_{π} that is returned to the adversary, and the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_{\pi})$ is stored in list $\mathcal{L}_{\mathrm{Identifiers}}$.
- Else if $\mathsf{label}.\mathrm{st}_{\mathrm{key}} = \mathsf{revealed}$, then query $\mathsf{SetKey}(\mathsf{label}.U, \mathsf{label}.V, \kappa^*)$ to G_{π} which returns a key identifier kid_{π} that is returned to the adversary, and the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\pi})$ is stored in list $\mathcal{L}_{\mathrm{Identifiers}}$.
- Return all answers from the subgames to the adversary.

Figure 4.5: Send query for the game $G_{\mathsf{ke},\pi,n}$.

We now describe the adversary $\mathcal{B}$ against. To simulate the games $G_{\mathsf{ke};\pi}$ and $G_{\mathsf{ke};\pi,n}$, respectively, $\mathcal{B}$ will make use of the session-matching algorithm. It will simulate G_{π} faithfully and pass on the keys to G_{π} by asking Reveal queries to $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$. For a random accepted session $k \leftarrow \{1, ..., n\}$, the adversary $\mathcal{B}$ will use the Test-query instead of the Reveal query. If the secret bit of the $G_{\mathsf{ke}}^{\mathsf{BR},\mathcal{D}}$ game is 0, then $\mathcal{B}$ perfectly simulates $G_{\mathsf{ke};\pi}$; if the secret bit is 0, $\mathcal{B}$ perfectly simulates $G_{\mathsf{ke};\pi,n}$. When $\mathcal{A}$ terminates, then $\mathcal{B}$ evaluates whether $\mathcal{A}$ has won against the internally simulated game G_{π}. $\mathcal{B}$ returns 1, if $\mathcal{A}$ has won, and 0 else. If $\mathcal{A}$'s success probability differs between $G_{\mathsf{ke};\pi}$ and $G_{\mathsf{ke};\pi,n}$, then this difference corresponds to $\mathcal{B}$'s success probability. We omit the formal details of the simulation, as they are analogous to Theorem 1.

We now prove that if $\mathcal{A}$ is successful against $G_{\mathsf{ke};\pi,n}$, then there is a successful adversary $\mathcal{C}$ against $G_{\pi-1}$. Analogously to the proof of Lemma 3, the adversary $\mathcal{C}$ simulates the game $G_{\mathsf{ke};\pi,n}$ except for the G_{π} session key, where the key from the key exchange

stage is replaced by a random key. For this session key, the adversary $\mathcal{C}$ relays all queries of $\mathcal{A}$ to his game $G_{\pi-1}$. With probability $\frac{1}{n}$, this is the session the Target query is asked to, whence the advantage of $\mathcal{C}$ is $\frac{1}{n}$ times the advantage of $\mathcal{A}$ against $G_{\mathsf{ke};\pi,n}$. Again, we omit the formal details. □

Chapter 5

Composing General Key Exchange Protocols

In the previous chapter, we have seen that key indistinguishability assures compositionality for key exchange protocols. As the latter are never used on their own, key indistinguihsability should be an important design criterion. However, in practice, explicit key confirmation is equally desired and unfortunately harms key indistinguishability. While a key-refresh step is in principle a remedy for this problem [BR93, BPR00] it is common that such protocols omit such a step—however, protocols such as TLS do not not suffer from obvious attacks. The reason for this apparent contradiction, and this is the topic of this chapter, is, that key confirmation, albeit a theoretical problem, usually do not give rise to security gaps.

The security definitions that we defined in Chapter 3 offer two important benefits. Our notion is weaker than the more established ones and thus allows the analysis of a larger class of protocols. Furthermore, and this is the topic of this chapter, they enjoy rather general composability properties.

Specifically, for protocols whose security relies exclusively on some underlying symmetric primitive we show that they can be securely composed with key exchange protocols provided that two main requirements hold: 1) no adversary can break the underlying *primitive*, even when the primitive uses keys obtained from executions of the key exchange protocol in the presence of the adversary, and 2) the security of the protocol can be reduced to that of the primitive, no matter how the keys for the primitive are distributed. Proving that the two conditions are satisfied, and then applying our generic theorem, should be simpler than performing a monolithic analysis of the composed protocol.

The second condition, a so-called *key-independent* reduction is the main ingredient in our composition theorem that we describe in detail in the first section of this chapter. Note that, throughout this chapter, we will make substantial use of the notion of split adversaries, as introduced in Section 3.2.

5.1 Key-Independent Reductions

As we weaken the condition on the protocol, we make up for it by strengthening the definition of a reduction to what we call a *key-independent reduction.* Assume that a protocol π uses in its construction some primitive ζ. A typical (black-box) reduction is some transformation R that transforms an adversary against π into one against ζ, such that if the probability that $\mathcal{A}$ breaks π is non-negligible then the probability that $R(\mathcal{A})$ breaks ζ is also non-negligible. The probabilities are over the choice of keys and coins in the system. A key-independent reduction is a strengthening of the above requirement: such a reduction is required to work no matter what the distribution over the keys is (the remaining coins are still selected uniformly at random). In particular, the reduction should work even if the adversary has arbitrary information about the keys (even the key itself!).

The example in the next Section 5.1.1 indicates that such reductions are in fact quite common in cryptography, and are simply a minor adaptation of many existing black-box style reductions. Nevertheless, for sake of completeness, we also show that the converse is not true in general: there exist black-box reductions that are not key-independent. Next we discuss our second composition theorem and clarify why key-independent reductions are a crucial component.

We now define key-independent reductions more formally. Our focus is on the case of some protocol π whose security relies solely on that of some underlying symmetric primitive ζ. We assume that the keys, which are passed as input to the protocol, are used to only key the underlying primitive. This assumption is satisfied, for instance, by standard authenticated and private channel protocols. Moreover, the case when a protocol uses several primitives (*e.g.* both encryption and authentication as for secure channels) can be cast as an instance of this setting.

Just as for standard reductions, a key-independent reduction uses an adversary $\mathcal{A}$ against protocol π to construct an adversary $R(\mathcal{A})$ against primitive ζ. Crucially, we require that the reduction works, independent of the distribution of keys that are input to the protocol. Roughly speaking, for any key distribution, if adversary $\mathcal{A}$ breaks the protocol π then adversary $R(\mathcal{A})$ breaks the primitive *for the same distribution* on the keys. This property is difficult to formalize: primitives may use different keys in their instantiations, whereas protocols may use the same key in two sessions so we have to clarify what "the same distribution" means. We do this by explicitly showing how such a reduction maps the keys used in the protocol to the keys used by the primitive.

Let $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ be a query-respecting adversary for the protocol game G_π. The adversary $\mathcal{A}$ constructed via the reduction internally maintains the list $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$, which consists of tuples of the form (U, V, kid) that record the keys shared by pairs of users. We need however to be more specific. For the particular type of reduction that we consider, we demand the existence of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against G_ζ, where the $\mathcal{A}_1$ component of $\mathcal{A}$ manages the keys in a particular way. Intuitively $\mathcal{A}_1$ is in charge of maintaining a mapping between the keys used in the game G_ζ, which $\mathcal{A}$ is playing, and the game G_π, which $\mathcal{A}$ simulates for $\mathcal{B}$. The mapping between the keys used in the

simulation and the keys used in the game of $\mathcal{A}$ may not be straightforward as the same key for a primitive may be used to simulate two or more sessions of the protocol. While we fix how $\mathcal{A}_1$ should be constructed from $\mathcal{B}_1$ we do not impose any restriction on $\mathcal{A}_2$.

Formally, we define the adversary $\mathcal{A}_1(\mathcal{B}_1)$ that works as an interface between $\mathcal{B}_1$, which expects to communicate with G_π, and the game G_ζ, as follows:

- When $\mathcal{B}_1$ writes a value k onto the input tape $G_\pi^{\mathsf{k\text{-}in}}$, algorithm $\mathcal{A}_1$ writes k on the input tape $G_\zeta^{\mathsf{k\text{-}in}}$.
- Whenever $\mathcal{B}_1$ sends a $\mathsf{NewKey}(U, V)$ query, algorithm $\mathcal{A}_1$ sends a $\mathsf{NewKey}()$ query to the game G_ζ. When the key identifier kid is returned, algorithm $\mathcal{A}_1$ stores the tuple (U, V, kid) on its list $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$. Finally $\mathcal{A}_1$ forwards kid to $\mathcal{B}_1$.
- Whenever $\mathcal{B}_1$ sends a $\mathsf{SameKey}(U, V, \mathsf{kid})$ query, algorithm $\mathcal{A}_1$ searches $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ for a tuple (V, U, kid). If there is such a tuple, then $\mathcal{A}_1$ adds the tuple (U, V, kid) to $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ and returns kid. Otherwise, $\mathcal{A}_1$ returns $\perp$.
- Whenever $\mathcal{B}_1$ issues a $\mathsf{Corrupt}_\pi(\mathsf{kid})$ query to the protocol game G_π, algorithm $\mathcal{A}_1$ sends the $\mathsf{Corrupt}_\zeta(\mathsf{kid})$ query to the primitive game. Algorithm $\mathcal{A}_1$ relays the answer it obtains to $\mathcal{B}_1$.
- Whenever $\mathcal{B}_1$ passes control to $\mathcal{B}_2$ by outputting some state st_i, algorithm $\mathcal{A}_1$ passes st_i together with the list $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ to $\mathcal{A}_2$.

We require that the algorithm $\mathcal{A}_2$ makes only black-box use of $\mathcal{B}_2$. In addition, we require that whenever $\mathcal{B}_2$ outputs some state st_i and passes control to $\mathcal{B}_1$, that $\mathcal{A}_2$ then sends st_i to $\mathcal{A}_1$, who then runs $\mathcal{B}_1$ on this state.

Note, algorithm $\mathcal{A}_2$ is allowed arbitrary queries to the primitive game G_ζ except NewKey queries. In particular, $\mathcal{A}_2$ can use the primitive oracle to answer $\mathcal{B}_2$'s queries. We stress that we do not specify how $\mathcal{A}_2$ answers $\mathcal{B}_2$'s queries, we only require that such an $\mathcal{A}_2$ exists. Also, notice that by the above description, if $\mathcal{B}$ is query-respecting for the protocol then $\mathcal{A}$ is query-respecting for the primitive.

We can now define what it means to have a key-independent reduction from a protocol to a primitive.

Definition 30 (Key-Independent Reduction)**.** *We say there is a key independent* $((G_\zeta, \delta_\zeta), (G_\pi, \delta_\pi))$*-reduction from the protocol to the primitive, if for all* query-respecting *split adversaries* $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ *against the protocol game* G_π*, there is a* query-respecting *split adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *against the primitive game* G_ζ *with* $\mathcal{A}_1$ *depending on* $\mathcal{B}_1$*, as described above, such that*

$$\Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B})(1^\eta) = 1\right] - \delta_\pi \leq \Pr\left[\mathsf{Exec}(G_\zeta, \mathcal{A})(1^\eta) = 1\right] - \delta_\zeta.$$

In other words if ζ is δ_ζ-secure then π is δ_π-secure, for whatever distribution on keys is determined by $\mathcal{B}_1$ (and $\mathcal{A}_1$ constructed as above). Notice that we do not restrict the adversary $\mathcal{B}$ in the definition to be key-benign so the reduction should work no matter what partial information the adversary has about the keys (even the keys themselves). Note that, if we quantify over the smaller class of key-benign adversaries $\mathcal{B}$, then we obtain the standard notion of cryptographic reductions. Again, if the games for the primitive and the protocol are clear from the context, we may simply say $(\delta_\zeta, \delta_\pi)$-reduction.

5.1.1 Key-Independent Reduction – Example

Assume an overly simplified setting where a sender and a receiver share a symmetric key which they use to establish an authenticated channel π. The protocol they use simply sends messages concatenated with a counter, together with a message authentication code of this concatenation produced under the shared key. A reduction would use an adversary against the authenticated channel to break the security of the MAC scheme as follows. The MACs are produced with the help of the tagging oracle specific to the MAC game. Verification by the receiver is done with the help of the corresponding verification oracle. If the receiver accepts some message that was not sent by the sender, then the message comes with a tag that was not produced by the sender (and hence the tagging oracle) and constitutes a forgery for the MAC game. Notice that the reduction is indeed independent of the distribution of the keys. If an adversary knows the key which authenticates the channel π, then it can trivially break the security of the channel, but such an adversary is still turned into one which "forges" a MAC via the above reduction.

KEY-INDEPENDENT VS. BLACK-BOX REDUCTIONS. While key-independent reductions are black-box reductions, the converse is not necessarily true. To see this, consider an arbitrary symmetric encryption scheme with a black box reduction to some underlying primitive. Now modify the scheme so that if the encryption key is the all-0 string then the encryption function is the identity. While the black-box reduction still works (the probability that the key is the all-0 string is negligible) no key-independent reduction to the underlying primitive exists.

5.2 Composition Theorem

Recall that we consider protocols π that are built on top of a symmetric primitive ζ so that the keys needed by ζ during executions of π are obtained from the key exchange protocol ke.

To show that the composition of ke with π is secure we proceed in two stages. First, we show that ke is suitable-for-ζ. As explained before, this means that an adversary who tampers with the key exchange protocol does not obtain sufficient information to break the primitive. The second required step is to exhibit a key-independent reduction from the protocol π to the primitive ζ. If these two conditions are satisfied, our theorem concludes that ke can be safely used to provide keys for π.

The following high-level overview of the proof of the theorem also sheds light on the role that key-independent reductions play in our result. Consider how would one prove security of the composition between ke and π, given that there exists a reduction R which transforms an adversary A against π into an adversary $R(A)$ against ζ (when keys are generated using some key generation algorithm), and assuming that ke can be securely composed with ζ. The only viable path is to extend R such that it takes an adversary against $\mathsf{ke}; \pi$ and produces one against $\mathsf{ke}; \zeta$, i.e. that the same reduction works even if keys are generated using ke. If after running ke keys are indinstinguishable from random ones (or if the execution of the ke can be simulated) the reduction can indeed

be extended. However if the adversary manages to obtain non-trivial information about the key (e.g., this is the case when the key is used for confirmation) a normal reduction would not work anymore. Key-independence deals precisely with this issue: R is required to work indepdent of what information the adversary obtains about the key, including through its uses in ke. This intuition sits at the core of our composition result.

An important issue is how difficult is it to prove security in the sense that we define. Proving that a key-exchange protocol is suitable for a primitive is independent of the protocol(s) where that primitive is used, can be carried out once and then reused. Also, proving suitability of a key-exchange protocol for a primitive should be easier than proving suitability for a protocol simply because the models that are involved are simpler. Furthermore, the second step should not be more difficult than a standard reduction from the protocol to the primitive for the simple case of randomly and independently distributed keys. An analysis based on our composition principle should therefore a) be simpler than a monolithic analysis of the whole system (even if only for the fact that many of the details of such an analysis are captured in the proof of our general theorem) and b) allow for reusing steps, especially for the case when the key-exchange and the protocol can each be implemented in more than one way. The latter is precisely the setting offered by practical protocols where one is offered a variety of chiper suites to select from at the beginning of the protocol.

More formally, our theorem relates the security of the composition of a key exchange with a protocol $(\mathsf{ke};\pi)$ to the security of the key exchange with a primitive $(\mathsf{ke};\zeta)$, assuming that the key exchange protocol is Match secure. The theorem says that once we have proved a key exchange protocol to be suitable for a given primitive, then this key exchange protocol can be used with any protocol whose security can be reduced (in a key-independent way) to the security of the primitive. We first give the theorem as well as several remarks, and then provide a brief overview of the proof, and a formal proof in the next Section 5.3. We finally show how our model helps to overcome a well-known problem in the security analysis of TLS.

Theorem 3. *Let ζ be a primitive, π be a protocol and $\mathsf{ke} = (\mathsf{kg}_{\mathsf{ke}}, \mathsf{P}_{\mathsf{ke}})$ be a key exchange protocol. Assume the following conditions hold.*

(1) *The key exchange protocol* ke *is* Match*-secure.*

(2) *The key exchange protocol* ke *is (G_ζ, δ_ζ)-suitable-for-ζ.*

(3) *There exists a key-independent $((G_\zeta, \delta_\zeta), (G_\pi, \delta_\pi))$-reduction from π to ζ.*

Then ke *is (G_π, δ_π)- suitable-for-π.*

Remark: Our theorem relies on the Match property in Definition 24 which, as formulated, provides strong guarantees regarding the identities of the parties that are involved. We want to emphasize that these restrictions (i.e. the Match property) can be relaxed. In fact the theorem relies on properties that Match security entail but which are strictly weaker. More specifically, Match implies that at most two sessions can have equal session identifiers (and that such sessions must have derived equal keys). These weaker guarantee is sufficient to prove the security of the composition. Technically, they guarantee that the adversary against π that we construct out of adversary against $\mathsf{ke};\pi$

is a valid adversary for the game that defines the security of π: such adversaries are allowed to set keys for the sessions of π, but at most two sessions are allowed to have equal keys. In another incarnation, our theorem could relax the Match requirement (or even completely drop it!) at the expense of strengthening the last requirement which would need to ask the existence of a key-independent reduction from a game for π where the adversary has more liberty in how he sets the keys of the sessions.

PROOF IDEA. Consider a key-benign adversary $\mathcal{C}$ playing the game $G_{\mathsf{ke};\pi}$. We transform $\mathcal{C}$ into a non key-benign adversary $\mathcal{C}^*$ still playing $G_{\mathsf{ke};\pi}$. The first part of $\mathcal{C}^*$ makes the key exchange queries, while the second part of $\mathcal{C}^*$ makes all protocol queries. The next step transforms adversary $\mathcal{C}^*$ into an adversary $\mathcal{B}$ against the protocol game G_π. To do this the first part of $\mathcal{B}$ internally simulates the key exchange game, with the keys from this simulation used as the session keys for the protocol game.

Provided the key exchange is Match-secure, then adversary $\mathcal{B}$ is query-respecting by construction, so the key-independent reduction from π to ζ yields an adversary $\mathcal{A}$ against the primitive game G_ζ. Since the construction of the first part of $\mathcal{A}$ is well-defined, we are able to remove the simulation of the key exchange within the adversary, thus providing an adversary $\mathcal{A}'$ against the composed key exchange and primitive game $G_{\mathsf{ke};\zeta}$. A final transformation turns $\mathcal{A}'$ into a key-benign adversary $\mathcal{A}^*$ against $G_{\mathsf{ke};\zeta}$. This contradicts that ke is suitable-for-ζ, and so it follows that ke is suitable-for-π.

5.3 Proof of Composition Theorem

We now prove our second composition Theorem 3.

Proof. STEP 1: CONVERSION TO NON-KEY-BENIGN ADVERSARY. Let $\mathcal{C} = (\mathcal{C}_1, \mathcal{C}_2)$ be a key-benign adversary playing the (forward-secure) game $G_{\mathsf{ke};\pi}$ of the composed protocol $(\mathsf{ke};\pi)$. Remember that $\mathcal{C}_2$ basically plays the whole composed game, while $\mathcal{C}_1$ merely generates the long-term keys used by the parties in the key exchange protocol and makes the $\mathsf{NewKey}_{\mathsf{ke}}$ queries. Algorithm $\mathcal{C}_1$ then passes a key identifier to $\mathcal{C}_2$. As before, we denote the subgames of $G_{\mathsf{ke};\pi}$ by G_{ke} and G_π. We can view the adversary $(\mathcal{C}_1, \mathcal{C}_2)$ according to Figure 5.1.

As an intermediate step we convert the adversary $\mathcal{C}$ into a specific non-key-benign adversary $\mathcal{C}^* = (\mathcal{C}_1^*, \mathcal{C}_2^*)$ which we will subsequently turn into a query-respecting adversary $\mathcal{B}$ and which will attack the protocol game instead of the composed game.

Algorithms $\mathcal{C}_1^*$ and $\mathcal{C}_2^*$ each run their local copy of $\mathcal{C} = (\mathcal{C}_1, \mathcal{C}_2)$. At first, $\mathcal{C}_1^*$ initializes $\mathcal{C}$. Now, $\mathcal{C}$ can take the following actions:

- Write a key to the input tape of the key exchange game.
- Issue a $\mathsf{NewKey}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{NewSession}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{Send}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{Corrupt}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{Reveal}_{\mathsf{ke}}$ query.
- Issue a $\mathsf{NewSession}_\pi$ query.

- Issue a $\mathsf{Corrupt}_\pi$ query.
- Issue a Name_π query.

For the first six actions, $\mathcal{C}_1^*$ just forwards the output of $\mathcal{C}$, *i.e.* $\mathcal{C}_1^*$ writes keys to the key exchange game input tape if $\mathcal{C}$ intends to do so. If $\mathcal{C}$ sends a query to the key exchange game, then so does $\mathcal{C}_1^*$ and the game's answer is forwarded to $\mathcal{C}$. If $\mathcal{C}$ sends a NewKey_π query to the protocol game, $\mathcal{C}_1^*$ forwards the query to the game and relays the game's answer to $\mathcal{C}$.

For the three latter actions, $\mathcal{C}_1^*$ sends the output of $\mathcal{C}$ together with the whole state of the Turing machine $\mathcal{C}$ to $\mathcal{C}_2^*$. Algorithm $\mathcal{C}_2^*$ then forwards $\mathcal{C}$'s query to the game and relays the response to $\mathcal{C}$, where $\mathcal{C}$ runs with the state that $\mathcal{C}_2^*$ obtained from $\mathcal{C}_1^*$. Now, symmetrically, for any of the three latter queries, $\mathcal{C}_2^*$ relays messages between the game and $\mathcal{C}$. For any of the six first actions, $\mathcal{C}_2^*$ passes control to $\mathcal{C}_1^*$ by giving the whole state of $\mathcal{C}$ as well as $\mathcal{C}$'s output.

When $\mathcal{C}$ terminates, then so does $\mathcal{C}^*$. As the input to $\mathcal{C}$ and the inputs to the game are distributed as in the previous game, the probability that the game outputs 1 remains unchanged. Adversary $\mathcal{C}^*$ is illustrated in Figure 5.1.

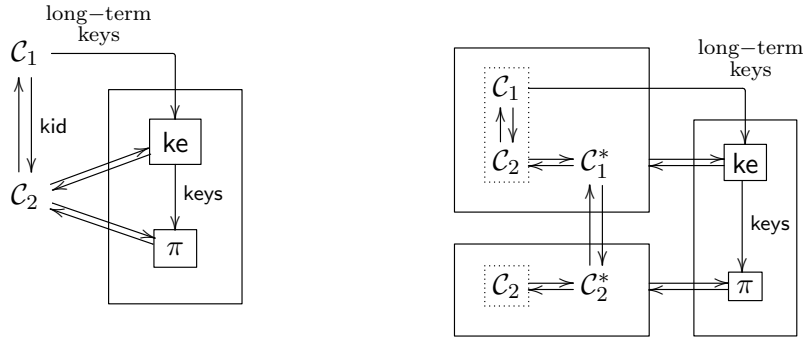

Figure 5.1: The left diagram shows the interaction between the key-benign adversary $\mathcal{C} = (\mathcal{C}_1, \mathcal{C}_2)$ and the composed game $G_{\mathsf{ke};\pi}$. The right diagram shows the transformation to the non key-benign adversary $\mathcal{C}^*$, still playing against $G_{\mathsf{ke};\pi}$.

We have that

$$\begin{aligned}\Pr\left[\mathsf{Exec}(\ G_{\mathsf{ke};\pi}), \mathcal{C} : \mathsf{kg}_{\mathsf{ke}}\right) = 1] - \delta_\pi \\ = \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1\right] - \delta_\pi.\end{aligned}$$

Step 2: Folding. We will now transform the adversary $\mathcal{C}^*$ playing $G_{\mathsf{ke};\pi}$ into a query-respecting adversary $\mathcal{B}$ playing G_π. Basically, $\mathcal{B}_2$ equals $\mathcal{C}_2^*$ with the difference that $\mathcal{B}_2$ plays directly with game G_π, while $\mathcal{C}_2^*$ expects to play with the composed game $G_{\mathsf{ke};\pi}$. Thus, whenever $\mathcal{C}_2^*$ issues a $\mathsf{NewSession}_\pi$, $\mathsf{Corrupt}_\pi$ or Name_π query, $\mathcal{B}_2$ sends the corresponding $\mathsf{NewSession}$, $\mathsf{Corrupt}$ or Name query to game G_π and relays the game's answer to $\mathcal{C}_2^*$.

We define E to be the "good" event that in the composed game $G_{\mathsf{ke};\pi}$ for all tuples $(\mathsf{label}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ there exists at most one tuple $(\mathsf{label}', \mathsf{kid}'_{\mathsf{ke}}, V', U', \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}}{}')$, and if it exists then $U = U'$ and $V = V'$. We next condition on the event E to occur, and now turn to the description of $\mathcal{B}_1$. Algorithm $\mathcal{B}_1$ internally simulates the key exchange game G_{ke} and further makes all steps that the (forward secure) composed game would have, if G_{ke} and G_π were composed together. Algorithm $\mathcal{B}_1$ internally runs $\mathcal{C}_1^*$ and passes queries from $\mathcal{C}_1^*$ to the internally simulated game G_{ke}. Whenever $\mathcal{C}_1^*$ sends a $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ query, $\mathcal{B}_1$ relays this query to G_π and passes the game's answer to $\mathcal{C}_1^*$. We now describe $\mathcal{B}$ formally: $\mathcal{B}_1$ internally models G_{ke}.

- If $\mathcal{C}_1^*$ makes any query $\mathsf{NewKey}_{\mathsf{ke}}$, $\mathsf{SameKey}_{\mathsf{ke}}$ or $\mathsf{NewSession}_{\mathsf{ke}}$ (we write $\mathsf{Name}_{\mathsf{ke}}$ for these) queries to the key exchange game then $\mathcal{B}_1$ internally simulates the key exchange game G_{ke}. The state of G_{ke} is updated within $\mathcal{B}_1$ and the response (if applicable) is returned to $\mathcal{C}_1^*$, which then updates its state as though it received the response from the real G_{ke} game.
- If $\mathcal{B}_1$ receives control and state st from $\mathcal{B}_2$, $\mathcal{B}_1$ forwards st directly to $\mathcal{C}_1^*$ and execution continues within $\mathcal{C}_1^*$.
- If $\mathcal{C}_1^*$ outputs state st, and therefore control, then $\mathcal{B}_1$ passes st to $\mathcal{B}_2$, and $\mathcal{B}_2$ passes st to $\mathcal{C}_2^*$, and execution continues within $\mathcal{C}_2^*$.
- If $\mathcal{C}_1^*$ executes $\mathsf{kg}_{\mathsf{ke}}$ and writes k to the tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$. The tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ is simulated within $\mathcal{B}_1$, so k is written to this and then used by G_{ke}, internally within $\mathcal{B}_1$.
- If $\mathcal{C}_1^*$ outputs a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ query, then $\mathcal{B}_1$ forwards this message to the internally simulated key exchange game G_{ke}. If the corresponding session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange does not accept a key, the answer from the internally simulated key exchange game is relayed to $\mathcal{C}_1^*$.
- If $\mathcal{C}_1^*$ issues a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ query such that the session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange (internally simulated within $\mathcal{B}_1$) accepts, *i.e.* when session $\mathsf{label}_{\mathsf{ke}}$ receives query $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ and changes the value of the variable $\mathrm{st}_{\mathrm{exec}}$ to $\mathsf{accepted}$ in the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ stored in list $\mathcal{L}_{G_{\mathsf{ke}}}$, then the internally simulated key exchange game writes (sid, κ) to its output tape and sends the message $\mathsf{accepted}$, which $\mathcal{B}_1$ relays to $\mathcal{C}_1^*$. Algorithm $\mathcal{B}_1$ searches the list $\mathcal{L}_{G_{\mathsf{ke}}}$ for the tuple $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$.
  - If no such tuple exists, $\mathcal{B}_1$ writes the key κ on the input tape of the protocol game G_π. Since the event E occurs it follows that the key κ has not been written to the key input tape before. Now $\mathcal{B}_1$ queries $\mathsf{NewKey}_\pi(U, V)$ to the protocol game which returns a key identifier kid_π to $\mathcal{B}_1$. Algorithm $\mathcal{B}_1$ stores the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ in list $\mathcal{L}_{\mathrm{Identifiers}}$.
  - Else, if in list $\mathcal{L}_{G_{\mathsf{ke}}}$ there exists a tuple $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$, the algorithm $\mathcal{B}_1$ searches the list $\mathcal{L}_{\mathrm{Identifiers}}$ for the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and queries $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ to the protocol game G_π which returns kid_π to $\mathcal{B}_1$. Since the event E occurs, $\mathcal{B}_1$ will only ever make one such $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ query.
- If $\mathcal{C}_1^*$ issues a $\mathsf{Reveal}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}})$, $\mathcal{B}_1$ simulates the key exchange and relays its answer to $\mathcal{C}_1^*$. Moreover, $\mathcal{B}_1$ searches the list $\mathcal{L}_{\mathrm{Identifiers}}$ for the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and queries

$\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to G_π.

- If $\mathcal{C}_1^*$ issues a $\mathsf{Corrupt}(\mathsf{kid}_{\mathsf{ke}})$ query, we need to distinguish between forward secure games and non-forward secure games. In the first game, $\mathcal{B}_1$ simply passes the query to the internally simulated game G_{ke} and returns its answer to $\mathcal{C}_1^*$. In the latter case, $\mathcal{B}_1$ does the following for all tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, *, *, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ in $\mathcal{L}_{G_{\mathsf{ke}}}$: Search $\mathcal{L}_{\mathrm{Identifiers}}$ for all tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to G_π and return G_π's answer to C_1^*.
- If $\mathcal{C}_2^*$ issues any query Name_π to the subgame G_π of the composed game $G_{\mathsf{ke};\pi}$ $\mathcal{B}_2$ passes the query Name_π to G_π and returns the game's answer to $\mathcal{C}_2^*$.

We can view the interaction of $\mathcal{B}$ with the game G_π in Figure 5.2.

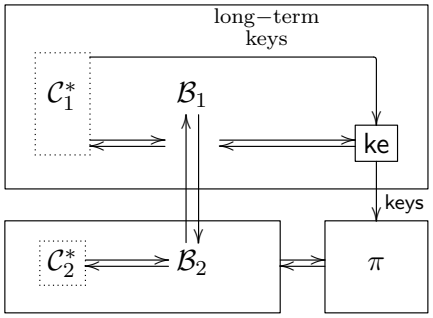

Figure 5.2: Adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ playing game G_π. Algorithm $\mathcal{B}_1$ runs using $\mathcal{C}_1^*$ and simulates the key exchange internally. Keys output by the simulated key exchange are written to the key input tape of G_π.

If the algorithm $\mathcal{B}_1$ needed to make more than one query $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ for some triple (V, U, kid), this would correspond to a tuple $(\mathsf{label}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ accepting a key, which had already been accepted by tuple $(\mathsf{label}', \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}}{}')$; *i.e.* the event E did not occur. Therefore $\mathcal{B}_1$ will only ever make one such $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ query. The same argument applies to $\mathcal{B}_1$ never making a $\mathsf{SameKey}_\pi(U, V, \mathsf{kid})$, when it has made a query, $\mathsf{NewKey}_\pi(U, V)$, which responded with kid. Therefore adversary $\mathcal{B}$ will be query-respecting.

If we now assume that the event E does not occur, then in the composed game, some session has accepted a key which violates on of the properties of the Match-security property of the key exchange. Given the composed adversary $\mathcal{C}^*$ that caused such an event, one can trivially construct an adversary $\mathcal{D}$ against the $G_{\mathsf{ke}}^{\mathsf{Match}}$ game.

This leads us to the following:

$$\begin{aligned}\Pr\left[\mathsf{Exec}(\ G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1 | E\right] - \delta_\pi \\ = \Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B})(1^\eta) = 1\right] - \delta_\pi,\end{aligned}$$

and

$$\begin{aligned}\Pr\left[\mathsf{Exec}(\ G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1 | \neg E\right] \\ = \Pr\left[\mathsf{Exec}(G^{\mathsf{Match}}_{\mathsf{ke}}, \mathcal{D} : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1\right] \leq \epsilon(1^\eta),\end{aligned}$$

and we can see that

$$\begin{aligned}\Pr[\ \mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1] \\ &= \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1 \cap E\right] \\ &\quad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1 \cap \neg E\right] \\ &= \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1 | E\right] \cdot \Pr\left[E\right] \\ &\quad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1 | \neg E\right] \cdot \Pr\left[\neg E\right] \\ &< \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1 | E\right] \\ &\quad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1 | \neg E\right] \\ &= \Pr\left[\mathsf{Exec}(G_{\pi}, \mathcal{B})(1^\eta) = 1\right] \\ &\quad + \Pr\left[\mathsf{Exec}(G^{\mathsf{Match}}_{\mathsf{ke}}, \mathcal{D} : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1\right].\end{aligned}$$

Step 3: Reducing to the primitive. Currently we have an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ playing G_π, where $\mathcal{B}_1$ writes keys to the input tape $G^{\mathsf{k\text{-}in}}_\pi$. By construction the algorithm $\mathcal{B}_1$ makes only NewKey_π, $\mathsf{SameKey}_\pi$ and $\mathsf{Corrupt}_\pi$ queries to G_π, whilst $\mathcal{B}_2$ makes all other queries to G_π as well as $\mathsf{Corrupt}_\pi$ queries. It follows that $\mathcal{B}$ is a query-respecting adversary according to Section 5.1.

Hence, by assumption, we are given a key-independent $(\delta_\zeta, \delta_\pi)$-reduction from the protocol to the primitive. Therefore there exists an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ playing G_ζ, where $\mathcal{A}_1$ is constructed from $\mathcal{B}_1$ as defined in Section 5.1. We now have that

$$\Pr\left[\mathsf{Exec}(G_\pi, \mathcal{B}^*)(1^\eta) = 1\right] - \delta_\pi \leq \Pr\left[\mathsf{Exec}(G_\zeta, \mathcal{A})(1^\eta) = 1\right] - \delta_\zeta.$$

Step 4: Unfolding. We now show how to unfold the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, so that the key exchange is no longer simulated within the adversary. Currently, we have that $\mathcal{A}_1$ is constructed using $\mathcal{B}_1$, and in turn, $\mathcal{B}_1$ is constructed using $\mathcal{C}^*_1$. We now look at how to construct $\mathcal{A}'_1$ for the adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}_2)$ playing the composed game $G_{\mathsf{ke};\zeta}$, where $\mathcal{A}'_1$ is constructed using $\mathcal{C}^*_1$ directly, thus eliminating $\mathcal{B}_1$. This construction is illustrated within Figure 5.3.

Let us now examine how $\mathcal{A}_1$ executes, and we construct $\mathcal{A}'_1$, so that an execution of $\mathcal{A}$ and $\mathcal{A}'$ will be identical. Let $\mathcal{L}^{\mathsf{keys}}_{A'}$ be an initially empty list of triples of the form (U, V, kid_π).

- If $\mathcal{C}^*_1$ makes any query $\mathsf{NewKey}_{\mathsf{ke}}$ or $\mathsf{NewSession}_{\mathsf{ke}}$ (we write $\mathsf{Name}_{\mathsf{ke}}$ for those) queries to the key exchange:
  - $\mathcal{A}_1$: $\mathcal{B}_1$ receives $\mathsf{Name}_{\mathsf{ke}}$ and internally simulates the key exchange game G_{ke}. The state of G_{ke} is updated within $\mathcal{B}_1$ and the response (if applicable) is returned to $\mathcal{C}^*_1$, which then updates its state as though it received the response from the real G_{ke} game.

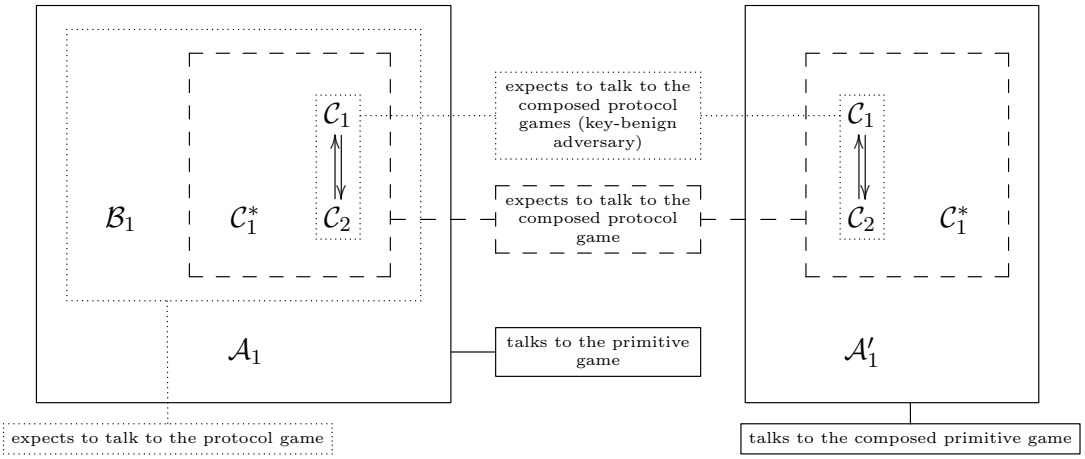

Figure 5.3: Construction showing what each adversary interacts with and which adversaries they run as a subroutine. We note that adversary $\mathcal{B}_1$ of the left picture internally simulates the key exchange, while on the right picture this key exchange has been unfolded from the adversary so that $\mathcal{A}_1'$ interacts with the composed primitive game.

  - $\mathcal{A}_1'$: $\mathcal{A}_1'$ receives $\mathsf{Name}_{\mathsf{ke}}$ and forwards this to the real key exchange game G_{ke}. The response of G_{ke} is received by $\mathcal{A}_1'$ and forwarded directly to $\mathcal{C}_1^*$, which updates its state exactly as it does within $\mathcal{A}_1$.
- If $\mathcal{A}_1$ (or $\mathcal{A}_1'$) receives control and state st from $\mathcal{A}_2$:
  - $\mathcal{A}_1$: The state st is passed directly from $\mathcal{A}_1$ to $\mathcal{B}_1$. In turn $\mathcal{B}_1$ forwards st directly to $\mathcal{C}_1^*$ and execution continues within $\mathcal{C}_1^*$.
  - $\mathcal{A}_1'$: The state st is passed directly to $\mathcal{C}_1^*$ and execution continues within $\mathcal{C}_1^*$.
- If $\mathcal{C}_1^*$ outputs state and therefore control:
  - $\mathcal{A}_1$: Control and state is passed from $\mathcal{C}_1^*$ to $\mathcal{B}_1$. Now $\mathcal{B}_1$ forwards this state directly to $\mathcal{A}_1$, who in turn passes control and state on to $\mathcal{A}_2$ and execution continues within $\mathcal{A}_2$.
  - $\mathcal{A}_1'$: Control and state is passed from $\mathcal{C}_1^*$ to $\mathcal{A}_1'$. Now $\mathcal{A}_1'$ passes control and state to $\mathcal{A}_2$ and execution continues within $\mathcal{A}_2$.
- If $\mathcal{C}_1^*$ executes $\mathsf{kg}_{\mathsf{ke}}$ and writes k to the tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$:
  - $\mathcal{A}_1$: The tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ is simulated within $\mathcal{B}_1$, so k is written to this and then used by G_{ke}, internally within $\mathcal{B}_1$.
  - $\mathcal{A}_1'$: $\mathcal{A}_1'$ takes k and writes this onto the tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}in}}$ for the real game G_{ke}. Essentially this is a copy operation for $\mathcal{A}_1'$, and we notice that $\mathcal{A}_1'$ does not store any information about k.
- If $\mathcal{C}_1^*$ outputs a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ query, then $\mathcal{A}_1$ forwards this message to the in-

ternally simulated key exchange game G_{ke}, and $\mathcal{A}_1^*$ forwards this message to the key exchange game G_{ke} being a subgame of the composed primitive game $G_{\mathsf{ke};\zeta}$. If the corresponding session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange does not turn its state into accepted, answers from the key exchange game are relayed to $\mathcal{C}_1^*$ (either through D_1 in the case of A_1, or directly in the case of $\mathcal{A}_1'$).

- If $\mathcal{C}_1^*$ outputs a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ query such that the corresponding session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange turns its state into $\mathsf{accepted}$, $\mathcal{B}_1$ simulates internally, that the game G_{ke} writes to its output tape $G_{\mathsf{ke}}^{\mathsf{k\text{-}out}}$, then $\mathcal{B}_1$ undertakes certain actions that were modified by transforming $\mathcal{B}_1$ into $\mathcal{A}_1$. We need to show that they are now identical to what the composed game of key exchange and primitive would have done. This can be verified by examining the two columns in Figure 5.4
- $\mathcal{C}_1^*$ issues a $\mathsf{Reveal}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}})$ query.
  - $\mathcal{A}_1$: $\mathcal{B}_1^*$ receives this query and passes it to the internally simulated key exchange game G_{ke}. $\mathcal{B}_1^*$ relays the game's answer to $\mathcal{C}_1^*$. Furthermore, $\mathcal{B}_1^*$ performs a lookup on the list $\mathcal{L}_{\text{Identifiers}}$ to find an entry $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$. If such an entry exists, it intends to send the query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to the protocol game G_π. $\mathcal{A}_1$ passes the query $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\pi)$ to the primitive game which returns k'. It follows that $\kappa = k'$, and $\mathcal{A}_1$ returns κ to the adversary to $\mathcal{B}_1^*$ that relays κ to $\mathcal{C}_1^*$.
  - $\mathcal{A}_1'$ receives this query and passes it to the key exchange game. $\mathcal{A}_1'$ relays the game's answer to $\mathcal{C}_1^*$. The composed game performs a lookup on the list $\mathcal{L}_{\text{Identifiers}}$ to find and entry $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$. If such a tuple exists, it sends the query $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\zeta)$ to the G_ζ subgame which returns k'. It follows that $\kappa = k'$ and $G_{\mathsf{ke};\zeta}$ returns κ to the $\mathcal{A}_1'$ that relays it to $\mathcal{C}_1^*$.
- $\mathcal{C}_1^*$ issues a $\mathsf{Corrupt}_{\mathsf{ke}}$ query. We first describe the step for the non-forward secure proof.
  - $\mathcal{A}_1$: The query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid}_{\mathsf{ke}})$ is received by $\mathcal{B}_1$ and passed to the internally simulated game G_{ke} which returns a key k. This key is passed back to $\mathcal{C}_1^*$. Furthermore, for all sessions $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, *, *, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ in $\mathcal{L}_{G_{\mathsf{ke}}}$, $\mathcal{B}_1$ searches the list $\mathcal{L}_{\text{Identifiers}}$ for tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\pi)$ and intends to send the query $\mathsf{Corrupt}_\pi(\mathsf{kid}_\pi)$ to the protocol game G_π. $\mathcal{A}_1$ then sends $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\pi)$ to the primitive game G_ζ. The answer of the primitive game is received by $\mathcal{A}_1$ who returns it to $\mathcal{B}_1$ that relays it to $\mathcal{C}_1^*$.
  - $\mathcal{A}_1'$ receives the query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid}_{\mathsf{ke}})$ and passes it to the composed game that relays it to the subgame G_{ke} which returns a key k. $\mathcal{A}_1'$ passes this key to the $\mathcal{C}_1^*$. Furthermore, for all sessions $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, *, *, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$ in $\mathcal{L}_{G_{\mathsf{ke}}}$, the composed game searches the list $\mathcal{L}_{\text{Identifiers}}$ for tuples $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$ and sends the query $\mathsf{Corrupt}_\zeta(\mathsf{kid}_\zeta)$ to the primitive game G_ζ. The answer of the subgame is relayed to the $\mathcal{A}_1'$ that passes it to $\mathcal{C}_1^*$.
- $\mathcal{C}_1^*$ issues a $\mathsf{Corrupt}_{\mathsf{ke}}$ query. We now detail this step for the forward secure proof.
  - $\mathcal{A}_1$: The query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid})$ is received by $\mathcal{B}_1$ and passed to the internally simulated game G_{ke} which returns a key k. This key is passed back to $\mathcal{C}_1^*$.
  - $\mathcal{A}_1'$ receives the query $\mathsf{Corrupt}_{\mathsf{ke}}(\mathsf{kid})$ and passes it to the composed game that relays

it to the subgame G_{ke} which returns a key k. $\mathcal{A}'_1$ passes this key to the $\mathcal{C}^*_1$.

It follows from the above descriptions that if the random bits used by the internally simulated game G_{ke} and the key exchange subgame G_{ke} of the composed game $(G_{\mathsf{ke}}, G_\zeta)$ are the same and the randomness used by the adversaries $\mathcal{A}$ and $\mathcal{A}'$ in particular steps is also equal, and if the random bits used by the game G_ζ and by the subgame G_ζ of the composed game $(G_{\mathsf{ke}}, G_\zeta)$ are equal then an execution of $\mathcal{A}$ or $\mathcal{A}'$ will result in the same keys being written to the $G_\zeta^{\mathsf{k\text{-}in}}$ tape of G_ζ in either case so that $\mathcal{C}^*_1$ in both cases receives and returns the same state and queries (if also $\mathcal{C}^*_1$ is used with the same random bits in both cases. Therefore we have

$$\Pr\left[\mathsf{Exec}(G_\zeta, \mathcal{A})(1^\eta) = 1\right] = \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}')(1^\eta) = 1\right].$$

Diagrammatically we see $\mathcal{A}$ interacting with $G_{\mathsf{ke};\zeta}$ in Figure 5.5.

Step 5: Conversion to key-benign adversary. The final step requires us to convert $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}_2)$ playing $G_{\mathsf{ke};\zeta}$ into a key-benign adversary $\mathcal{A}^* = (\mathcal{A}^*_1, \mathcal{A}^*_2)$ playing the composed game $G_{\mathsf{ke};\zeta}$. Remember that this means that almost all functionality of $\mathcal{A}'$ is moved into $\mathcal{A}^*_2$, while $\mathcal{A}^*_1$ will be bound to run the key generation algorithm of the key exchange, to write long-term keys to the input tape of the key exchange game and to pass the key identifiers to $\mathcal{A}^*_2$. Algorithm $\mathcal{A}^*_2$ decides when such an action shall take place.

Presently we have $\mathcal{A}'_1$ constructed using $\mathcal{C}^*_1$. Moreover $\mathcal{C}^*$ was constructed based upon $\mathcal{C}$, and therefore $\mathcal{A}'_1$ was constructed from $\mathcal{C}$; where $\mathcal{C}$ is the key-benign adversary against $G_{\mathsf{ke};\pi}$.

Pictorically, our goal is illustrated in Figure 5.6. On the left side, you see the current situation while on the right side, you see the goal of the transformation we are going to undertake.

We now examine the interaction between $\mathcal{C}_1$ and $\mathcal{C}_2$. Adversary $\mathcal{C}$ is a key-benign adversary. Hence, setting $\mathcal{A}^*_1 := \mathcal{C}_1$, the first part of $\mathcal{A}^*$ is already well-formed. We now need to define $\mathcal{A}^*_2$ in such a way that the only messages sent by $\mathcal{A}^*_2$ to $\mathcal{A}^*_1$ are of the form $\mathsf{NewKey}(U, V)$ or $\mathsf{SameKey}(U, V, \mathsf{kid})$.

We define $\mathcal{A}^*_2$ as follows: $\mathcal{A}^*_2$ equals $\mathcal{A}'$ except that whenever within $\mathcal{A}'$, $\mathcal{C}_2$ sends a message to $\mathcal{C}_1$, this message is not sent to the internal copy of $\mathcal{C}_1$ but instead, $\mathcal{A}^*_2$ relays it to $\mathcal{A}^*_1 = \mathcal{C}_1$ which acts as described. Its answer is returned to $\mathcal{C}_2$.

Now, $\mathcal{A}^*$ is a key-benign adversary because the communication between $\mathcal{A}^*_1$ and $\mathcal{A}^*_2$ equals the communication of $\mathcal{C}_1$ and $\mathcal{C}_2$. Furthermore, the inputs provided by the adversary $\mathcal{A}^*$ to the game $(G_{\mathsf{ke}}, G_\zeta)$ are identical to those provided by $\mathcal{A}'$. Thus, the success probability does not change.

Therefore $\mathcal{A}^*$ is a key-benign adversary such that

$$\begin{aligned}\Pr\left[\mathsf{Exec}(\ G_{\mathsf{ke};\zeta}, \mathcal{A}')(1^\eta) = 1\right] \\ = \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}^* : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1\right].\end{aligned}$$

Finally, given that the composition $(\mathsf{ke}; \zeta)$ is (forward-secret) δ_ζ-secure we have that

$$\Pr[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}^* : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1] - \delta_\zeta \leq \epsilon(1^\eta).$$

The above leads to

$$\begin{aligned}
\Pr[\ & \mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C} : \mathsf{kg}_{\mathsf{ke}}) = 1] - \delta_\pi \\
&= \Pr[\mathsf{Exec}(G_{\mathsf{ke};\pi}, \mathcal{C}^*)(1^\eta) = 1] - \delta_\pi \\
&< \Pr[\mathsf{Exec}(G_\pi, \mathcal{B})(1^\eta) = 1] - \delta_\pi \\
&\qquad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}}, \mathcal{D} : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1\right] \\
&\leq \Pr[\mathsf{Exec}(G_\zeta, \mathcal{A})(1^\eta) = 1] - \delta_\zeta \\
&\qquad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}}, \mathcal{D} : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1\right] \\
&= \Pr\left[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}')(1^\eta) = 1\right] - \delta_\zeta \\
&\qquad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}}, \mathcal{D} : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1\right] \\
&= \Pr[\mathsf{Exec}(G_{\mathsf{ke};\zeta}, \mathcal{A}^* : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1] - \delta_\zeta \\
&\qquad + \Pr\left[\mathsf{Exec}(G_{\mathsf{ke}}^{\mathsf{Match}}, \mathcal{D} : \mathsf{kg}_{\mathsf{ke}})(1^\eta) = 1\right] \\
&\leq \epsilon(1^\eta),
\end{aligned}$$

and hence the composition $(\mathsf{ke}; \pi)$ is (forward-secret) δ_π-secure. □

$\mathcal{A}_1$: If $\mathcal{C}_1^*$ issues a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ query such that the session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange (internally simulated within $\mathcal{B}_1$) accepts, *i.e.* when session $\mathsf{label}_{\mathsf{ke}}$ receives query $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ and changes the value $\mathrm{st}_{\mathrm{exec}}$ to $\mathsf{accepted}$ in the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ stored in list $\mathcal{L}_{G_{\mathsf{ke}}}$, then the key exchange game writes (sid, κ) to its output tape and sends the message $\mathsf{accepted}$ which $\mathcal{B}_1$ relays to $\mathcal{C}_1^*$. $\mathcal{A}_1$ searches the list $\mathcal{L}_{G_{\mathsf{ke}}}$ for tuples $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$. If no such tuple exists, $\mathcal{B}_1$ intends to write the key κ on the input tape of the protocol game G_π - and $\mathcal{A}_1$ writes κ to the input tape of primitive game G_ζ. $\mathcal{B}_1$ intends to query $\mathsf{NewKey}_\pi(U, V)$ to the protocol game - and $\mathcal{A}_1$ queries $\mathsf{NewKey}_\zeta()$ to the primitive game which returns a key identifier kid_ζ that $\mathcal{A}_1$ passes as the expected kid_π to $\mathcal{B}_1$. $\mathcal{B}_1$ stores the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$ in list $\mathcal{L}_{\mathrm{Identifiers}}$ and passes kid_ζ to $\mathcal{C}_1^*$.
Else, if in list $\mathcal{L}_{G_{\mathsf{ke}}}$ there exists a tuple $(\mathsf{label}', \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$, $\mathcal{B}_1$ searches the list $\mathcal{L}_{\mathrm{Identifiers}}$ for a triple $(\mathsf{label}, \mathsf{sid}, \mathsf{kid}_\pi)$ and intends to issue the query $\mathsf{SameKey}_\pi(V, U, \mathsf{kid}_\pi)$ to the protocol game G_π. $\mathcal{A}_1$ then searches $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ for a tuple (V, U, kid). If there is such a tuple, $\mathcal{A}_1$ adds the tuple (U, V, kid) to $\mathcal{L}_{\mathcal{A}}^{\mathsf{keys}}$ and returns kid to $\mathcal{B}_1$ which passes it to $\mathcal{C}_1^*$.

$\mathcal{A}'_1$: If $\mathcal{C}_1^*$ issues a $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ query that is relayed by $\mathcal{A}'_1$ to the composed game $G_{\mathsf{ke};\zeta}$ and makes the session $\mathsf{label}_{\mathsf{ke}}$ of the key exchange accept, *i.e.* session $\mathsf{label}_{\mathsf{ke}}$ receives query $\mathsf{Send}_{\mathsf{ke}}(\mathsf{label}_{\mathsf{ke}}, \mathsf{msg})$ query and changes the value $\mathrm{st}_{\mathrm{exec}}$ to $\mathsf{accepted}$ in the tuple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, \kappa, \mathrm{st}_{\mathrm{key}})$ stored in list $\mathcal{L}_{G_{\mathsf{ke}}}$, then the key exchange game writes (sid, κ) to its output tape and sends the message $\mathsf{accepted}$ which $\mathcal{A}'_1$ relays to $\mathcal{C}_1^*$. The composed game searches the list $\mathcal{L}_{G_{\mathsf{ke}}}$ for tuples $(\mathsf{label}'_{\mathsf{ke}}, \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$. If no such tuple exists, the composed game writes the key κ on the input tape of the primitive game G_ζ. The composed game then sends query $\mathsf{NewKey}_\zeta()$ to the primitive game which returns a key identifier kid_ζ that $\mathcal{A}_1$ passes as the expected kid_π to $\mathcal{C}_1^*$. The game stores the triple $(\mathsf{label}_{\mathsf{ke}}, \mathsf{sid}, \mathsf{kid}_\zeta)$ in list $\mathcal{L}_{\mathrm{Identifiers}}$.
Else, if in list $\mathcal{L}_{G_{\mathsf{ke}}}$ there exists a tuple $(\mathsf{label}', \mathsf{kid}_{\mathsf{ke}}, V, U, \mathsf{sid}, \mathsf{accepted}, \kappa, \mathrm{st}_{\mathrm{key}})$, the composed game does not write key κ to the input tape of the primitive game. $\mathcal{A}'_1$ then searches $\mathcal{L}_{\mathcal{A}'}^{\mathsf{keys}}$ for a tuple (V, U, kid). If there is such a tuple, $\mathcal{A}'_1$ adds the tuple (U, V, kid) to $\mathcal{L}_{\mathcal{A}'}^{\mathsf{keys}}$ and returns kid to $\mathcal{C}_1^*$.

Figure 5.4: Transformation from $\mathcal{A}_1$ into $\mathcal{A}'_1$

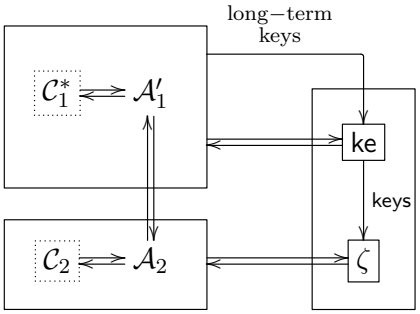

Figure 5.5: The construction of adversary $\mathcal{A}'$ playing the $G_{\mathsf{ke};\zeta}$ game.

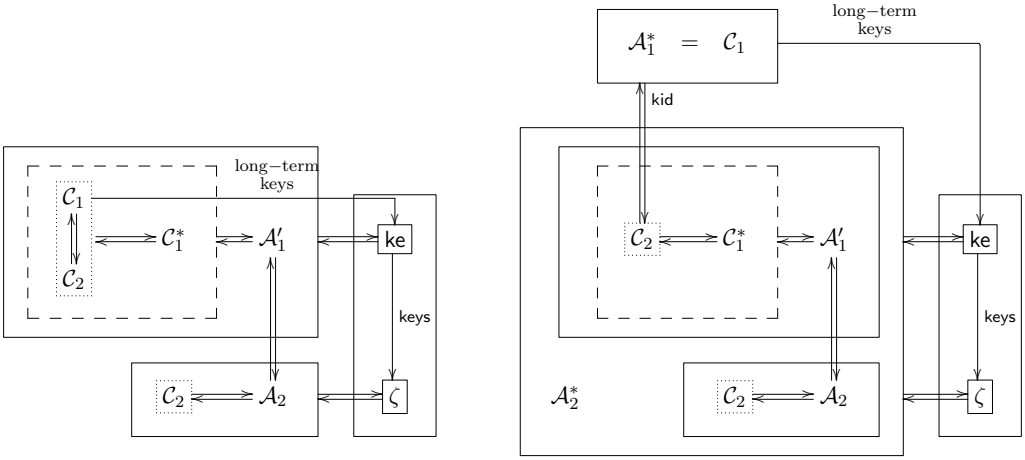

Figure 5.6: The left side shows the construction of $\mathcal{A}'$ interacting with the composed game $G_{\mathsf{ke};\zeta}$. The right side shows the construction of the key-benign adversary $\mathcal{A}^*$ playing the $G_{\mathsf{ke};\zeta}$ composed game. Notice that only key identifiers are passed from $\mathcal{A}_1^*$ to $\mathcal{A}_2^*$, and excluding the NewKey queries, $\mathcal{A}_2^*$ makes all queries to the composed game $G_{\mathsf{ke};\zeta}$.

Chapter 6

On the Security of TLS

We now sketch how we use the theorem above to prove that the composition of the TLS handshake protocol (which implements a key-exchange) with one particular instantiation of the TLS record layer protocol (which implements a secure channel).

The record layer protocol implements a secure channel with multiple security guarantees among which we are mainly concerned with privacy of messages, length hiding, and authentication of messages. The implementation essentially encrypts the payload together with a sequence number via a Length Hiding Authenticated Encryption (LHAE) scheme. TLS offers multiple choices for the implementation of each of the two components.

In TLS, the last message of the handshake protocol, i.e. the `FINISHED` message acting as a key confirmation, is already sent over the record layer and hence the handshake actually uses the keys later employed by the record layer. In practice, this does not create a problem with message authentication (*e.g.* the `FINISHED` message cannot be replayed) due to the use of appropriately initialized sequence numbers. The sequence number is encrypted together with the payload to prevent replay attack and out-of-order delivery, and to allow the receiver to distinguish protocol messages from the `FINISHED` message. So, although the derived keys are not indistinguishable from random ones in the end of the TLS handshake, it appears that they can be safely used for the record layer.

TLS falls within the setting where our composition theorem applies and its security follows from three steps: a) the handshake satisfies the Match property, b) the key exchange is suitable for a (variant of) LHAE encryption scheme, and c) the security of the record layer reduces to (that variant of) the encryption scheme via a key-independent reduction.

The benefits of this modular approach should be clear. To analyze TLS for a different instantiation of the key exchange one needs to show that the variant of key exchange is good for the LHAE scheme (this step is inevitable no matter what approach one takes) and that the record layer indeed employs an LHAE scheme. Thus, our approach allows reusing step c) across different implementations (there is no need to repeat this step for a different implementation of the handshake part). In contrast a monolithic

analysis would have to repeat the reduction argument for each possible instantiation (key-exchange, record layer). Of course, one can hand-wavingly appeal to the (inevitable) similarities between the corresponding proofs, but our rigorous approach is obviously cleaner. Finally, for the record layer protocol one can concentrate the analysis on the underlying encryption scheme and ignore the difficulties associated with statefulness, sequence numbers, etc which are delt with once and for all. We do precisely that when we rely on the result of [PRS11] which proves that one particular implementation for the record protocol is LHAE.

6.1 Protocol description

In the TLS handshake protocol, the parties agree on application keys for the secure channel protocol, called record layer. Firstly, they derive a so-called pre-master secret via a key transport (KT) protocol or via a Diffie-Hellman key exchange. A transformation of the pre-master secret then yields the so-called master secret, which, in turn, is used to compute the application keys. Note that each party holds two application keys: one is used for sending and one for receiving messages. Finally, the parties engage in a key confirmation step, the `FINISHED` messages. As explained earlier the use of the application keys in the `FINISHED` messages violates key indistinguishability and renders an analysis in the BR-model impossible.

The protocol description in Figure 6.1 provides an overview on the TLS handshake protocol. Depending on whether one opts for computing the pre-master secret via Diffie-Hellman (DH) or via key transport (KT), one either skips all steps with label "(KT)" or all steps labeled "(DH)". The Diffie-Hellman key exchange yields a forward-secure protocol, while the key transport protocol only provides a non-forward secure key exchange. We refrain from allowing parties and/or adversaries to decide on the fly the pre-master secret computation mode, as this involves a rather complicated mixed models overhead. Strictly speaking, our analysis only holds for concurrently running protocol executions that always derive the pre-master secret in the same way and those executions in which client authentication is performed.

Note that we abstract out the concrete header information for encryption resp. authentication, as well as the type of cipher used etc. Our result applies to all implementations of the record layer, current ones or even future ones for which one can show LHAE security. In such a case, the protocol specifies the header, but the encryption primitive even remains secure when queried on non-well-formed headers (it might reject those headers, though). Note that Paterson et al. [PRS11] proved the LHAE security of the record layer encryption scheme according to the current TLS standard when the cipher is used in CBC mode.

The use of the term "header" is sometimes ambiguous. In particular, we have to distinguish between the string H, which is an input to the encryption scheme with authenticated data, and the header for packages in the TLS protocol. In particular, $H = n|H_1|H_2$, where n is a locally maintained sequence number (i.e., which is not transmitted but each party keeps track itself), H_1 is further locally maintained header

information, and H_2 is the part of the header that appears in the beginning of a TLS record protocol package. The only property of the header that we use is that the position of the sequence number in H_1 is uniquely defined, we denote this by $n|H_1$.

	client (Alice)		server (Bob)
1.	$r_c \leftarrow \{0,1\}^t$	$\xrightarrow{r_c \| ID_c}$	
2.		$\xleftarrow{r_s \| ID_s}$	$r_s \leftarrow \{0,1\}^t$
2a.	$\mathsf{Ver}(pk_{\mathrm{CA}}, \mathsf{Cert}_{\mathrm{CA}}(pk_s \| ID_s)) = 1$?	$\xleftarrow{\mathsf{Cert}_{\mathrm{CA}}(pk_s \| ID_s)}$	
2b.		$\xleftarrow{\text{“CertificateRequest”}}$	
		$\xleftarrow{\text{“ServerHelloDone”}}$	
2c.			$(p,q,g) \leftarrow \mathsf{Params}(1^t)$
			$b \leftarrow \mathbb{Z}_q$, $Y_s := g^b$
	$h_1 \leftarrow RO(r_c \| r_s \| p \| g \| Y_s)$, $\mathsf{Ver}(pk_s; h_1) = 1$?	$\xleftarrow{(p,q,g,Y_s,\sigma_s)}$	$h_1 \leftarrow RO(r_c \| r_s \| p \| g \| Y_s)$, $\sigma_s \leftarrow \mathsf{Sig}(sk_s, h_1)$
3.		$\xrightarrow{\mathsf{Cert}_{\mathrm{CA}}(pk_c \| ID_c)}$	$\mathsf{Ver}(pk_{\mathrm{CA}}, \mathsf{Cert}_{\mathrm{CA}}(pk_c \| ID_c)) = 1$?
4a.	$a \leftarrow \mathbb{Z}_q$, $Y_c := g^a$	$\xrightarrow{Y_c}$	
4b.	$\kappa \leftarrow \{0,1\}^t$, $C \leftarrow \mathsf{Enc}_{pk_s}(\kappa)$	$\xrightarrow{C}$	
5.	premaster		premaster
6.	master		master
7.	(κ_c, κ_s)		(κ_c, κ_s)
8.	$h_2 :=$ RO(Transcript), $\sigma_c \leftarrow \mathsf{Sig}(sk_c; h_2)$	$\xrightarrow{\sigma_c}$	$h_2 :=$ RO(Transcript), $\mathsf{Ver}(pk_c; h_2, \sigma_c) = 1$?
9.		$\xrightarrow{\text{“ChangeCipherSpec”}}$	
	$h_3 :=$ RO(Transcript), $\tau_1 \leftarrow \mathsf{Mac}_{\mathsf{master}}(h_3)$,		$h_3 :=$ RO(Transcript)
	$F_c \leftarrow \mathsf{LHEnc}(\kappa_c; \ell, 0 \| H_1^c \| H_2^c, \tau_1)$	$\xrightarrow{H_2^c, F_c}$	$\tau_1 \leftarrow \mathsf{LHDec}(\kappa_c; F_c)$, $\mathsf{Ver}(\mathsf{master}; h_3, \tau_1) = 1$?
		$\xleftarrow{\text{“ChangeCipherSpec”}}$	
	$h_4 :=$ RO(Transcript)		$h_4 :=$ RO(Transcript), $\tau_2 \leftarrow \mathsf{Mac}(\mathsf{master}; h_4)$,
	$\tau_2 \leftarrow \mathsf{LHDec}(\kappa_s; F_s)$, $\mathsf{Ver}(\mathsf{master}; h_4, \tau_2) = 1$?	$\xleftarrow{H_2^s, F_s}$	$F_s \leftarrow \mathsf{LHEnc}(\kappa_s; \ell, 0 \| H_1^s \| H_2^s, \tau_2)$

Figure 6.1: TLS protocol with pre-master computation being either DH or KT. The numbered stages refer to 1. Client Hello, 2. Server Hello, 2a. Certificate Transfer, 2b. Certificate request, 2c. Server Key Exchange (DH), 3. Certificate Transfer, 4a. Client Key Exchange (DH), 4b. Client Key Exchange (KT), 5. Derivation of pre-master secret, 6. Derivation of master secret, 7. Derivation of application keys, 8. Client Authentication, 9. Finished Messages

The participants derive the pre-master secret, the master secret and the application keys as follows: in the key transport case, the client chooses the pre-master secret premaster and sends an encryption of it to the server. In the Diffie-Hellman case, the parties run an ephemeral Diffie-Hellman key agreement to establish the pre-master secret premaster. In both cases, both derive the master secret via a key derivation function RO (which we model as a random oracle below, hence the name) as

$$\mathsf{master} := \mathrm{RO}(\mathsf{premaster}, \text{“master secret”}, r_c, r_s).$$

Afterwards, they query the random oracle as follows:

$$\mathsf{key\ block} := \mathrm{RO}(\mathsf{master}, \text{“key expansion”}, r_s, r_c)$$

They cut key block into four pieces key block$_1$,...,key block$_4$ of equal length and query the random oracle to compute the following keys μ_c and μ_s for message authentication and ϵ_c, ϵ_s for encryption:
- $\mu_c := \mathrm{RO}(\text{key block}_1, \text{“client-write-MAC-secret”}, r_c, r_s)$,
- $\mu_s := \mathrm{RO}(\text{key block}_2, \text{“server-write-MAC-secret”}, r_c, r_s)$,
- $\epsilon_c := \mathrm{RO}(\text{key block}_3, \text{“client-write-key”}, r_c, r_s)$,
- $\epsilon_s := \mathrm{RO}(\text{key block}_4, \text{“server-write-key”}, r_c, r_s)$,

They set $\kappa_s := (\epsilon_s, \mu_s)$ and $\kappa_c := (\epsilon_c, \mu_c)$ and return (κ_c, κ_s) as the application keys. The keys output by the key exchange protocol then consist of (κ_c, κ_s). As session identifiers, we define the pair $(r_c||ID_c, r_s||ID_s, \text{premaster})$ and show that with these, the TLS protocol satisfies Match security.

6.2 Match security

In this section, we show the first of the three cornerstones in our analysis of the TLS protocol.

Theorem 4 (Match Security of TLS)**.** *The TLS Handshake protocol satisfies* Match *security.*

Proof. If both parties derive the same $\text{sid} = (r_c||ID_c, r_s||ID_s, \text{premaster})$, then the first condition of Match security is trivial. For the second condition, we observe that starting from the parameters $(r_c, r_s, \text{premaster})$, key generation is deterministic and thus both parties derive the same master secret and application keys whenever they have the same sid. □ □

We now turn to the security models that we use for the TLS primitive and the TLS channel security.

6.3 Length-Hiding-Authenticated Encryption Models

Paterson et al. [PRS11] recently proved that the TLS record protocol meets a notion called length-hiding authenticated encryption (LHAE). This notion says that an adversary cannot distinguish, in the usual left-or-right sense, between encryptions of messages which are not necessarily of the same length. In addition, the adversary is unable to generate new ciphertexts for which the decryption algorithm does not return an error message. Both properties are combined into a single game, in which the adversary gets access to an left-or-right encryption oracle (with secret key K and bit b) which for input (ℓ, H, M_0, M_1), the length parameter ℓ, the header data H and two messages M_0, M_1, computes $C_0 \leftarrow \mathsf{LHEnc}_K(\ell, H, M_0)$, $C_1 \leftarrow \mathsf{LHEnc}_K(\ell, H, M_1)$, and returns C_b if both $C_0, C_1 \neq \perp$, and $\perp$ else. If it returns a ciphertext, it adds the pair (H, C) to the initially empty list $\mathcal{L}_{\mathsf{Enc}}$. The decryption oracle, when called about H, C, rejects if $b = 0$ or (H, C) is in $\mathcal{L}_{\mathsf{Enc}}$, i.e. comes from a previous query to the left-or-right oracle. Else, it returns $\mathsf{LHDec}_K(H, C)$.

Definition 31 (LHAE security)**.** *We say that the LHAE-scheme* (KeyGen, LHEnc, LHDec) *is* $\frac{1}{2}$*-secure, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, the probability that the game* G_{LHAE} *outputs* 1 *is at most negligibly greater than* $\frac{1}{2}$*. This probability is taken over all random bits of the game, of the key generation algorithm and of the adversary.*

Note that unlike in our definition, the decryption oracle in Paterson et al. [PRS11] rejects all previously returned ciphertexts C and not only previous header-ciphertext *pairs* (C, H). However, they prove that TLS also satisfies this notion of ciphertext integrity. And as both security notions use the same oracles, the results compose and show the security of the TLS primitive in the above game.

We now define a multi-session version of the above game as the TLS primitive game. One further modification is that pairs of the form $(0|H_1|H_2, C)$ never count as successful forgeries, although they might be "fresh". Whenever a new key is initiated with session identifier kid, the TLS primitive game runs the key generation algorithm KeyGen which returns two random strings (κ_c, κ_s). Moreover, it flips two random coins b^c_{kid} and b^s_{kid}, one for the client and one for the sever in that session, and initiates the two lists $\mathcal{L}_{\mathsf{Enc}}(kid, s) := \emptyset$, and $\mathcal{L}_{\mathsf{Enc}}(\mathsf{kid}, c) := \emptyset$. Upon a query $\mathsf{NewKey}(U, V)$, besides the actions mentioned in Section 3.1 for the primitive game, there are several queries allowed to the adversary given below. Let $u \in \{c, s\}$:

- $\mathsf{LHEnc}(\mathsf{kid}, u, \ell, n|H_1|H_2, m_0, m_1)$:
  Run $C_b \leftarrow \mathsf{LHEnc}_{\kappa_u}(\ell, n|H_1|H_2, m_b)$ and return C_b, if $C_b \neq \perp$ and $\perp \neq C_{1-b} \leftarrow \mathsf{LHEnc}_{\kappa_u}(\ell, n|H_1|H_2, m_{1-b})$. Set $\mathcal{L}_{\mathsf{Enc}}(\mathsf{kid}, u) := \mathcal{L}_{\mathsf{Enc}}(\mathsf{kid}, u) \cup \{(H, C_b)\}$.

- $\mathsf{LHDec}(\mathsf{kid}, u, n|H_1|H_2, C)$:
  If $(n|H_1|H_2, C) \in \mathcal{L}_{\mathsf{Enc}}(\mathsf{kid}, u)$ return $\perp$, else run $m \leftarrow \mathsf{LHDec}_{\kappa_u}(n|H_1|H_2, C)$. If now $b = 1$ *or* $n = 0$, return m. Else, return $\perp$.

- $\mathsf{Target}(\mathsf{kid}, u, b)$:
  If $b = b^u_{\mathsf{kid}}$ and $\mathrm{st}_{\mathsf{kid}}$ is not $\mathsf{corrupted}$, return 1. Else, return 0.

Definition 32 (TLS Primitive Game)**.** *We say that the TLS primitive* (KeyGen, LHEnc, LHDec) *is* $\frac{1}{2}$*-secure, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, the probability that the game* $G_{TLS\text{-}Prim}$ *outputs* 1 *is at most negligibly greater than* $\frac{1}{2}$*. This probability is taken over all random bits of the game, of the key generation algorithm and of the adversary.*

Note that the games above do not touch the issues of replay attacks, package reordering, package dropping, etc. and do thus do not provide a secure channel per se. Similarly, the stateful version of such games, such as the one proposed in [JKSS12] which is attributed to [PRS11], appears to be even closer to the properties one would expect from a secure channel, but it still does not seem to capture the aforementioned properties. Moreover, the definition in [JKSS12] seems to assume that sender and receiver share counters, as their decryption oracle uses the counter value i from the encryption step. While counters are a common mean to build secure channels we believe they should not be part of the security definition.

We nonetheless rely on their definition of stateful LHAE. Starting from the LHAE primitive we next define an LHAE-channel game, equally as a multi-session-game. Each session involves two users, say a client and a server, and a channel in each direction. Each direction is indexed by the server or the client and consists of a sending and a receiving oracle. For both, the server and the client key, $u \in \{c, s\}$, the game relies on a queue $\mathcal{Q}(\mathsf{kid}, u)$ with two methods, $\mathsf{Enqueue}(C)$ and $\mathsf{Dequeue}()$, with the usual semantics that $\mathsf{Enqueue}(C)$ puts an element into the data structure, and $\mathsf{Dequeue}$ returns an element (or $\perp$, if empty). We assume the usual first-in first-out behavior of $\mathcal{Q}$. For sake of distinction we refer to the two oracles to which the adversary in the game has access, as the sender $\mathsf{sender}(\mathsf{kid}, u)$ and the receiver $\mathsf{receiver}(\mathsf{kid}, u)$, respectively.

Both oracles are initialized by a generation algorithm KeyGen with the symmetric key κ_u and some fixed initial state st_0. The sender oracle also holds a random secret bit b. If called on ℓ, m_0, m_1, then, assuming it is in state st^{sender}, it runs $(C_0, st_0^{\mathsf{sender}}) \leftarrow \mathsf{Send}_{\kappa_u}(\ell, m_0; st^{\mathsf{sender}})$, $(C_1, st_1^{\mathsf{sender}}) \leftarrow \mathsf{Send}_K(\ell, m_1; st^{\mathsf{sender}})$, and returns C_b and sets $st^{\mathsf{sender}} \leftarrow st_b^{\mathsf{sender}}$, if both encryptions succeed; in this case it also calls $\mathcal{Q}(\mathsf{kid}, u).\mathsf{Enqueue}(C_b)$. Else it merely returns $\perp$. The receiver oracle, if called about some C, first runs $(m, st_*^{\mathsf{receiver}}) \leftarrow \mathsf{Receive}_{\kappa_u}(C; st^{\mathsf{receiver}})$. It then returns $\perp$, unless $b = 1$ and $\mathcal{Q}.\mathsf{Dequeue}() \neq C$, in which case it returns M. As a multi-session game, we also allow a query $\mathsf{Target}(\mathsf{kid}, u, d)$, where the game returns 1 if and only if $b = d$, and the session kid is uncorrupted.

Definition 33 (TLS Channel)**.** *We say that the TLS Channel* $(\mathsf{KeyGen}, \mathsf{Send}, \mathsf{Receive})$ *is* $\frac{1}{2}$*-secure, if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, the probability that the game outputs* 1 *is at most negligibly greater than* $\frac{1}{2}$*. This probability is taken over all random bits of the game, of the key generation algorithm and of the adversary.*

6.4 The Key Exchange is Suitable for the Primitive

In this section, we prove that the TLS Handshake is suitable the TLS primitive according to Definition 32.

Theorem 5 (Suitability for Primitive)**.** *The TLS Handshake protocol is* $(G_{TLS\text{-}Prim}, \frac{1}{2})$*-suitable for the TLS primitive* $(\mathsf{KeyGen}, \mathsf{LHEnc}, \mathsf{LHDec})$ *if*

- *the encryption scheme used in the Record Layer is LHAE-secure,*
- *the certification authority uses an UNF-CMA signature scheme*[1] *,*
- *the signature scheme for the pre-master-secret phase is UNF-CMA,*
- *the Diffie-Hellmann assumption holds resp. the key transport encryption scheme is IND-CCA,*
- *the MAC scheme for the* `FINISHED` *message is UNF-CMA, and*

[1] More abstractly, any kind of UNF-CMA certification scheme would work, but we stick to signature-based certificates for sake of simplicity.

- *the deployed hash function is modeled via a random oracle.*

Proof. We say that $(\mathsf{label}_1, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathsf{sid}_1, \mathrm{st}_{\mathrm{exec}1}, \kappa_1, \mathrm{st}_{\mathrm{key}_1})$ is temporarily partnered with $(\mathsf{label}_2, \mathsf{kid}_V, \mathsf{kid}_U, V, U, \mathsf{sid}_2, \mathrm{st}_{\mathrm{exec}2}, \kappa_2, \mathrm{st}_{\mathrm{key}_2})$, if the transcripts of these sessions contain the same pair of nonces. With overwhelming probability, at any point in the protocol (after the hello messages), each session has at most one partner session of this form.

The proof consists of several game hops. We define games 0 to 6 as follows.

- Game 0: The original game.
- Game 1: The parties execute step 8 of the protocol before step 5, i.e., the client sends his certificate verify message before deriving the keys, and the server verifies the signature, before deriving the keys. Moreover, the game aborts and returns 1, whenever there is a random oracle collision amongst all queries asked to it by the game and the adversary. Moreover, the game aborts with output 1, if a user chooses the same random string for the hello message more than once.
- Game 2: Let n be an upper bound on the number of $\mathsf{NewSession}$ queries that $\mathcal{A}$ asks. Game 1 draws a random number k between 1 and n. Let $(\mathsf{label}, \mathsf{kid}_U, \mathsf{kid}_V, U, V, \mathsf{sid}, \mathrm{st}_{\mathrm{exec}}, (\kappa_c, \kappa_s), \mathrm{st}_{\mathrm{key}})$ be the k'th session of the protocol. Throughout the execution of the game, the game checks whether one of the following events occurs (we distinguish between the key-exchange implemented with DH exchange or via key-transport): a) Key Transport: The keys corresponding to kid_V or kid_U are corrupted and b) DH: kid_V or kid_U is corrupted before session label accepts. As soon as one of these events occurs, the game aborts and returns a uniformly distributed bit.
  - U computes a key in session label which accepts, *and* this key is revealed in the G_{ke} subgame or corrupted in the G_ζ subgame.
  - U sends a *Finished* message in session label, and upon receiving this message, a session label' of V accepts its key, *and* this key is revealed in G_{ke} or corrupted in G_ζ.
  - The final output of the adversary contains as key identifier a value kid' that does not correspond to the key which is output of session label, respectively, session label did not accept.
- Game 3: the game aborts when one session accepts a certificate that is for a different key than the one that belongs to the partner.
- Game 4: the game aborts, when label and label' compute different pre-master secrets, but label and label' accept the key nevertheless.
- Game 5: the master secrets for session label and its partner are replaced with uniformly random keys.
- Game 6: the application keys session label and its partner are replaced with uniformly random keys.

It remains to transform any successful adversary $\mathcal{A}$ against game 6 into a successful adversary $\mathcal{B}$ against the LHAE game G_{LHAE}. For all sessions except for label and its temporal partner label', the adversary $\mathcal{B}$ simulates game 6 as described. For label and label', it proceeds as follows: the adversary $\mathcal{B}$ flips a random bit to decide, whether it uses G_{LHAE} for the encryption under the server's sending key or the client's sending key.

For ease of presentation, say, that it chooses the server's sending key. Then, $\mathcal{B}$ modifies the server's `FINISHED` message as follows: it submits $(\ell, 0|H_1^s|H^s, \tau_2)$ to the encryption oracle of G_{LHAE} and returns the output as the `FINISHED` message. To simulate the client when receiving this message, the adversary $\mathcal{B}$ accepts, if the message was transmitted in an unmodified way, and else, submits the modified message to the decryption oracle of G_{LHAE}. It rejects, whenever G_{LHAE} rejects, and else uses the decrypted value to check whether this is a valid MAC under the master secret. For any further query of the adversary $\mathcal{A}$ for the server's sending key, $\mathcal{B}$ relays the respective queries and answers between $\mathcal{A}$ and G_{LHAE}. Whenever an abort occurs in game 6, $\mathcal{B}$ returns a random bit.

Analysis. We first analyze the game hops and then show that if $\mathcal{A}$ has non-negligible advantage, then so has $\mathcal{B}$.

- Game 0 to game 1: as key derivation does not trigger any output in the game, the order of the computation does not change the game's interaction with the adversary. The probability of random oracle collisions is negligible, and so is the collision probability amongst random strings of a user.
- Game 1 to game 2: Let $\mathcal{A}$ be an adversary playing game 1. Let $p_0 := \frac{1}{2}+p$, with $p > 0$, be the probability of $\mathcal{A}$ winning game 1. Let n be an upper bound on the number of NewSession queries made by $\mathcal{A}$. The game guesses kid correctly with probability at least $\frac{1}{n}$, and in this case, the adversary $\mathcal{A}$ wins with probability at least $\frac{1}{2}+p$, because if the adversary wins, then the session accepted and is uncorrupted. Else, the adversary wins with probability at least $\frac{1}{2}$. Thus, the adversary's overall success probability in Game 1 is
$$\frac{1}{n}(\frac{1}{2}+p) + \frac{n-1}{n}\frac{1}{2} = \frac{1}{2} + \frac{1}{n}p.$$
- Game 2 to Game 3: the event that one accepts a certificate for another key than the partner's key is negligible, as the authority's signature scheme is unforgeable.
- Game 3 to Game 4: we have to bound the probability that they accept although they derive different keys. As key derivation is deterministic, it suffices to show that if they accept, they derived the same pre-master secret with overwhelming probability. In the following, we assume that the random oracle is collision-free amongst all queries queried by the game and the adversary, and that random nonces never occur twice.

  Let us consider the Diffie-Hellman-case first. As random nonces do not occur twice and as the random oracle is collision-free, the adversary either transfers the Diffie Hellman parameters of the server correctly, or modifies the Diffie Hellman parameters to (p^*, g^*, Y_s^*) and sends a signature over a random oracle value of the form $\text{RO}(r_c||r_s||p^*, g^*, Y_s^*)$. However, the server never issued such a signature by uniqueness of the nonce r_s. Thus, if the adversary had non-negligible probability in creating a valid signature over modified parameters, we could break the unforgeability of the underlying signature scheme. Similar reasoning, applied to the Client Authentication message and the earlier Diffie–Hellman flows, assures that the client's parameter Y_c is correctly transmitted. Thus, if both parties accept the Verify messages then with overwhelming probability, both parties hold the same parameters (p, g, Y_s, Y_c) and thus derive the same pre-master secret.

For the key transport case, it suffices to argue that the ciphertext C is correctly transmitted from the client to the server. Again, the uniqueness of the nonce r_c together with collision-freeness of the random oracle, and the unforgeability of the client's signature scheme guarantee that, if the server accepts, then the adversary has modified the ciphertext C only with negligible probability.

- Game 4 to Game 5: It suffices to show that the adversary does not query the random oracle on the value of the pre-master secret of session label. If so, the value of the master secret is statistically hidden from the adversary. Note that in this case, the modification does not affect other sessions, even if they happen to derive the same pre-master secret as the random oracle is queried on (premaster, "master secret", r_c, r_s) and the pair of randomnesses never occurs twice.

Diffie-Hellman Case. If an adversary $\mathcal{A}$ queries the random oracle on the pre-master secret with non-negligible probability, we can break the computational Diffie-Hellman (CDH) assumption (see Section 2) as follows. The adversary $\mathcal{B}$ against CDH gets (p, q, g, g^a, g^b) from the challenger and has to output a guess for g^{ab}. The adversary $\mathcal{B}$ simulates Game 4 with one modification. Let n be an upper bound on the sessions that $\mathcal{A}$ invokes. Then, $\mathcal{B}$ guesses a random value i between 1 and n embeds the parameters in the ith session and skips the pre-master derivation step for this session by directly choosing a random value for the master secret. Moreover, let q be an upper bound on $\mathcal{A}$'s random oracle queries. Then, $\mathcal{B}$ chooses a random value k between 1 and q and returns a prefix of the kth query of $\mathcal{A}$ to its oracle as a guess for g^{ab}, where the length of the prefix is the length of the pre-master secret.

For the analysis note that, before the adversary $\mathcal{A}$ queries (g^{ab}, "$mastersecret$", r_c, r_s) to the random oracle, the simulation is perfect. Thus, if $p(\lambda)$ is the probability that $\mathcal{A}$ queries the pre-master secret, i.e., queries (g^{ab}, "$mastersecret$", r_c, r_s), to the random oracle, then $\mathcal{B}$'s success probability in correctly determining g^{ab} is at least $\frac{p}{qn}$.

Key Transport Case. An analog analysis applies to the key transport case. Here, the security is reduced to the IND-CCA2-security of the encryption scheme. Let $\mathcal{A}$ be an adversary that queries the random oracle on the pre-master secret with non-negligible probability, let n be an upper bound on the number of users that he initiates and s be an upper bound on the number of sessions of this user. The adversary $\mathcal{B}$ simulates game 4 with one modification. It picks a random user to embeds the public key, and a random session i of this user to embed the challenge ciphertext, i.e., the adversary $\mathcal{B}$ draws two random strings $\mathsf{premaster}_0$ and $\mathsf{premaster}_1$ and send them to its encryption oracle to receive a ciphertext C that it embeds in session i. It skips the pre-master derivation step for this session by directly choosing a random value for the master secret. Moreover, let q be an upper bound on $\mathcal{A}$'s random oracle queries. Then, $\mathcal{B}$ chooses a random value k between 1 and q and returns a prefix of the kth query of $\mathcal{A}$ to its oracle as a guess for premaster, where the length of the prefix is the length of the pre-master secret. Moreover, all other sessions of this user are handled by using the decryption oracle - unless the same ciphertext occurs again, in

which case the adversary $\mathcal{B}$ also picks the master secret for these sessions at random. Note that these values are chosen independently, as the random oracle is queried on ($\mathsf{premaster}_b$, "$mastersecret''$", r_c, r_s), and collisions amongst nonces do not occur. If in any of the adversary's queries, the value ($\mathsf{premaster}_b$, "$mastersecret''$", r_c, r_s) is queried to the random oracle, then $\mathcal{B}$ returns b as its output. Else, it returns a random bit b.

For the analysis, note again that, before the adversary $\mathcal{A}$ queries ($\mathsf{premaster}_b$, "$mastersecret''$", r_c, r_s) to the random oracle, the simulation is perfect. Thus, if $p(\lambda)$ is the probability that $\mathcal{A}$ queries the pre-master secret, i.e. ($\mathsf{premaster}_b$, "$mastersecret''$", r_c, r_s), to the random oracle, then $\mathcal{B}$'s success probability in correctly determining b is at least $\frac{p}{sn}$ minus the negligible probability, that by coincidence, $\mathcal{A}$ queries about the (statistically hidden) value $\mathsf{premaster}_{1-b}$.

- Game 5 to Game 6: As before, it suffices to show that with overwhelming probability, the adversary does not query the random oracle on the master secret (more precisely, about ($\mathsf{master}||$"key expansion"$||r_s||r_c$)). Let $\mathcal{A}$ be an adversary which queries the random oracle on the master secret with non-negligible probability. Then, we construct an adversary $\mathcal{B}$ against the UNF-CMA property of the underlying Mac. Let n be an upper bound on the number of sessions that $\mathcal{A}$ initiates. The adversary $\mathcal{B}$ simulates game 4 with one modification. It draws a random number i between 1 and n. In the ith session, instead of querying the random oracle on the master secret, it picks two random values for the application keys. To compute the Mac values in the *Finished* messages, the adversary $\mathcal{B}$ queries the Mac oracle. Let q be an upper bound on the queries that $\mathcal{A}$ makes. Then, $\mathcal{B}$ draws a random number k between 1 and q and does the following for $\mathcal{A}$'s kth query to the random oracle. It uses a prefix of the length of the master secret and computes a Mac of a fresh message. It submits the Mac and the message to the unforgeability game.

Analysis Unless the adversary queries the random oracle on the master secret respectively on (master, "keyexpansion", r_c, r_s), the simulation is perfect. Thus, if $p(\lambda)$ is $\mathcal{A}$'s probability in querying the master secret to the random oracle, then $\mathcal{B}$'s success probability is at least $\frac{p}{qn}$ minus the negligible correctness error of the Mac scheme.

We now prove that if $\mathcal{A}$ is a successful adversary against game 6, then the adversary $\mathcal{B}$ that we constructed, is a successful adversary against G_{LHAE}. If $\mathcal{A}$ never makes any fresh query of the form $(0|H_1|H_2, C)$ such that the decryption algorithm does not reject, then the simulation is perfect. It thus suffices to bound this probability. Assume that $\mathcal{A}$ had non-negligible probability p in making successful fresh decryption queries of the form $(0|H_1|H_2, C)$. Then, we can use $\mathcal{A}$ as a distinguisher against G_{LHAE} by outputting 1, whenever G_{LHAE} accepts such a query and by outputting 0, else. Then, we win G_{LHAE} with probability $\frac{1}{2} \cdot p + \frac{1}{2} \cdot 1$. □ □

6.5 Key-Independent Reduction

In this section, we prove that the security of the TLS Record Layer Protocol reduces to the security of the TLS primitive. We now describe how to build the TLS channel

algorithms (KeyGen, Send, Receive) from the TLS Primitive. As mentioned before, we work with an abstracted version of headers etc. and thereby cover all implementations that could be shown LHAE-secure and that use counters in the way described below. To this end, let Header be a stateful, public algorithm that on input a message, a length parameter ℓ and a sequence number n returns a header $n|H_1|H_2$ with sequence number n and similarly for ReceivingHeader, which only returns the locally stored part $n|H_1$ of the header.

Algorithm KeyGen initializes the states for Header and ReceivingHeader, and initializes two counter values cnt_s and cnt_c by 0. It then draws two random keys (κ_c, κ_s) for the LHEnc scheme and initializes the states of boths senders and both receivers. The Send algorithm runs the algorithm Header the message, the parameter ℓ and the counter cnt_u to yiel $n|H_1|H_2$. Then, the Send algorithms runs $\mathsf{LHEnc}_{\kappa_u}(\ell, n|H_1|H_2, m)$ that returns a ciphertext C or $\perp$. If $C \neq \perp$, then Send increments its counter cnt_u by 1 and returns (C, H_2) as the channel message. The Receive algorithm on input (C, H_2) runs ReceivingHeader to return a header $n|H_1$. Receive then runs $\mathsf{LHDec}_{\kappa_u}(n|H_1|H_2, C)$ that returns a message m and an updated state. If $m \neq \perp$, then cnt_u is incremented by 1. When initializing the states, the KeyGen algorithm chooses an appropriate value ℓ and sends the message 0 on both channels and receives the message on both channels. This increments all counters to 1. Note that neither Header nor ReceivingHeader requires any secret information at any point.

Theorem 6 (Protocol reduces to Primitive)**.** *There is a key-independent reduction from the security of the TLS Record Layer as a TLS-channel to the security of the TLS Primitive according to the TLS Primitive Game.*

Proof. Let $\mathcal{A}$ be a successful adversary against the TLS-channel, then we construct $\mathcal{B}$ as follows: for all (kid, u) with $u \in \{c, s\}$, the adversary $\mathcal{B}$ stores the public information for the header algorithm with the counter initially set to 0. It then sends the message 0 on both channels and receives the message on both channels.

For all send queries $(\mathsf{kid}, u, m_0, m_1, \ell)$ that $\mathcal{A}$ sends, $\mathcal{B}$ generates the header $n|H_1|H_2$ using the public header algorithm and queries $(\mathsf{kid}, u, n|H_1|H_2, m_0, m_1, \ell)$ to the LHEnc oracle. If the encryption oracle returns a ciphertext C, then $\mathcal{B}$ passes (H_2, C) to $\mathcal{A}$ and increments n by 1. Else, $\mathcal{B}$ returns $\perp$ to $\mathcal{A}$ and does not increment the counter. For any decryption query $(\mathsf{kid}, u, C, H_2)$, the adversary $\mathcal{B}$ generates $n|H_1$ using ReceivingHeader and queries $(\mathsf{kid}, u, n|H_1|H_2, C)$ to the LHDec oracle. If the answer is not $\perp$, it increments the counter by 1 and returns the answer. Else, it leaves the sequence number unchanged and returns $\perp$. In the end of the game, $\mathcal{B}$ relates $\mathcal{A}$'s Target query.

Analysis. To see that the distributions in both games are equal and thus the winning probabilities, we only have to argue that whenever $\mathcal{A}$ submits a ciphertext $C|H_2$ to the Receive oracle such that the latter does not return $\perp$, then the tuple $(\mathsf{kid}, u, n|H_1|H_2, C)$ is also "fresh" for the LHDec oracle. Firstly, $n \leq 1$. Moreover, C was not output by LHEnc as the nth query (as else, the game would have rejected it). Thus, $(n|H_1|H_2, C) \neq (H, C)$ for all previously queried (H, C). The analysis for the freshness condition applies to all key distributions, and thus, the reduction is key-independent. □

Chapter 7

Complexity-Theoretic Assumptions

In the previous chapter, we saw that the security of the TLS Handshake protocol is based on the Diffie-Hellman assumption respectively on the CCA2-security of an encryption scheme that is used for key transport. As a scheme is only as secure as the underlying problem is hard, an entire area of the foundations of cryptography targets at building cryptographic schemes from the weakest assumptions possible. To understand the different nature of cryptographic assumptions, let us consider the following three candidate assumptions:

(i) $\mathcal{NP} \neq \mathcal{P}$,

(ii) the existence of one-way functions,

(iii) the Diffie-Hellman assumption.

As the Diffie-Hellman assumption implies one-way functions, and as one-way functions imply hard problems in $\mathcal{NP}$, it would be most desirable to base a cryptographic scheme merely on the assumption that $\mathcal{NP}$ does not equal $\mathcal{P}$. However, there are substantial problems in doing so, e.g., that hardness of $\mathcal{NP}$ is a worst-case assumption, while cryptography requires average-case hardness. Given the choice between a general assumption such as the existence of one-way functions and a more specific assumption such as the Diffie-Hellman assumption, there are three advantages in building cryptography from general one-way functions: firstly, the construction is more general, i.e., the deployed one-way function can be realized via some Diffie-Hellman-type assumption, via lattice-assumptions, codes or even new assumptions that are yet to be discovered. Secondly, the assumption that one-way functions exist is weaker and probably even strictly weaker than the Diffie-Hellman assumption. Finally, the existence of one-way functions can be considered as a cryptographic average-case analog of the $\mathcal{NP}$ versus $\mathcal{P}$ question, i.e., not only do we need hard $\mathcal{NP}$-instances, but also, we want to sample them efficiently together with a solution, that is, we want to generate an instance which is easy for ourselves, but difficult for everybody else.

Looking at one-way functions in this way, one might not be surprised that most of cryptography, if not all of it, implies one-way functions [IL89]. Maybe more surprisingly, one-way functions are not only a necessary but also a sufficient assumption for many cryptographic tasks. The set of cryptographic primitives that are existentially equivalent to one-way funcstions are usually refered to as "Minicrypt" [Imp95] and include, e.g., symmetric-key encryption, message authentication codes and even signature schemes [Rom90]. Overall, the existence of one-way functions can be considered the most foundational question in cryptography, similar to the $\mathcal{NP}$ versus $\mathcal{P}$ question in complexity theory. Noteworthy, the non-existence of one-way functions would bury the entire area of cryptography.

Despite the importance of the question, we still know very little about one-way functions. In particular, after several decades of research in complexity theory, our confidence that $\mathcal{NP} \neq \mathcal{P}$ is much higher than our belief in the existence of one-way functions. Thus, the holy grail in the area of the foundations of cryptography is to answer the following question:

Can we base one-way functions on the mere assumption that $\mathcal{NP} \neq \mathcal{P}$?

Giving a positive answer to this question would constitute a major breakthrough in our understanding of cryptography—giving a negative answer would be as interesting although not as re-assuring. We mentioned that one-way functions are necessary for all of cryptography. Yet, since the seminal work by Impagliazzo and Rudich [IR90], we know that one-way functions do not suffice for more "fancy" cryptography such as trapdoor functions, oblivious transfer and public-key encryption. This set of cryptographic primitives is generally refered to as "Cryptomania" [Imp95], and the weakest assumption in Cryptomania is the existence of key agreement. The non-existence of key agreement would collapse Cryptomania to Minicrypt. Naturally, we would like to know:

Which assumptions do we need for key agreement besides one-way functions?

In this chapter, we cover the state-of-the-art in both areas of research and contribute via two oracle separations to each of them.

7.1 On Basing One-Way Functions on $\mathcal{NP}$-Hardness

Luckily for the cryptographers, most researchers believe that one-way functions exist. Similarly, most researchers conjecture that **NP** does not equal **P**. On the downside, our confidence that $\mathbf{NP} \neq \mathbf{P}$ is considerably higher than our belief in the existence of one-way functions. So, we would like to relate the existence of one-way functions to the $\mathbf{NP} \neq \mathbf{P}$ conjecture. Indeed, we know that if one-way functions exist, then $\mathbf{NP} \neq \mathbf{P}$, but ehe converse is an open question; a positive and a negative answer both seem a plausible outcome. Lattice-based cryptography and their associated worst-case to average-case reductions repeatedly gave hope to a positive answer, i.e., a one-way function based on an $\mathcal{NP}$-hard problem. Yet, it often turns out that the security of

such a candidate one-way function actually relies on a hard problem in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ instead of an $\mathcal{NP}$-hard problem [AR05, GG98, Ajt98, HR07, AD07, Cai98]. Similarly, a look on existing impossibility results [FF93, BT03, AGGM06, AGGM10, Wat10, Imp11] reveals that only non-adaptive reductions have been ruled out so far, i.e., reductions that determine their set of queries to the oracle and then ask them all at once. On the one hand, these negative results indicate that finding a reduction is difficult. On the other hand, they still leave ample loopholes to bypass them.

A noteworthy exception is the work by Akavia et al. [AGGM06]. Indeed, based on the assumption that $\mathcal{NP} \not\subseteq \mathbf{co}\mathcal{AM}$, they prove that, even via adaptive reductions, we cannot construct a regular one-way function from $\mathcal{NP}$-hardness. Inspecting their proof, a meta-reduction, even yields a stronger result, namely that regular one-way functions already imply hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. They also give an easier proof of the same statement, if additionally, one assumes that the regular function only has polynomially many pre-images per image. Notably, Akavia et al. are the first ones in this area to rule out *adaptive* reductions.

Unfortunately, in 2010, Akavia et al. [AGGM10] discovered a gap in the proof of the first theorem which rules out non-adaptive reductions from general regular one-way functions to $\mathcal{NP}$-hardness. The gap is related to universal hash functions in lower bound protocols that were mistakenly assumed to be drawn uniformly and independently from related values. In light of the limitations of more involved protocols with similar goals such as [HMX10], the gap in the proof seems to be hard to overcome. We provide a possible explanation, namely, we show that a new proof for their theorem would need to rely on non-relativizing techniques. Recall that their theorem would also imply that regular one-way functions entail hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. In contrast, we give an oracle relative to which there are no hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ while there are regular functions that are one-way for an infinite number of security paramters.

Towards this goal, we combine the generic oracle technique by Blum and Impagliazzo [BI87] with Nisan's [Nis91] notion of block sensitivity and certificate complexity. Our result shows that it is impossible to prove that one-way functions imply hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ via relativizing techniques.

7.2 On Necessary Assumptions for Key Agreement

One-way functions are a necessary assumption for Minicrypt and key agreement [IL89], yet not sufficient [IR90] for key agreement. Similarly, key agreement is a necessary condition for all of Cryptomania. Understanding Cryptomania thus requires to determine necessary and sufficient assumptions for key agreement that are easier than, say, the security definition by Bellare and Rogaway or any of the other security models developed in this thesis. Indeed, throughout this chapter, we will be able to work with the following simpler definition of passively secure key agreement. One might by tempted to argue that a weaker definition of key agreement strenthens our separation results. However, as we know by Impgliazzo and Luby [IL89], passively secure key agreement already yields one-way functions and thus, by Rompel [Rom90], digital signatures. Careful composition

of signatures and passively secure key agreement then yields, indeed, a Bellare-Rogaway secure key exchange protocol. And thus, passively secure key agreement is existentially equivalent to key exchange protocols that are secure in the Bellare-Rogaway model, so that separations for one of them immediately yields a separation for the other one.

Definition 34 (Secure Key Agreement)**.** *A key agreement protocol π consists of two efficient randomized algorithms A (Alice) and B (Bob) that receive as input the security parameter 1^n, run a protocol where they generate a transcript T and each of them outputs a key; a run is denoted by $(k, T, k') \leftarrow \langle A(1^n), B(1^n)\rangle$, where T is the transcrip, k is Alice's key and k' is Bob's key. Correctness requires that Alice's and Bob's key are equal with overwhelming probability, i.e.,*

$$\epsilon(n) := \text{Prob}\left[\, k \neq k' | (k, T, k') \leftarrow \langle A(1^n), B(1^n)\rangle\right]$$

is a negligible function in n. A key agreement protocol is called perfectly correct*, if $\epsilon(n) = 0$. In this case, k always equals k' and thus, we may write $(k, T) \leftarrow \langle A(1^n), B(1^n)\rangle$ by a slight abuse of notation.*

We say that π is secure*, if for all efficient adversaries $\mathcal{A}$, we have that*

$$\text{Prob}[\,\mathcal{A}(1^n, T, k_b) = b] \approx \tfrac{1}{2},$$

where b is a random bit, $k_0 := k$ for $(k, T, k') \leftarrow \langle A(1^n), B(1^n)\rangle$ and $k_1 \leftarrow \{0,1\}^{|k_0|}$. A key agreement protocol is called infinitely often secure*, if there is a strictly monotone sequence $(n_i)_{i\in\mathbb{N}}$ of natural numbers such that for all efficient adversaries $\mathcal{A}$,*

$$\text{Prob}[\,\mathcal{A}(1^{n_i}, T, k_b) = b] \approx \tfrac{1}{2}.$$

As we know that this type of key agreement implies one-way functions [IL89], and as one-way functions imply signatures [Rom90], we can turn a passively secure key agreement, which is only secure against eavesdroppers, against a BR-secure key agreement protocol by the signature-based compiler of Katz and Yung [KY07]. Thus, existentially, it suffices to consider the simpler version of key agreement as given in Definition 34. Although easier to assess, Definition 34 still involves randomness and interaction and is a fairly involved statement to study.

Likewise, sufficient assumptions for key agreement such as the Diffie-Hellman assumption (Definition 4) reflect the interactive structure. An interesting observation is that most concrete assumptions such as DDH imply hard problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$. Indeed, substantial effort has been made to construct secure key agreement from problems *outside* $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$, in particular in the area of lattice-based cryptography. Unfortunately, most lattice-based schemes turn out to be related to hard problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ or at least in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ [AR05, GG98, Ajt98, HR07, AD07, Cai98]. Our claim is that, although no positive result is known, there are no relativizing reasons that prevent building key agreement from assumptions outside $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$.

In the next subsections, we show that the Diffie-Hellman assumption implies hard problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$. We then give a proof that a subclass of secure key agreement protocols implies hard problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ and show that lattice-based schemes are unlikely to fall into this category of protocols. Our contribution is an oracle relative to which secure key agreement exists while $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is contained in $\mathcal{P}$.

Diffie-Hellman and $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ Let us consider the folklore statement that the Diffie-Hellman assumption implies a hard problem in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$. Consider the language of tuples (G, g, g^a, g^b, g^{ab}), where G is the group description. Assume that it is easy to check whether a 5-tuple (G, g, g^a, g^b, g^c) is a valid encoding of a group description and four group elements, then the three exponents (a, b, c) form a witness either of the fact that the 5-tuple is in the language or outside the language. Namely, on input the witness (a, b, c) and the 5-tuple (G, g, A, B, C), let the $\mathcal{NP}$-verifier and the $\mathbf{co}\mathcal{NP}$-verifier both first verify whether indeed, a, b and c are valid discrete logarithms, i.e., $g^a = A$, $g^b = B$ and $g^c = C$. Then, let us have the $\mathcal{NP}$-verifier return 1 if and only if $ab = c$ and let the $\mathbf{co}\mathcal{NP}$-verifier returns 1 if and only if $ab \neq c$. If this language were easy, one could break the Diffie-Hellman assumption with overwhelming probability. Thus, the Diffie-Hellman assumption implies that there is a hard problem in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$.

Key Agreement and $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ We now give a (slightly flawed) proof that a secure key agreement protocol implies hard problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$, namely the language $\mathcal{L}$ of all pairs (i, T), where T is a protocol transcript such that the ith bit of the shared secret associated with the transcript is 1. A witness for $\mathcal{L}$ is the randomness used by the two parties to generate the transcript T. Knowing their randomness, one can (efficiently) simulate their local computation and thus derive the shared secret k. One can then simply check whether the ith bit of k is 0 or 1. Similarly, if (i, T) is such that the ith bit of the key associated with the transcript T, is 0, then the randomness can serve a witness to prove that the pair (i, T) is not in the language $\mathcal{L}$. Thus, for the language $\mathcal{L}$, we can witness membership and non-membership. Moreover, if the underlying key agreement protocol is secure, then it should be hard to decide $\mathcal{L}$. In conclusion, if key agreement exists, then there must be hard problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$.

There are two catches here. Firstly, $\mathcal{L}$ is rather a promise problem than a language problem, as not all strings T are necessarily valid protocol transcripts—moreover, the transcript range need not be co-range-verifiable, i.e., it might be hard to prove to somebody that a certain string is *not* a valid transcript. Secondly, T only defines a single key k if we assume perfect correctness of the key agreement protocol, i.e., the two parties always agree on the same key. In the case of imperfect correctness, the same transcript might be associated with several keys. We thus obtain two conditions of which either can potentially be used to circumvent the above proof:

(i) The key agreement protocol is not perfectly correct.

(ii) The transcript function is not co-range-verifiable,

which means that it is hard to prove to somebody (in $\mathcal{NP}$ or even $\mathcal{AM}$) that a string T is not a possible transcript of the key exchange protocol.

Indeed, lattice-based schemes often satisfy both or at least one of these two conditions. When encrypting in lattice-based schemes, one usually chooses a lattice point and disturbs it to obtain another point in the plane. The receiver has some trapdoor to recover the lattice point that is closest to this point in the plane. If one disturbs the lattice

point only slightly then the original lattice-point is uniquely determined. However, the more one disturbs the lattice point, the harder the problems gets, i.e., the more secure the scheme gets. And thus, most schemes disturb the lattice point very much so that with some small probability, one ends up closer to another lattice point than the original one. Thereby, the encryption scheme looses its perfect correctness property—correctness only holds with overwhelming probability now.

In turn, when aiming at perfect correctness, then one has to disturb the point only slightly. In this case, however, there are large areas in the plane that are far away from all lattice points and will never come up in an encryption. This is a good basis for having an encryption scheme that is not co-range verifiable (or at least not co-range decidable), because if every point can be in the range of the encryption scheme, then the decision problem of decising the range is trivial. So, the question is whether deciding the range of a lattice-based encryption scheme is in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$.

In principal, it should be hard to decide whether one is given a close or a distant point. In particular, for the closest vector problem (CVP), the decision, search and optimization problems are all equivalent and $\mathcal{NP}$-hard [DKS98] for certain parameters and thus, for these parameters, it is unlikely that vertifying co-the range of a lattice-based encryption scheme is easy as this would imply that $\mathcal{NP} \subseteq \mathbf{co}\mathcal{AM}$ or even $\mathcal{NP} = \mathbf{co}\mathcal{NP}$, causing the the polynomial hierarchy to collapse.

Hence, there is reason to hope that lattice-based public-key encryption schemes are either not perfectly correct or not co-range verifiable or, in the best case, maybe even both and thus allow for building cryptography beyond the threshold of $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. Indeed, lattice problems often come with worst-case to average-case reductions to $\mathcal{NP}$-hard problems [Ajt96, Reg03, MR04, Reg05]. Unfortunately, parameter choice is a crucial issue in the area of lattice problems. For example, the parameters γ, for which γ-GapSVP (Gap version of the **S**hortest **V**ector **P**roblem in lattices) is $\mathcal{NP}$-hard, are not known to yield crypto-systems. In turn, known cryptosystems based on γ-GapSVP use parameters γ such that the corresponding problem γ-GapSVP is already in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ and thus unlikely to be $\mathcal{NP}$-hard [GG98, AR05]. As this holds for most schemes (see also [Ajt98, HR07, AD07, Cai98]), one might suspect inherent reasons for our failure to escape $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ when constructing key agreement schemes.

In contrast, our impossibility result proves that, at least, success is not inhibited by relativizing arguments. Our separation is based on the *generic oracle technique*, as introduced by Blum and Impagliazzo [BI87]. In particular, we introduce the notion of *mergeable oracle classes* and show that, relative to a generic oracle for those,

(i) $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is contained in $\mathcal{P}$.

In particular, we then define a specific mergeable oracle class and show that relative to a generic oracle for this class, infinitely often secure key agreement exists, i.e., there is an inifinite sequence of security parameters such that the key agreement is secure for those. Putting both results together, we yield an oracle relative to which

(i) $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is contained in $\mathcal{P}$.

(ii) key agreement exists that is infinitely often secure.

We now introduce generic oracles and then explain how to use them to prove our results.

7.3 Generic Oracles

Generic oracles have been introduced by Blum and Impagliazzo [BI87] to prove oracle separations in complexity theory such as the existence of an oracle such that $\mathcal{NP} \cap \mathbf{co}\mathcal{NP} \subseteq \mathcal{P}$, while $\mathcal{NP} \neq \mathcal{P}$, and, more generally, that this statement can be proved for any level of the polynomial hierarchy, that is, there is an oracle such that for any level n of the polynomial hierarchy, $\Delta_n = \Pi_n \cap \Sigma_n$, while the polynomial hierarchy does not collapse. See Chapter 2 for a review of the polynomial hierarchy and relativized complexity classes. For formal definitions of oracles and generic oracles, see Section 7.7. In the following, we will give a more informal description of these definitions. An oracle O is a function from $\{0,1\}^*$ to $\{0,1\}$ (or $\{0,1\}^*$, depending on the context), a partial oracle w is a partial function, i.e., a function that has not been defined everywhere, and a finite oracle is a partial oracle whose domain is finite. Two (partial) oracles are consistent, if they agree as functions (on the intersection of their domains). A partial oracle w is a prefix of another partial oracle v (or an oracle O), denoted $w \leq v$ (or $w \leq O$) if they are consistent and the domain of w is contained in the domain of v (the latter always holds for an oracle O that is defined everywhere). A class $\mathcal{C}$ of oracles is a set of oracles. A partial oracle w is in $\mathcal{C}$, if there is an oracle O in the class $\mathcal{C}$ such that $w \leq O$.

A generic oracle O for a class of oracles $\mathcal{C}$ is an oracle in the class $\mathcal{C}$ that has the following additional diagonalization properties, namely, for each oracle Turing Machine T with a one-bit output, one of the following two condition holds. Either (i) T^O returns 0 on input x_n for infinitely many values x_n, or (ii) there is a threshold $\Lambda \in \mathbb{N}$ and a finite oracle prefix $w \leq O$ such that, w forces T to return 1 on all x with $|x| > \Lambda$. Forcing means that all possible extensions of w have this property, formally, for all $O' \in \mathcal{C}$, it holds that if $w \leq O'$, then $T^{O'}(x) = 1$ for all x with $|x| > \Lambda$. This property follows from the definition of generic oracles, but, as we will see, we can also construct such an oracle with this property directly.

Lemma 4 (Generic Oracle)**.** *Let O be a generic oracle for a class of oracles $\mathcal{C}$. Then, for all Oracle Turing machines T that halt on all inputs with all oracles and produce one bit of output, one of the following conditions holds:*

(i) There exists an infinite sequence $(x_i)_i$ with $|x_0| < |x_1| < |x_2| < ...$ such that $T^O(x_i) = 0$.

(ii) There is a prefix $w \leq O$ and a threshold $\Lambda \in \mathbb{N}$ such that w forces T to be 1 on inputs x with $|x| \geq \Lambda$, i.e., for all x with $|x| \geq \Lambda$ and all extensions $w \leq O' \in \mathcal{C}$, it holds that $T^{O'}(x) = 1$.

Constructing such an oracle is surprisingly simple using an infinite procedure that defines the oracle bit-by-bit. It starts with an empty oracle w and then extends w slowly. The procedure enumerates all Turing Machines and make sure to revisit each machine T an infinite amout of times. When visiting T for the kth time, it either checks whether item (ii) is already satisfied for w. If not, it extends w (within the class $\mathcal{C}$) such that $T^w(x_i) = 0$ holds for at least k values x_i with $|x_0| < ... < |x_k|$; this is possible, if condition (ii) is not yet satisfied. Continuing this process infinitely yields an oracle O that for all machines T, satisfies either item (i) or item (ii).

In Section 7.7, we will introduce generic oracles, as defined by Blum and Impagliazzo [BI87], but throughout this thesis, it will be sufficient to work with the property stated in Lemma 4 and the fact that the above construction of a generic oracle relativizes. An oracle O that is generic for a class $\mathcal{C}$ *relative to an oracle* Π take as a base universe all oracle Turing Machines T^Π that have a second oracle slot for the oracle O. Formally, we yield the following.

Lemma 5 (Relativized Generic Oracles)**.** *Let Π be an oracle. Let O be an oracle that is generic for the class of oracles $\mathcal{C}$ relative to Π. Then, for all Oracle Turing machines T that have two oracle slots for O and Π and that halt on all inputs with all oracles O' (when Π is fixed) and produce one bit of output, one of the following conditions holds:*

(i) There exists an infinite sequence $(x_i)_i$ with $|x_0| < |x_1| < |x_2| < ...$ such that $T^{\Pi,O}(x_i) = 0$.

(ii) There is a prefix $w \leq O$ and a threshold $\Lambda \in \mathbb{N}$ such that w forces $T^{\Pi,\cdot}$ to be 1 on inputs x with $|x| \geq \Lambda$, i.e., for all x with $|x| \geq \Lambda$ and all extensions $w \leq O' \in \mathcal{C}$, it holds that $T^{\Pi,O'}(x) = 1$.

If Π is a computable function, then (i) and (ii) also hold for a generic oracle for $\mathcal{C}$.

One can construct oracles with the properties stated in Lemma 5 by using the same construction as before, this time going through all oracle Turing Machines $T^{\Pi,\cdot}$ with two oracle slots of which one is used for Π. Note that, if Π is a computable function such as a **PSPACE** oracle, then Lemma 4 already implies the properties of Lemma 5, because then, for every oracle Turing Machine $T^{\Pi,\cdot}$ with one open slot for O, there is an equivalent oracle Turing machine T' such that for all oracles O and for all inputs x, $T'^O(x) = T^{\Pi,O}(x)$. For readability, we hide a Π oracle under the rug in the following subsections. We will deal with relativized claims in Section 7.5.

7.3.1 Non-Existence of One-Way Permutations

We now use this property to derive statements that hold relative to a generic oracle. For example, let $\mathcal{C}$ be the class of all length-preserving functions. We will see that there is a huge class of events that have the property that they happen relative to a generic oracle O for $\mathcal{C}$ only if they happen with respect to *all* oracles from $\mathcal{C}$. For example, let G be a potential oracle construction of a one-way permutation. Let T_G be the oracle Turing Machine that on input x, returns 1, if and only if $G^{O'}(\{0,1\}^{|x|}) = \{0,1\}^{|x|}$, that is, if G

is a permutation on length $|x|$. Note that T_G is computable, namely, T_G goes over all inputs of length $|x|$ and checks, whether G is indeed a permutation on this input length. We will see that if G is permutation with respect to a generic oracle O for $\mathcal{C}$, then G is a permutation with respect to all oracles from $\mathcal{C}$. Being a permutation with respect to all oracles from $\mathcal{C}$ turns out to be a strong condition, which, maybe surprisingly, makes G^O invertible. In particular, we prove that relative to a generic oracle O, one-way permutations do not exist. In Section, we also prove the stronger statement that $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$ as well as that $\mathcal{NP}^O \not\subseteq \mathcal{P}^O$ and that infinitely often one-way functions exist relative to O.

Using the following two statements, we will prove that one-way permutations do not exist relative to a generic oracle O for the class $\mathcal{C}$ of length-preserving functions.

(i) If G implements a permutation relative to all oracles $O' \in \mathcal{C}$, then there is an polynomial-time Oracle Turing Machine $\mathcal{A}$ that for all oracles $O' \in \mathcal{C}$, inverts $G^{O'}$ with probability 1.

(ii) If G^O is a permutation relative to a generic oracle O, then there is an oracle algorithm G' such that $G'^O = G^O$ and such that G' implements a permutation with respect to all oracles $O' \in \mathcal{C}$.

Note that (i) means that there cannot even be permutations that are one-way infinitely many often. Putting (i) and (ii) together, we have that, relative to the generic oracle O, there are no secure one-way permutations. Towards contradiction, assume that G^O is a one-way permutation. By (ii), we have that G' is equivalent to G relative to O and that G' implements a permutation with respect to all oracles $O' \in \mathcal{C}$. Thus, by (i), we have that $\mathcal{A}^O$ inverts G'^O with probability 1 and therefore also G^O. Thus, assuming the above claims, we established the following proposition.

Proposition 1. *(informal) Let $\mathcal{C}$ be the class of all length-preserving functions. Let O be a generic oracle for $\mathcal{C}$. Then, relative to O, there are no secure one-way permutations.*

It remains to prove statements (i) and (ii), where Rudich [Rud88] establishes (i). We first use his result in a black-box way and then revisit it in Section 7.4. Let us now prove and state statement (ii).

Claim 1. *Let O be a generic oracle for the class $\mathcal{C}$ of all length-preserving oracles. If G^O is a polynomial-time computable permutation relative to O, then there exists a polynomial-time computable permutation G' such that $G'^O = G^O$ and such that G' implements a permutation with respect to all oracles $O' \in \mathcal{C}$.*

Proof. Let T_G be the oracle Turing Machine that on input x, returns 1, if and only if $G^{O'}(\{0,1\}^{|x|}) = \{0,1\}^{|x|}$, that is, if $G^{O'}$ is a permutation on length $|x|$. Let us consider the two conditions in Lemma 4. As G^O is a permutation, condition (i) of Lemma 4 is not satisfied, i.e., there are not infinitely many x_i such that $T_G^O(x_i) = 0$; actually, there is not even a single such input x_i. As Lemma 4 sais that one of the two conditions has to hold, we conclude that condition (ii) of Lemma 4 is true, i.e., there is a finite prefix w

of O that forces T to return 1 above a certain threshold Λ. We hardcode the behaviour of G^O up to length Λ into a construction G', and we also hardcode the prefix w into G', namely, G' runs G as a subroutine on values x with $\Lambda < |x|$; and whenever G makes a query to its oracle that is in w, then G' does not query its real oracle O', but instead, it returns the answer stored in w. G' is polynomial-time, because these operations take constant-time only. Moreoever, by construction, G' implements a permutation relative to any oracle $O' \in \mathcal{C}$ and is equivalent to G relative to O. □

In an analogous way, we will now prove that relative to a generic oracle O for $\mathcal{C}$, all problems $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O$ are contained in $\mathcal{P}^O$.

7.3.2 $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$

We consider the same class $\mathcal{C}$ of length-preserving functions with O being a generic oracle for this class. Before, T_G decided whether a candidate permutation was indeed a permutation (on length λ). In this section, T will decide whether a candidate language for $\mathcal{NP}^{O'} \cap \mathbf{co}\mathcal{NP}^{O'}$ is indeed in $\mathcal{NP}^{O'} \cap \mathbf{co}\mathcal{NP}^{O'}$ (on length λ). Namely, a candidate langue in $\mathcal{NP}^{O'} \cap \mathbf{co}\mathcal{NP}^{O'}$ consists of two non-deterministic oracle machines M and N. The language $\mathcal{L}_M^{O'}$ is defined as the set of x such that there is a witness z such that $M^{O'}(x, z) = 1$, and the language $\mathcal{L}_N^{O'}$ is defined as the set of x such that there is a witness z such that $N^{O'}(x, z) = 1$. For M and N to define a language in $\mathcal{NP}^{O'} \cap \mathbf{co}\mathcal{P}^{O'}$, we have to have that $\mathcal{L}_M^{O'}$ is the complement of $\mathcal{L}_N^{O'}$. The algorithm T, on input x, now tests whether this is indeed the case on length $|x|$, i.e., $T^{O'}$ returns 1 if and only if

$$\mathcal{L}_M^{O'} \cap \{0,1\}^{|x|} = \{0,1\}^{|x|} \setminus (\mathcal{L}_N^{O'} \cap \{0,1\}^{|x|}),$$

which, from now on, we call *complementary* machines relative to O' (on length λ). Note that T is a computable function, as a machine can go (in exponential time) through all the inputs of length $|x|$ and all the witnesses and thereby checks whether M and N are complementary on length λ.

Similarly to the case of one-way permutations, we will prove the following two statements.

(i) If $M^{O'}$ and $N^{O'}$ are complementary for all oracles $O' \in \mathcal{C}$, then there is a deterministic polynomial-time oracle Turing Machine $\mathcal{A}$ that for all oracles $O' \in \mathcal{C}$, makes only polynomially many queries to O' and decides $\mathcal{L}_M^{O'}$ (and thus also $\mathcal{L}_N^{O'}$).

(ii) If M^O and N^O are complementary for the generic oracle O, then there are machines M' and N' such that $\mathcal{L}_N^O = \mathcal{L}_{N'}^O$ and $\mathcal{L}_M^O = \mathcal{L}_{M'}^O$ and such that, for all oracles $O' \in \mathcal{C}$, M' and N' are complementary relative to O'.

Putting (i) and (ii) together, we have that, relative to the generic oracle O, $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$. $\mathcal{L}$ be a language in $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O$, that is, let M and N be two algorithms that are complementary relative to O such that $\mathcal{L} = \mathcal{L}_M^O$. By (ii), we have that M' and N' are equivalent to M and N relative to O and that M' and N' are complementary relative to all oracles $O' \in \mathcal{C}$. Thus, by (i), we have that $\mathcal{A}^O$ decides

$\mathcal{L}^O_{M'}$ and therefore also $\mathcal{L}^O_M = \mathcal{L}$. Hence, $\mathcal{L}$ is decidable in polynomial-time relative to O. Thus, assuming the two above claims, we established the following proposition.

Proposition 2. *Let $\mathcal{C}$ be the class of all length-preserving functions. Let O be a generic oracle for $\mathcal{C}$. Then, $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$.*

It remains to prove statements (i) and (ii). Blum and Impagliazzo [BI87] already establish item (i) and for now, we will use their result as a black-box. We will prove (ii) next, and later in Section 7.5, we will investigate how to implement their decision algorithm $\mathcal{A}$ in deterministic polynomial-time.

Claim 2. *Let O be a generic oracle for the class $\mathcal{C}$ of all length-preserving oracles. Let M and N be such that there is a polynomial p_M and p_N such that*

$$\mathcal{L}^O_M := \{x | \exists z : |z| \leq p_M(|x|) \text{ and } M^O(x; z) = 1\}$$

is the complement of

$$\mathcal{L}^O_N := \{x | \exists z : |z| \leq p_N(|x|) \text{ and } M^O(x; z) = 1\}.$$

Then, there exist two polynomial-time oracle algorithms M' and N' such that $\mathcal{L}^O_M = \mathcal{L}^O_{M'}$ and $\mathcal{L}^O_N = \mathcal{L}^O_{N'}$ and such that for all oracles $O' \in \mathcal{C}$, the algorithms M' and N' are complementary.

Proof. Let T be the oracle Turing Machine that on input x, returns 1, if and only if M and N are complementary relative to O' on input length $|x|$. Let us consider the two conditions in Lemma 4. As M and N are complementary relative to O, condition (i) is not satisfied, i.e., there are not infinitely many x_i such that $T^O(x_i) = 0$; actually, there is not even a single such input x_i. Thus, we know that condition (ii) of Lemma 4 holds, i.e., there is a finite prefix w of O that forces T to return 1 above a certain threshold Λ. We hardcode the behaviour of M^O up to length Λ into M' and the behaviour of N^O up to length Λ unto N', and we also hardcode the prefix w into M' and N', namely, M' runs M as a subroutine on witness z and values x with $\Lambda < |x|$; and whenever M' makes a query to its oracle that is in w, then M' does not query its real oracle O', but instead, it returns the answer stored in w. We define N' in the same way. M' and N' are polynomial-time, because these operations take constant-time only. Moreoever, by construction, M' and N' are complementary relative to any oracle $O' \in \mathcal{C}$, as the prefix w forced M and N to be complementary above certain threshold and as, below the threshold Λ, M' and N' are complementary, as M^O and N^O are. Moreover, relative to O, we have that M'^O behaves like M^O and N'^O behaves like N^O. □

The proof that $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$ is indeed very similar to the proof that relative to the generic oracle O, no one-way permutations exists. This is due to the the fact that one-way permutations already define a hard language in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$. Namely, if G is a permuation, then the language of all tuples (n, y) such that the nth bit of the pre-image of y under G is 1, is a language in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$. As a witness that (n, y) is in the language, we give the pre-image x of y, check whether indeed, $G(x) = y$ and whether the nth bit of x is 1. As a witness that (n, y) is not in the langauge, we also use the pre-iamge x of y.

7.3.3 Existence of i.o. One-Way Functions

We proved that relative to a generic oracle O for $\mathcal{C}$, the class $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O$ is contained in $\mathcal{P}^O$ and that thus, in particular, no one-way permutations exist relative to the generic oracle O. Note that this is also true relative to a **PSPACE** oracle, as, relative to a **PSPACE** oracle, $\mathcal{NP}^{\mathbf{PSPACE}} \subseteq \mathcal{P}^{\mathbf{PSPACE}}$ and thus also $\mathcal{NP}^{\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{\mathbf{PSPACE}} \subseteq \mathcal{P}^{\mathbf{PSPACE}}$. However, our goal is to prove that relative to some oracle O, $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$, while simultaneously, some cryptography such as one-way functions or key agreement exists, implying that $\mathcal{NP}^O \not\subseteq \mathcal{P}^O$.

We will now prove that relative to a generic oracle O for $\mathcal{C}$, there are efficiently computable functions that are one-way for infinitely many security parameters (i.o. one-way functions). Note that this implies that $\mathcal{NP}^O \not\subseteq \mathcal{P}^O$.

Lemma 6 (Existence of i.o. One-Way Functions)**.** *Let O be a generic oracle for the class $\mathcal{C}$ of length-preserving functions. Then, relative to O, i.o. one-way functions exists, namely the function $f^O : x \mapsto O(x)$ is i.o. one-way.*

Proof. We have to prove that there is an increasing sequence of security parameters λ_i such that for all polynomial-time oracle machines $\mathcal{A}$, the probability that $\mathcal{A}$ inverts f^O is negligible in λ_i. Actually, we are going to give a sequence of security parameters such that for all polynomial-time oracle machines $\mathcal{A}$, there exists an Λ such that for all $\lambda_i > \Lambda$, the success probability of $\mathcal{A}$ is smaller than $\epsilon(\lambda) := 2^{-\frac{\lambda}{2}}$.

We now consider an enumeration of all polynomial-time oracle inverters $(\mathcal{A}_k)_k$. Note that for any constant K, the following adversary $\mathcal{A}_K$ is also a polynomial-time oracle machine, as its running time is the sum of the running time of the first K adversaries plus a linear term in $K \cdot \lambda$.

Algorithm: $\mathcal{A}_K$
Input: $(1^\lambda, y)$
  From k from 1 to K do
    Run $x_i^* \leftarrow \mathcal{A}_i(1^\lambda, y)$
    If $f^O(x_i^*) = y$, return x^*.
  Return $0^{|y|}$.

Let T_K^O be the predicate that returns 1 on x, if

$$\mathrm{Prob}_{x \leftarrow \{0,1\}^\lambda}\left[\mathcal{A}_K(1^\lambda, f^O(x)) \rightarrow x' \in f^{O^{-1}}(y)\right]$$

and 0, else. Now, using Lemma 4, we note that the condition (ii) is not satisfiable for T_K. Namely, let $p_K(\lambda)$ be the running time of $\mathcal{A}_K$. Then, for a random length-preserving function, for large enough λ the success probability of $\mathcal{A}_K$ on input length λ is lower bounded by $\alpha p_K(\lambda) 2^{-\lambda}$ for some constant α.

Let w be a finite oracle, and let λ be large enough such that the partial oracle w has not fixed any answers to queries of length λ and such that $\mathcal{A}_K$'s success probability in inverting a random function of length λ is smaller than $2^{-\frac{\lambda}{2}}$. Extend w to v such that v agrees with w on w's domain, and such that $v(x) = 1^{|x|}$ for all x such that

– x is not in the domain of v, and

– $|x| \leq p_K(\lambda)$, and

– $|x| \neq \lambda$.

We now average out over all functions from $\{0,1\}^\lambda$ to $\{0,1\}^\lambda$ and obtain a single length-preserving function $C_\lambda : \{0,1\}^\lambda \to \{0,1\}^\lambda$ such that $\mathcal{A}_K$'s success probability in inverting C_λ is smaller than $2^{-\frac{\lambda}{2}}$. We now extend v by definition it to be equal to C_λ in length λ. For this λ, we have $T(0^\lambda) = 0$. We showed that any finite oracle w can be extended to an oracle that violates confition (ii) of Lemma 4, whence condition (i) must hold, i.e., there is an infinite sequence $(x_i)_i$ such that $T^O(x_i) = 0$, meaning that $\mathcal{A}_K$'s success probability to invert f^O on length $|x_i|$ is smaller than $2^\lambda 2$.

For each constant $K \in \mathbb{N}$, let us denote this sequence by $(x_i^K)_i$. Choose λ_1 to be equal to $|x_1^1|$. For $K > K'$ choose λ_K such that there is an element x_i^K in $(x_i^K)_i$ with $|x_i^K| > \lambda_{K'}$ (which exists, as $(x_i^K)_i$ is infinite and increasing).

For $(\lambda_K)_K$, we now have that for each polynomial-time adversary $\mathcal{A}_i$, its success probability in inverting f^O drops below $2^{\lambda_K} 2$ for $K > i$ and thus, for each polynomial-time adversary $\mathcal{A}_i$, the success probability is negligible. □

7.4 Mergeable Oracle Classes

We saw that generic oracles give us a natural way of achieving that $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}$, while $\mathcal{NP}^O \not\subseteq \mathcal{P}^O$. However, it is not clear that relative to a generic oracle for the class of length-preserving functions, there is a key agreement protocol that is secure for infinitely many security parameters, from now on called *i.o. key agreement protocol*. In the previous section, we had that O directly implements a function that is i.o. one-way. We would like to choose our class $\mathcal{C}$ is such a way that a generic oracle O for $\mathcal{C}$ implements a i.o. key agreement protocol. For Propositions 1 and 2, in item (ii), we had to prove that given two algorithms that were complementary relative to the generic oracle O, there were two algorithms that are equivalent relative to O but that have the additional property that they are complementary relative to all oracles O' in the class $\mathcal{C}$. Looking closely at the proof of item (ii) and at the proof of Proposition 2, we observe that we did not exploit any particularities of the class $\mathcal{C}$. Indeed, it holds for any class $\mathcal{C}$ that item (ii) is true and that, if additionally, item (i) is true for class $\mathcal{C}$, then the Proposition 2 hold, respectively. Thus, it is tempting to replace the class $\mathcal{C}$ of length-preserving functions by, say, the class of functions that implement key-agreement, that is, a function for Alice to generate her protocol messages, a function for Bob to compute his protocol messages as well as two key derivation functions, one for Alice and one for Bob. These functions implement a key agreement, if they satisfy the correctness property for key agreement. And we might want to choose to replace the classe of length-preserving functions by this 4-tuple of functions that implement key agreement.

However, we have not yet looked at the proof of item (i), and it turns out that its proof is not independent of the class $\mathcal{C}$. Namely, item (i) sais that if two algorithms M

and N are complementary relative to all oracles O' of $\mathcal{C}$, then the language $\mathcal{L}_M^{O'}$ is easy to decide relative to all oracles O' from $\mathcal{C}$. A weaker form of item (i) is that if an algorithm G implements a permutation with respect to all oracles O' from $\mathcal{C}$, then it is easy to invert relative to all oracles O' from $\mathcal{C}$. We will now reconsider the proofs by Rudich for one-way permutations and by Blum and Impagliazzo for $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ respectively, and we will see which property from $\mathcal{C}$ they rely on.

7.4.1 One-Way Permutations

Rudich proves in his thesis [Rud88] that one cannot build a one-way-permutation from a random oracle. We here present a simplified version of his proof establishing the following claim.

Claim 3. *Let $\mathcal{C}$ be the class of length-preserving oracles. If $M^{O'}$ and $N^{O'}$ are complementary for all oracles $O' \in \mathcal{C}$, then there is a deterministic polynomial-time oracle Turing Machine $\mathcal{A}$ that for all oracles $O' \in \mathcal{C}$, makes only polynomially many queries to O' and decides $\mathcal{L}_M^{O'}$ (and thus also $\mathcal{L}_N^{O'}$).*

Let G be an efficient, deterministic oracle machine such that for all oracles O' from the class $\mathcal{C}$ of length-preserving functions, the function $G^{O'}$ is a permutation. Then, we show that we can invert $G^{O'}$ with very few oracle queries. For now, we only consider query-complexity of the inverter. We will see in Section 7.5 how to make the inverter efficient. Assume, we want to invert $G^{O'}$ on a value y and assume for now that G only makes a single query to its oracle. As G is a permutation for all oracles O', we can find an input x, a query q and a response r such that G on input x queries q and returns y, if the oracle responds with r.[1] If there is only one such x, i.e., a value that can possibly (with the right oracle) be mapped to y, then we can invert $G^{O'}$ even without querying the oracle, as, regardless of the oracle's answers, there cannot be another pre-image than x. If there are two different such values x and x', then G must make the same query q on both inputs, x and x', as for if they make different queries q and q', then there is an oracle O' in $\mathcal{C}$ that comprises the appropriate answers to both queries. In this case, G maps both, x and x', to the same y, in contradiction to the assumption that $G^{O'}$ implements a permutation for each oracle O'.

Therefore, for all x that could potentially be mapped to the value y, the algorithm G makes the same query q. By querying the random oracle on only q, we can thus invert G. Carrying out this argument iteratively also yields an inverter for algorithms G that query its oracle several times. The number of queries of this inversion algorithm is quadratic in the number of queries asked by G. Therefore, if $G^{O'}$ implements a permutation for all oracles O, then G^O is easy to invert with polynomially many queries. Thus, we cannot build a one-way-permutation from a random oracle. Note that our inverter is not necessarily efficiently computable as the sampling step can only be carried out in non-deterministic polynomial-time. Indeed, in Section 7.5, we will add an **PSPACE**

[1] Sampling such a triple (x, q, r) is a problem in (unrelativized) $\mathcal{NP}$.

oracle to our world and see that all previous statements still hold. We prefer to keep this aspect under the rug for now.

Observe that the only place where we used $\mathcal{C}$ is when we argued that two evaluation pathes (pairs of queries and answers) can be *merged* into one oracle that is consistent with both pathes. We will see that this property is also needed to prove that, relative to a generic oracle O for class $\mathcal{C}$, $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$.

7.4.2 $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O$

Let M and N be two nondeterministic polynomial-time oracle machines such that for all oracles O' from the class of length-preserving functions $\mathcal{C}$, the $\mathcal{NP}$-languages $\mathcal{L}_M^{O'}$ and $\mathcal{L}_N^{O'}$ defined by $M^{O'}$ and $N^{O'}$ are complementary. In this case, Lemma **2.2** by Blum and Impagliazzo [BI87] tells us that relative to all $O' \in \mathcal{C}$, we can decide the language $\mathcal{L}_M^{O'}$ efficiently, i.e., with few oracle queries. Let us shortly summarize their approach that established the following claim.

Claim 4. *Let $\mathcal{C}$ be the class of all length-preserving oracles. If $M^{O'}$ and $N^{O'}$ are complementary for all oracles $O' \in \mathcal{C}$, then there is a deterministic polynomial-time oracle Turing Machine $\mathcal{A}$ that for all oracles $O' \in \mathcal{C}$, makes only polynomially many queries to O' and decides $\mathcal{L}_M^{O'}$ (and thus also $\mathcal{L}_N^{O'}$).*

For now assume that M and N only make a single query to O'. Our goal is to decide whether a given input x lies in $\mathcal{L}_M^{O'}$ or not. As $\mathcal{L}_M^{O'}$ and $\mathcal{L}_N^{O'}$ are complementary for all oracles from $\mathcal{C}$, there must be an input x, a witness z, a query q and a response r such that at least one of the machines on witness z and input x queries q and outputs 1, if the oracle responds with r. If such a potential triple (z, q, r) only exists for N, then x necessarily lies in $\mathcal{L}_N^{O'}$, the complement of $\mathcal{L}_M^{O'}$. If such a potential triple (z, q, r) only exists for M, then x lies in $\mathcal{L}_M^{O}$, the complement of $\mathcal{L}_N^{O'}$. If there exists a triple (z, q, r) for N and a triple (z', q', r') for M, then both must ask their oracle on the same query q as else, there would be an oracle $O' \in \mathcal{C}$ such that x is in both languages $\mathcal{L}_N^{O'}$ and $\mathcal{L}_M^{O'}$ in contradiction to the fact that the two machines are complementary relative to all $O' \in \mathcal{C}$. Therefore, all accepting pathes (triples (z, q, r)) on M and N necessarily ask their oracle on q. Thus, it suffices to learn the oracle's answer on query q to decide whether M or N has an accepting path.

Iterative application of this line of reasoning then yields a decision algorithm for algorithms M and N that query its oracle up to $n(|x|)$ times. As for the cryptography examples, the query complexity of the iterated decision algorithm is $(n(|x|))^2$. Also here, we used that two evaluation pathes could be merged into a single oracle from the class $\mathcal{C}$.

One subtlety that we hid under the carpet so far was whether the decision algorithm $\mathcal{A}^{O'}$ for $\mathcal{NP}^{O'} \cap \mathbf{co}\mathcal{NP}^{O'}$ and the inverter for the one-way permutation are efficiently implementable relative to O'. Again, the sampling of accepting pathes (z, q, r) can be done in non-deterministic polynomial-time, and in Section 7.5, we will add a oracle to indeed achieve an efficiently computable decision algorithm $\mathcal{A}$.

7.4.3 Mergeable Oracle Classes

The property that we used in both proofs was that two evaluation pathes can be merged into one oracle, i.e., that we can take a couple of query-answer pairs and have an oracle $O' \in \mathcal{C}$ that is consistent with all of them. A mergeable oracle class captures exactly this property.

Definition 35 (Mergeable Oracle Class)**.** *(informal) An oracle class $\mathcal{C}$ is called* mergeable*, if for any two polynomial-size finite oracles v and w in $\mathcal{C}$ the following holds:*

> *If v and w are consistent on their domain, then they are consistent in $\mathcal{C}$, i.e., there is an oracle $O' \in \mathcal{C}$ such that $w \cup v$ is a prefix of O'.*

Let us consider oracle classes that are not mergeable. For example, the partial oracles (q, y) and (q', y) with $q \neq q'$ cannot be merged into one permutation oracle, as the permutation would have a collision. Thus, the class $\mathcal{C}$ of permutation oracles in not mergeable.

Moreover, the class of perfectly correct key agreement protocols is not mergeable. Namely, if on transcript T, Alice derives key k, then Bob also has to derive key k. However, it turns out that if we allow for a small correctness error in the key agreement protocol, then we can define a class of oracles that is mergeable and implements a key agreement protocol that is correct with overwhelming probability. Thus, proving that there is an oracle relative to which i.o. key agreement is possible while $\mathcal{NP} \cap \mathbf{co}\mathcal{NP} \subseteq \mathcal{P}$ involves the following three steps:

(1.) Let $\mathcal{C}$ be a mergeable oracle $\mathcal{C}$ and let O be a generic oracle for $\mathcal{C}$. We prove that, relative to O, $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is easy, i.e., $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$.

(2.) We give a class $\mathcal{C}$ of (4-tuples of) oracles that satisfy the correctness requirements for key agreement and prove that $\mathcal{C}$ is mergeable. show that the class of all key agreement oracles is mergeable.

(3.) We argue that, a generic oracle O for $\mathcal{C}$ is i.o. secure key agreement.

The proof of (1.) mimics the proof of Lemma **2.2** by Blum and Impagliazzo [BI87] that we sketched in Section 7.4.2. We call (1.) the main Lemma (see Lemma 7), as it is of independent interest, as we will see later.

Lemma 7 (Main Lemma)**.** *(informal) Let $\mathcal{C}$ be a mergeable oracle class $\mathcal{C}$, and let O be a generic oracle for $\mathcal{C}$. Then, $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$.*

As the proofs of Proposition 1 and 2, the proof of Lemma 7 proceeds in two steps:

(i) If $M^{O'}$ and $N^{O'}$ are complementary for all oracles $O' \in \mathcal{C}$, then there is a deterministic polynomial-time oracle Turing Machine $\mathcal{A}$ such that for all oracles $O' \in \mathcal{C}$ the algorithm $\mathcal{A}^{O'}$, decides $\mathcal{L}_M^{O'}$ (and thus also $\mathcal{L}_N^{O'}$).

(ii) If M^O and N^O are complementary for the generic oracle O, then there are machines M' and N' such that $\mathcal{L}_N^O = \mathcal{L}_{N'}^O$ and $\mathcal{L}_M^O = \mathcal{L}_{M'}^O$ and such that, for all oracles $O' \in \mathcal{C}$, M' and N' are complementary relative to O'.

The proof of Claim 2 carries over to prove (ii) for arbitrary oracle classes $\mathcal{C}$, in particular for a mergeable oracle class. Moreover, the proof of Claim 4 equally carries over to prove (i) whenever a class of oracles is mergeable. As in the proof of Proposition 2, we can then put (i) and (ii) together and obtain the statement in Lemma 7. We start with two machines M and N that define a language in $\mathcal{NP}^O \cap \mathcal{NP}^O$. Using (ii), we can replace M and N by two machines that are equivalent relative to O and that are complementary with respect to all oracles from $\mathcal{C}$. Using (i), we then obtain a polynomial-time oracle algorithm that decides $\mathcal{L}_M^O$ (and thus $\mathcal{L}_N^O$).

7.5 Relativized Claims

Rudich's inverter in Subection 7.4.1 as well as the decision algorithm that we defined in Section 7.4.1 are only query-efficient. We will now see that we can actually turn them into efficient algorithms when we add a **PSPACE** oracle everywhere. However, we have to make sure that all claims still hold when adding a **PSPACE** oracle which is the goal of this section.

Claim 1. *(relativized) Let O be a generic oracle a class $\mathcal{C}$ of oracles, and let be a **PSPACE** oracle. If G^O is a polynomial-time computable permutation relative to O, then there exists a permutation G' such that $G'^{O,\mathbf{PSPACE}} = G^{O,\mathbf{PSPACE}}$ and such that G' is a polynomial-time computable permutation with respect to the oracles and all oracles $O' \in \mathcal{C}$.*

The proof of Claim 1 only exploits properties of relativized generic oracles as stated in Lemma 5 and thus still holds; the same holds for the following stronger claim and also for step (i) in the proof of our main lemma.

Claim 2. *(relativized) Let O be a generic oracle for a class $\mathcal{C}$ of oracles. Let be a oracle. Let M and N be such that there is a polynomial p_M and p_N such that*

$$\mathcal{L}_M^{O,\mathbf{PSPACE}} := \{x | \exists z : |z| \leq p_M(|x|) \text{ and } M^{O,\mathbf{PSPACE}}(x;z) = 1\}$$

is the complement of

$$\mathcal{L}_N^{O,\mathbf{PSPACE}} := \{x | \exists z : |z| \leq p_N(|x|) \text{ and } M^{O,\mathbf{PSPACE}}(x;z) = 1\}.$$

Then, there exist two polynomial-time oracle algorithms M' and N' such that $\mathcal{L}_M^{O,\mathbf{PSPACE}} = \mathcal{L}_{M'}^{O,\mathbf{PSPACE}}$ and $\mathcal{L}_N^{O,\mathbf{PSPACE}} = \mathcal{L}_{N'}^{O,\mathbf{PSPACE}}$ and such that for all oracles $O' \in \mathcal{C}$, the algorithms $M'^{O',\mathbf{PSPACE}}$ and $N'^{O',\mathbf{PSPACE}}$ are complementary.

Moreover, we also yield the following relativized claim.

Claim 4. *(relativized) Let $\mathcal{C}$ be a mergeable oracle class and let **PSPACE** be a **PSPACE** oracle. Let M and N be two polynomial-time machines and let p_M and p_N be two polynomials such that for all oracles $O' \in \mathcal{C}$,*

$$\mathcal{L}_M^{O',\mathbf{PSPACE}} := \{x | \exists z : |z| \leq p_M(|x|) \text{ and } M^{O,\mathbf{PSPACE}}(x;z) = 1\}$$

is the complement of

$$\mathcal{L}_N^{O',\boldsymbol{PSPACE}} := \{x | \exists z : |z| \leq p_N(|x|) \text{ and } M^{O,\boldsymbol{PSPACE}}(x;z) = 1\}.$$

Then, there exist a polynomial-time oracle algorithm $\mathcal{A}$ such that for all $O' \in \mathcal{C}$, $\mathcal{A}^{O',\boldsymbol{PSPACE}}$ decides $\mathcal{L}_{M'}^{O,\boldsymbol{PSPACE}}$.

Note that the relativized version of item (i) proved in Proposition 2 follows when taking $\mathcal{C}$ to be the class of length-preserving functions as the latter is mergeable. Another corollary is that, if $\mathcal{C}$ is the class of length-preserving functions and if $G^{O',\mathbf{PSPACE}}$ implements a permutation with respect to all $O \in \mathcal{C}$, then G is easy to invert relative to $O', \mathbf{PSPACE}$.

The proofs of Claim 4 or item (i) for Proposition 2 respectively only exploit the mergeability of $\mathcal{C}$ and the polynomial query complexity of polynomial-time algorithms. It thus also carries over in the presence if a **PSPACE** oracle.

Finally, putting the relativized version of Claim 4 and Claim 2 together, we obtain our the relativized version of our Main Lemma 7 and, as a corollary, also the relativized versions of Proposition 2 and Proposition 1 as the oracle classes that are considered in the latter two propositions are mergeable. as a corollary.

Lemma 7 (Main Lemma)**.** *(formal) Let $\mathcal{C}$ be a mergeable oracle class $\mathcal{C}$, and let O be a generic oracle for $\mathcal{C}$. Then, $\mathcal{NP}^{O,\boldsymbol{PSPACE}} \cap \boldsymbol{co}\mathcal{NP}^{O,\boldsymbol{PSPACE}} \subseteq \mathcal{P}^{O,\boldsymbol{PSPACE}}$.*

Proposition 2. *(formal) Let $\mathcal{C}$ be a mergeable oracle class. Let O be a generic oracle for $\mathcal{C}$. Then, relative to O and a oracle, there are no secure one-way permutations.*

Proposition 1. *(formal) Let $\mathcal{C}$ be a mergeable oracle class. Let O be a generic oracle for $\mathcal{C}$. Then, relative to O and a oracle, it holds that languages in $\mathcal{NP} \cap \boldsymbol{co}\mathcal{NP}$ are decidable in deterministic polynomial-time, formally*

$$\mathcal{NP}^{O,\boldsymbol{PSPACE}} \cap \boldsymbol{co}\mathcal{NP}^{O,\boldsymbol{PSPACE}} \subseteq \mathcal{P}^{O,\boldsymbol{PSPACE}}.$$

Finally, also the proof for the existence of i.o. one-way functions carries over to the relativized setting, as the proof of the unrelativized Lemma 6 considers security against computable adversaries whose query-complexity is polynomial. Hence, the set of adversaries remain unchanged by adding a **PSPACE** oracle.

Lemma 6 (Existence of i.o. One-Way Functions)**.** *(relativized) Let O be a generic oracle for the class $\mathcal{C}$ of length-preserving functions. Then, relative to O and a oracle, i.o. one-way functions exists, namely the function $f^O : x \mapsto O(x)$ is i.o. one-way.*

7.6 Further Complexity Measures and $\mathcal{AM}^O \cap \mathbf{co}\mathcal{AM}^O$

In this section, we review our results for $\mathcal{AM}^O \cap \mathbf{co}\mathcal{AM}^O$ and discuss the complexity measures that we use as tools in our proofs.

A function f is regular if for all y, y' with the same length $|y| = |y'|$, the pre-image space of y and y' is of the same size, $|f^{-1}(y)| = |f^{-1}(y')|$. We prove the existence of an oracle O relative to which languages in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ are decidable in deterministic polynomial-time while at the same time, regular functions exist that are one-way infinitely many often, so-called regular i.o. one-way functions. I.e., relative to our oracle O,

- $\mathcal{AM}^O \cap \mathbf{co}\mathcal{AM}^O \subseteq \mathcal{P}^O$ and
- regular i.o. one-way functions exist.

Note that Akavia et al. [AGGM06] considered a bigger class of functions, where the pre-image size is not the same for all y of the same length. Instead, they only asked for the pre-image size to be verifiable given an additional witness. Note that regular functions are a particular subclass of these and thus, our result holds a fortiori for the class considered by Akavia et al. [AGGM06]. Moreover, note that there is a second Theorem in Akavia et al. [AGGM06] (which does not suffer from the aforementioned problem) that established that a regular one-way function f implies hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$, if for each y, the pre-image size $|f^{-1}(y)|$ is polynomially in the length $|y|$. Indeed, we only consider regular functions with exponential-size domain and thus exhibit that the second Theorem by Akavia et al. might be a tight result.

Firstly, using Main Lemma 7, we obtain an oracle O, relative to which

- $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$ and
- regular i.o. one-way functions exist.

Namely, for a constant c, let $\mathcal{C}_c$ be the class of regular functions (oracles) O' such that $|O'^{-1}(y)| = 2^{|y|/c}$. This class is mergeable, as a polynomial number of values cannot inhibit the regularity property. Thus, Lemma 7 yields that relative to a generic oracle O for $\mathcal{C}_c$, we have that $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O \subseteq \mathcal{P}^O$. Moreover, as before, we will see that O is i.o. one-way (and regular by definition). We prove this formally in Theorem 8.

Now, Theorem 10 extends this result to $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$, i.e., relative to a generic oracle O for $\mathcal{C}_c$, i.o. one-way functions exist while $\mathcal{AM}^O \cap \mathbf{co}\mathcal{AM}^O \subseteq \mathcal{P}^O$. Although $\mathcal{AM}$ is merely a randomized version of $\mathcal{NP}$, extending the result is not immediate. Let us go back to the proof that each language $\mathcal{L}^O$ in $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O$ also lies in $\mathcal{P}^O$ and introduce the notion of certificate complexity for oracles.

A finite prefix w of an oracle O *certifies* that x is in $\mathcal{L}^O$, if for all possible extensions O' of w, it holds that x is in $\mathcal{L}^O$. A certificate w in O certifies that x is not in $\mathcal{L}^O$, if for all possible extensions O' of w, it holds that x is not in $\mathcal{L}^O$. Let us see why languages in $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O$ have polynomial-size certificates. Let M and N be the machines associated with a language $\mathcal{L}^O \in \mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}^O$. For each x, we either have a witness z such that $M^O(z, x) = 1$ or that $N^O(z, x) = 1$. Taking the partial oracle w of all queries and answers made along the path defined by z and x yield a certificate that x is (not) in $\mathcal{L}^O$. Moreover, our Claim 4 (Lemma **2.5** in Nisan [Nis91] and Theorem **2.3** in Blum and Impagliazzo [BI87]) can also be phrased in terms of certificate complexity. We showed that two (polynomial-size) certificates w and w' one for being in $\mathcal{L}^O$, one for being not in $\mathcal{L}^O$ have to intersect in a common query. Iterative application of this argument leads

to an algorithm that decides $\mathcal{L}^O$ using only polynomially many queries. Thus, we can phrase Claim 4 as proving that languages with small certificate complexity are in $\mathcal{L}^O$. And in particular, this applies to $\mathcal{NP}^O \cap \mathbf{co}\mathcal{NP}$.

If we can prove that languages in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ also have small certificate complexity, then we also yield that languages in $\mathcal{AM}^O \cap \mathbf{co}\mathcal{AM}^O$ can be decided in $\mathcal{P}^O$. The main part of Section 7.10 is devoted to proving that, relative to the class $\mathcal{C}_c$ of regular functions, $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ has a small certificate complexity.

Towards this goal, we also build on another complexity measure, the so-called block-sensitivity. Lemma **2.4** in Nisan [Nis91] states that, whenever a language has a low block sensitivity, then it also has a low certificate complexity. Thus, we revert to proving that languages in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ have low block-sensitivity.

Overall, we have three proof steps:

(1) Small oracle certificate complexity of M implies that the language $\mathcal{L}_M^O$ is easy to decide, relative to a generic oracle. (Lemma **2.5**, Nisan [Nis91]/Theorem **2.3**, Blum, Impagliazzo [BI87])

(2) Low block sensitivity of M implies small certificate complexity of M. (Lemma **2.4**, Nisan [Nis91])

(3) Languages in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ have low block sensitivity. (This work)

Our proof of step (3.) is inspired by Nisan's proof that languages in $\mathcal{BPP}$ have low block sensitivity.

Nisan's notions of certificate complexity and block sensitivity both refer to the class of all oracles while we refer to the class $\mathcal{C}_c$ of regular function oracles. We thus have to adapt step (1.), (2.) and (3.) for $\mathcal{C}_c$.

Let us thus consider step (2). A string x and a language $\mathcal{L}^O$ are said to be sensitive to a block (part) of the oracle O, if flipping these bits in the oracle changes whether x belongs to $\mathcal{L}^O$. The block-sensitivity then captures an upper bound on the number of disjoint blocks, a language is sensitive to. In other words, the block sensitivity of $\mathcal{L}_M^O$ on (x, O) is the number of disjoint sets S_i such that $x \in \mathcal{L}_M^O$ if and only if $x \notin \mathcal{L}_M^{O_{(S_i)}}$, where $O_{(S_i)}$ is equal to O except for queries in S_i, where the output of the oracle is flipped.[2] Nisan's proof for step (2) does not carry over immediately.

The core argument in his proof is to pick, iteratively, disjoint, minimal blocks that the function is sensitive to. He then argues that *after* flipping all of the bits of one of these blocks, the language has to be sensitive to each bit in the block, as the block was chosen to be minimal. Therefore, he argues, each of the bits in the block has to be contained in a potential certificate.

Implicitly, this proof uses that flipping one bit of an oracle yields a valid oracle again. However, when O implements a regular function and one switches, say, $O(x) = 0|y$ to $O(x) = 1|y$, then the modified oracle does not implement a regular function anymore and is thus an oracle "outside the class". Yet, we will show that the argument applies

[2] Without loss of generality, we consider one-bit output oracles that encode regular functions in a unique way.

to flipping small groups of bits such that the mofication transforms a regular function into regular function.

7.7 Generic Oracles and Forcing

We now recall the formal definitions of generic oracles as in [BI87] and explain how they relate to what we already mentioned.

Definition 36 (Oracle)**.** *An* oracle *O is a total, deterministic function from $\{0,1\}^*$ to $\{0,1\}$. A* partial oracle *v is a partial deterministic function from a subset of $\{0,1\}^*$ to $\{0,1\}$. In particular, an oracle is also a partial oracle. A* partial oracle *is called a* finite *oracle, if its domain is finite. Two partial oracles v and w are* consistent*, if they agree as functions on the intersection of their domain. If, furthermore, the domain of v is a subset of the domain of w, then we say that w extends v, or that v is a prefix of w, denoted $v \leq w$. A* class *of oracles $\mathcal{C}$ is a set of oracles. By abuse of notation, we write $v \in \mathcal{C}$ for a partial oracle v, if there is a $O \in \mathcal{C}$ such that $v \leq O$. We write $(q,r) \in O$, if $O(q) = r$.*

An appropriate encoding transforms oracles with several bits of output into oracles with a single bit output, and vice versa. For cryptography, it is more convenient to define oracles as functions with several bits of output, while for certain complexity-theoretic considerations, it is preferable to think of oracles as having only a single bit of output. Therefore, we will use both notions throughout this paper, depending on which one is more convenient for the matter under consideration. Oracles can be imagined as infinitely long look-up tables, which, in turn, can be considered as infinitely long bitstrings. A finite oracle w is then a word over the alphabet $\{0,1\}$ or the alphabet $\{0,1\} \cup \{[]\}$ which has a special sign for undefined oracle values. Sometimes, it will be convenient to think of an oracle w as an input to a machine rather than an oracle (where the machine only reads parts of its input). In particular, we will manipulate the bitstring w by flipping a certain number of bits to see whether, e.g., a certain oracle machine changes its answer under these bit flips. Note that other inputs to the machine will usually be denoted by x, y and z. In particular, z usually denotes a witness, if we deal with non-deterministic machines.

We give our definitions with respect to a class $\mathcal{C}$ of oracles. The standard notions of genericity or forcing are obtained when using the class of all oracles from $\{0,1\}^*$ to $\{0,1\}$.

Definition 37 (Forcing)**.** *Let T, M and N be an Oracle Turing Machines (OTM) that halt on all inputs with all oracles in a class $\mathcal{C}$, we say that a finite oracle w* forces *M and N to be* different*, if there is an x such that $M^w(x)$ and $N^w(x)$ are both defined and $M^w(x) \neq N^w(x)$. More generally, we say that w forces T to be wrong, if there is an x such that $T^w(x)$ is defined and such that $T^w(x) = 0$. We say that w $\mathcal{C}$-*forces *T to be true, if no extension $w \in \mathcal{C}$ forces T to be wrong.*

Define $T_{M,N}$ as the oracle algorithm that, on input x with oracle O runs $M^O(x)$ and $N^O(x)$ and returns 1 if and only if $M^O(x) \neq N^O(x)$. Then, a finite oracle w forces $T_{M,N}$ to be wrong if and only if w forces M and N to be different. And a finite oracle w $\mathcal{C}$-forces $T_{M,N}$ to be true if and only if w forces M and N to be the same. We sometimes omit the class $\mathcal{C}$ when it is clear from context.

The following definition of genericity is a weak version of the one introduced by Blum and Impagliazzo [BI87]. They called this notion 1-genericity.

Definition 38 (Generic Oracle)**.** *A set D of finite oracles is* $\mathcal{C}$-dense *for a class of oracles $\mathcal{C}$, if every finite oracle in $\mathcal{C}$ has an extension to a finite oracle in D. Let $C = \{D_i | i \in \mathbb{N}\}$ be a countable collection of $\mathcal{C}$-dense sets of finite oracles. An oracle O is $(C,\mathcal{C})$-generic, if for every $i \in \mathbb{N}$, there is a finite oracle $w \in D_i$ such that w is a finite prefix $w \leq O$.*

We say that O is $\mathcal{C}$-generic, if it is $(C,\mathcal{C})$-generic for the following countable collection of $\mathcal{C}$-dense sets:

$$\{D_T | T \text{ OTM that halt on all inputs with all oracles in } \mathcal{C}\}$$

where D_M is defined as

$$\begin{aligned} D_T := & \\ & \{v | v \text{ finite, } v \in \mathcal{C}, v \text{ forces } T \text{ to be wrong.}\} \\ \cup & \{v | v \text{ finite, } v \in \mathcal{C}, v\,\mathcal{C}-\text{forces } T \text{ to be true.}\} \end{aligned}$$

Let us see why indeed, the set D_T is $\mathcal{C}$-dense. Let $w \leq O \in \mathcal{C}$ be a finite oracle in the class $\mathcal{C}$. Either w $\mathcal{C}$-forces T to be true, or there is an extension $w \leq v \in \mathcal{C}$ such that v forces T to be wrong. In the first case, w itself is in D_T; in the second case, w can be extended to a finite oracle v in D_T. Thus, D_T is dense. Our generic oracles diagonalize against relatively simple properties. Indeed, Blum and Impagliazzo [BI87] diagonalized against all dense arithmetic sets thus also including sets of higher type. We sketched the proof of the existence in Section 7.3, a proof can also be found in [BI87]. We now prove that genericity implies Lemma 4. Note that Lemma 5 then follows analogously.

Lemma 4 (Generic Oracle)**.** *Let O be a generic oracle for a class of oracles $\mathcal{C}$. Then, for all Oracle Turing machines T that halt on all inputs with all oracles and produce one bit of output, one of the following conditions holds:*

(i) There exists an infinite sequence $(x_i)_i$ with $|x_0| < |x_1| < |x_2| < \ldots$ such that $T^O(x_i) = 0$.

(ii) There is a prefix $w \leq O$ and a threshold $\Lambda \in \mathbb{N}$ such that w forces T to be 1 on inputs x with $|x| \geq \Lambda$, i.e., for all x with $|x| \geq \Lambda$ and all extensions $w \leq O' \in \mathcal{C}$, it holds that $T^{O'}(x) = 1$.

Proof. Let T be an OTM that halts on all inputs with all oracles. Based on T, we now define a countable number T_1, T_2, T_3,... of machines that do the following: $T_k^O(x)$ checks whether there are $k-1$ different values x' with $|x'| < |x|$ such that $T^O(x') = 0$. If so,

$T_k^O(x) := T^O(x)$. If not, then $T_k^O(x) := 1$. We now proceed by case distinction. Case I: For each $k \in \mathbb{N}$, there is an x such that $T_k^O(x) = 0$. Then, by definition of T_k^O, there are infinitely many values x such that T^O returns 0 on them and thus condition (i) is satisfied. Case II: There is a k such that T_k^O is constantly 1. Let us consider the set D_{T_k} as in the definition of generic oracles. By genericity of O, there is a finite prefix $w \subseteq O$ that either has the property that it forces T_k to be wrong, or it $\mathcal{C}$-forces T_k to be true. As T_k^O is constantly 1, the first condition cannot be satisfied. Thus, there has to be a finite prefix $w \leq O$ that $\mathcal{C}$-forces T_k to be true, that is, there is a prefix w such that for all extensions $w \leq O'\mathcal{C}$, $T^{O'}$ returns 0 on at most k values. Note that w does not yet define the threshold Λ that we are looking for, as Λ might depend on O'. Let $X \subseteq \{0,1\}^*$ be the set of values that T^O returns 0 on. Let v be the finite prefix of O such that T^v is well-defined on X. Let Λ be a natural number that is greater than the length of the longest string in X. Let $v \cup w$ be the smallest finite prefix of O that covers the domain of both, v and w. Then, $v \cup w$ forces T to return 1 on all values x with $|x| \geq \Lambda$ as desired. □

We now define what it means for an oracle machine to define an $\mathcal{NP}$-language or an $\mathcal{AM}$-language for *all oralces* from some class. Note that this is as stronger requirement than defining an $\mathcal{NP}$-language or $\mathcal{AM}$-language relative to one specific oracle.

Definition 39. *We say that a deterministic polynomial-time oracle machine M defines an $\mathcal{AM}$-language $\mathcal{L}^O$ for all oracles $O \in \mathcal{C}$, if there are fixed polynomials p, q and R such that for all x and for all $O \in \mathcal{C}$, it holds that either*

$$\mathrm{Prob}_{r \leftarrow \{0,1\}^{R(|x|)}} \left[\exists z_r,\ |z_r| \leq q(|x|),\ M^O(x, r; z_r) = 1 \right] \geq 1 - 2^{-|x|}$$

or

$$\mathrm{Prob}_{r \leftarrow \{0,1\}^{R(|x|)}} \left[\exists z_r,\ |z_r| \leq q(|x|),\ M^O(x, r; z_r) = 1 \right] \leq 2^{-|x|},$$

where in all cases, M asks at most $p(|x|)$ queries to its oracle O. The $\mathcal{AM}$-language $\mathcal{L}^O$ consists of all those x, for which the above probability is greater than $1 - 2^{-|x|}$. For any nondeterministic polynomial-time OTM M, the $\mathcal{NP}$-language $\mathcal{L}^O$ is defined as

$$\{x | \exists z :\ |z| \leq q(|x|) \text{ and } M^O(x; z) = 1\}.$$

If M is not clear from context, this notation can be augmented to $\mathcal{L}_M^O$.

7.8 The Main Technical Lemma

We now turn to defining mergeable classes and showing that relative to an oracle O that is generic for a mergeable class, all problems in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ are easy in terms of query-complexity and thus, using relativization and adding a **PSPACE** oracle, we also have that $\mathcal{NP}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O,\mathbf{PSPACE}} \subseteq \mathcal{P}^{O,\mathbf{PSPACE}}$. A useful notation for merging will be $v \cup w$ as the smallest finite oracle that extends the two consistent finite oracles v and w.

Definition 40 ($\mathcal{C}$-Consistent Finite Oracles)**.** *Two consistent finite oracles v and w are called* $\mathcal{C}$-consistent*, if there is an oracle $O \in \mathcal{C}$ such that $v \cup w \leq O$.*

For any partial oracle, we denote by $v^{<n}$ and v^{n} the restricted oracle v for input length smaller than n or equal to n, respectively. We similarly make use of $\leq$, $\geq$ and $<$. By $|v|$, we denote the size of the support of v. An oracle class is mergeable, if any polynomial-size (or even slightly exponential-size) oracles v and w that are consistent can be merged into an oracle in $\mathcal{C}$, i.e., v and w are $\mathcal{C}$-consistent.

Definition 35 (Mergeable Oracle Class)**.** *An oracle class $\mathcal{C}$ is called* mergeable*, if there is a threshold $\nu(n) = 2^{n/c}$ for some constant $c \geq 1$ such that the following holds:*

> *Let v and w be two finite oracles in $\mathcal{C}$ such that w and v are consistent, $w^{<n}$ and $v^{<n}$ are consistent in $\mathcal{C}$ and such that $|w^{\geq n} \cup v^{\geq n}| \leq \nu(n)$. Then, v and w are consistent in $\mathcal{C}$.*

We now state our main Lemma.

Lemma 7. *[Main Lemma] Let O be a generic oracle for a mergeable oracle class $\mathcal{C}$ and let* ***PSPACE*** *be a* ***PSPACE*** *oracle. Then, $\mathcal{NP}^{O,\boldsymbol{PSPACE}} \cap \boldsymbol{co}\mathcal{NP}^{O,\boldsymbol{PSPACE}} \subseteq \mathcal{P}^{O,\boldsymbol{PSPACE}}$.*

Proof. The proof follows the outline given in the proof of Lemma 7 in Section 7.3; we prove the following two items.

(i) If two polynomial-time oracle Turing machines $M^{O',\mathbf{PSPACE}}$ and $N^{O',\mathbf{PSPACE}}$ define a language in $\mathcal{NP}^{O',\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O',\mathbf{PSPACE}}$ for all oracles $O' \in \mathcal{C}$, then there is a deterministic polynomial-time oracle Turing Machine $\mathcal{A}$ such that for all oracles $O' \in \mathcal{C}$ the algorithm $\mathcal{A}^{O',\mathbf{PSPACE}}$ decides $\mathcal{L}_M^{O'}$ (and thus also $\mathcal{L}_N^{O'}$).

(ii) If $M^{O,\mathbf{PSPACE}}$ and $N^{O,\mathbf{PSPACE}}$ define a language in $\mathcal{NP}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O,\mathbf{PSPACE}}$ for the generic oracle O then there are machines M' and N' such that $\mathcal{L}_N^{O,\mathbf{PSPACE}} = \mathcal{L}_{N'}^{O,\mathbf{PSPACE}}$ and $\mathcal{L}_M^{O,\mathbf{PSPACE}} = \mathcal{L}_{M'}^{O,\mathbf{PSPACE}}$ and such that, for all oracles $O' \in \mathcal{C}$, $M'^{O,\mathbf{PSPACE}}$ and $N'^{O,\mathbf{PSPACE}}$ define a language in $\mathcal{NP}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O,\mathbf{PSPACE}}$.

Let us see how we can derive the claim from these two items. Let (M, N) be a pair of machines that, for the generic oracle O, defines a language $\mathcal{L}_M^O$ in $\mathcal{NP}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O,\mathbf{PSPACE}}$. Let us consider the two machines M' and N' defined by (ii). For those two machines, (i) guarantees a deterministic polynomial-time decision algorithm $\mathcal{A}$ such that $\mathcal{A}^{O',\mathbf{PSPACE}}$ decides $\mathcal{L}_M^{O'}$ for all $O' \in \mathcal{C}$. In particular, $\mathcal{A}^{O,\mathbf{PSPACE}}$ decides $\mathcal{L}_{M'}^O = \mathcal{L}_M^O$ proving that $\mathcal{L}_M^O$ is in $\mathcal{P}^{O,\mathbf{PSPACE}}$.

The proof of item (ii) is analogous to Claim 2, as the proof works for all classes $\mathcal{C}$. In turn, for the proof of (i), we will use the mergeability condition. Let M and N be two machines that are complemetary with respect to all oracles $O' \in \mathcal{C}$, that is, $M^{O',\mathbf{PSPACE}}$ and $N^{O',\mathbf{PSPACE}}$ define a language in $\mathcal{NP}^{O',\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O',\mathbf{PSPACE}}$ for all $O' \in \mathcal{C}$.

We construct a deterministic polynomial-time algorithm $\mathcal{A}$ that decides $\mathcal{L}_M^O$. Let $p(|x|)$ be a polynomial bound on the number of queries that M and N make on (x, z) for a witness z of suitable length. Then, there is a threshold $N_p(|x|)$ such that $(p(|x|))^2 + p(|x|) \leq \nu(n)$ for all $n \geq N_p(|x|)$ but not for $n < N_p(|x|)$. The function $N_p(|x|)$ is efficiently computable.

On input x, the algorithm $\mathcal{A}$ queries the oracle to obtain all values in $O^{<N_p(|x|)}$. These are polynomially many queries, as $2^{N_p(|x|)} \leq (p(|x|))^c$. From now on, we denote by u the prefix of O which $\mathcal{A}$ already queried. For a non-deterministic polynomial-time OTM M, let the set $\mathtt{qry}_M(x, z, v)$ denote the set of queries that M makes on input x with witness z and oracles $v, \mathbf{PSPACE}$. If v is defined on all in M queries, then the behaviour of $M^{v,\mathbf{PSPACE}}(x, z)$ on input x with witness z is entirely specified. We thus consider a pair (z, v) of a witness z together with a set of matching query-response-pairs v (a finite oracle) as an *evaluation path* of M. We say that $M^{v,\mathbf{PSPACE}}$ has an accepting path on x, if there is a witness z such that $M^{v,\mathbf{PSPACE}}(x; z)$ is defined and returns 1. The algorithm $\mathcal{A}$ will iteratively sample accepting paths of M. As we are given a oracle, on a ground universe, $\mathcal{NP}^{\mathbf{PSPACE}} = \mathcal{P}^{\mathbf{PSPACE}}$ and thus, the sampling can be done in polynomial-time.

Algorithm: $\mathcal{A}$
Input: x
Output: $\chi_{\mathcal{L}^{O'}_M}(x)$
Set $u := O'^{\leq \max\{N_p(|x|),k\}}$ and $i_0 := -1$.
For i from 1 to $p(|x|)$ do:
If possible, find a witness z and an extension $v \in \mathcal{C}$ of u
such that $M^{v,\mathbf{PSPACE}}(x; z)$ accepts. Query $\mathtt{qry}_M(x, z, v)$ to O and update u.
If finding such a path was possible, define $i_0 := i$.
Search for a witness z such that $M^{u,\mathbf{PSPACE}}(x; z)$ is defined and accepts.
(1.) Return 1, if such a witness is found.
(2.) Return 0, if $i_0 < p(|x|)$.
Search for a witness z such that $N^{u,\mathbf{PSPACE}}(x; z)$ is defined and accepts.
(3.) Return 0, if such a witness is found.
(4.) Else, return 1.

We now analyse the correctness of the algorithm.

Case (1.) If we find an accepting path for x on $M^{u,\mathbf{PSPACE}}$, then there is also an accepting path on $M^{O',\mathbf{PSPACE}}$ and thus, x is in $\mathcal{L}^{O'}_M$.

Case (2.) If there is no accepting path for x on $M^{u,\mathbf{PSPACE}}$ and if $i_0 < p(|x|)$, then there is no extension $v \in \mathcal{C}$ of u such that $M^{v,\mathbf{PSPACE}}$ has an accepting path. In particular, the machine $M^{O',\mathbf{PSPACE}}$ does not have an accepting path and thus, $x \notin \mathcal{L}^{O'}_M$.

Case (3.) If we find an accepting path for x on $N^{u,\mathbf{PSPACE}}$, then there is also an accepting path on $N^{O',\mathbf{PSPACE}}$ and thus, x is in $\mathcal{L}^{O'}_N$, the complement of $\mathcal{L}^{O'}_M$.

Case (4.) If $i = p(x)$, then we found $p(x)$ accepting paths in the first loop. Assume that there is neither an accepting path for x on $M^{u,\mathbf{PSPACE}}$, nor on N^u. We will prove that then, there must be an accepting path on $M^{O'}$.

We start with several observations. As $\mathcal{C}$ is a mergeable class, the queries along an accepting path of M on x has to share a query with each accepting path of N on x, and this query must lie in $O'^{>N_p(x)}$, as both paths are consistent with $O'^{\leq N_p(x)}$: If the accepting paths of M and N on x did not share a query, there would be a partial oracle u_M and a witness z such that $M^{u_M,\mathbf{PSPACE}}(x;z)$ accepts and a partial oracle u_N and a witness z' such that $N^{u_N}(x,z')$ accepts as well. As disjoint the partial oracles u_N and u_M are mergeable to a partial oracle $u_N \cup u_M \in \mathcal{C}$—an extension of u—we yield a contradiction to the fact that M and N are complementary with respect to all oracle from $\mathcal{C}$.

Therefore, an accepting path on M for x covers at least one query along an accepting path of N for x. Applying the argument iteratively yields that all queries of all accepting paths of N have been asked within the $p(|x|)$ paths of M that we queried. Here, we use that, as long as $|u_M^{>N_p(|x|)}|$ is smaller than $(p(|x|))^2$, then $|u_M^{>N_p(|x|)} \cup u_N^{>N_p(|x|)}|$ is smaller than $(p(|x|))^2 + p(|x|)$.

Thus, if there had been an accepting path on $N^{O',\mathbf{PSPACE}}$, the algorithm would have discovered it already in (3.). As this is not the case, there are no accepting paths on $N^{O',\mathbf{PSPACE}}$ and thus, there must be an accepting path on $M^{O',\mathbf{PSPACE}}$, as $M^{O',\mathbf{PSPACE}}$ and $N^{O',\mathbf{PSPACE}}$ are complementary.

Runtime Analysis The query-complexity of $\mathcal{A}$ is polynomial. Moreover, finding accepting paths on $M^{\cdot,\mathbf{PSPACE}}$ and $N^{\cdot,\mathbf{PSPACE}}$ is a problem in $\mathcal{NP}^{\mathbf{PSPACE}}$ and thus efficiently computable, as $\mathcal{NP}^{\mathbf{PSPACE}} \subseteq \mathcal{P}^{\mathbf{PSPACE}}$. Thus, the algorithm $\mathcal{A}$ runs in deterministic polynomial-time relative to O' and **PSPACE**. □

For regular (one-way) functions (see Section 7.10), the previous Lemma 7 is general enough to be applied to the class of regular one-way functions—using the lemma, we will yield that relative to a generic oracle for this class, regular one-way functions exist, while $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ collapses to $\mathcal{NP}$. In contrast, for key agreement (see Section 7.9), we have to relax the conditions for mergeable classes. In particular, we only want to require a mergeable oracle class to allow gluing of pairs of finite oracles v and w of the right form.

Definition 41 (Closure Operator)**.** *A* closure condition *is an efficiently computable map* Γ *that maps all finite oracles* v *to an extended domain of* v*. A finite oracle* v *is* closed*, if* $\Gamma(v)$ *is equal to the domain of* v*. We also require from* Γ *to be local, i.e., each closure rule only depends on a single query-response-pair* (q,r) *in* v*:*

$$\Gamma(v) = \bigcup_{(q,r)\in v} \Gamma(q,r).$$

The locality condition is very convenient to work with—however, relaxed locality conditions should also be sufficient for a generalized form of Lemma 7.

Definition 42 (Mergeable Oracle Class for Closed Finite Oracles)**.** *Let* $\mathcal{C}$ *be an oracle class and* Γ *be a closure condition. We say that* $\mathcal{C}$ *is* Γ-mergeable*, if there is a threshold* $\nu(n) = 2^{n/c}$ *for some constant* $c \geq 1$ *such that the following holds:*

Let v and w be two Γ-closed finite oracles in $\mathcal{C}$ such that w and v are consistent, $w^{<n}$ and $v^{<n}$ are consistent in $\mathcal{C}$ and such that $|w^{\geq n} \cup v^{\geq n}| \leq \nu(n)$. Then, v and w are consistent in $\mathcal{C}$.

We now define the class of closure operators and then state a generalized version of Lemma 7 for Γ-mergeable classes.

Definition 43 (Finite Closure Operator)**.** *A closure operator Γ for a class $\mathcal{C}$ is called* (k,l)-finite *for two constants k and l, if the following holds:*

(i) $\Gamma(v)$ *is at most k times greater than the domain of v.*

(ii) For all finite oracles v in $\mathcal{C}$ and all oracles O in $\mathcal{C}$ that extend v, we have that $\Gamma_O^{l+1}(v) = \Gamma_O^l(v)$, i.e., applying the closure operator Γ l times iteratively to a finite oracle v in $\mathcal{C}$ yields a Γ-closed finite oracle.

Here, the operator Γ_O maps finite prefixes of O to greater finite prefixes of O, namely, $\Gamma_O(v)$ is the oracle O restricted to the domain of $\Gamma(v)$.

Lemma 8. *Let $\mathcal{C}$ be a Γ-mergeable class, where Γ is a (k,l)-finite closure operator and let O be a generic oracle for $\mathcal{C}$. Then, $\mathcal{NP}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O,\mathbf{PSPACE}} \subseteq \mathcal{P}^{O,\mathbf{PSPACE}}$.*

Proof. To adapt the proof of Lemma 7, we need to assure that runs of M and N create Γ-closed finite oracles. Towards this goal, we modify the machines M and N such that in the end of their run, they consider the finite oracle v that they created and apply the closure operator Γ_O iteratively l times. Thereby, we blow up the number of queries by M and N to $q(|x|) := lk \cdot p(|x|)$, if $p(|x|)$ was the original upper bound on their queries. Note that one would need to argue more about the size of $q(|x|)$, if we had not added the locality condition to the definition of Γ. Using $q(|x|)$ instead of $p(|x|)$ in the proof of Lemma 7 and closing the initial sets now yields the desired result. □

7.9 On Key Agreement and Hardness of $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$

Towards obtaining an oracle, relative to which infinitely often secure key exchange exists while $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is easy, we now provide an oracle family for key exchange. We will show in Lemma 10 that this oracle family is Γ-mergeable for an appropriate closure operator Γ. By Lemma 8, this implies that $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is in $\mathcal{P}$, relative to a generic oracle from this class of key exchange oracles (and a **PSPACE** oracle). Finally, to prove Theorem 7, it then remains to show that relative to a generic oracle for this class (and a **PSPACE** oracle), infinitely often key agreement exists, which follows from genericity as we sketched for i.o. one-way functions in Section 7.3.3. Let us now define a class of key exchange oracles. Note that one can encode several oracles into one by padding.

Definition 44. *For a constant c, a strictly monotone, polynomially bounded transcript length function $\tau(n) \geq \frac{n}{c}$ and a polynomially bounded key size function $\kappa(n) \geq n$, we define the class $\mathcal{C}_{\tau,\kappa}$ of key exchange oracles. An oracle O is in $\mathcal{C}_{\tau,\kappa}$, if it encodes three functions, namely a transcript function T, a key function K and a transcript range verification function I, such that the following conditions are satisfied:*

(i) $T(\{0,1\}^n) \subseteq \{0,1\}^{\tau(n)}$ *(mapping randomness to transcript),*

(ii) $K(\{0,1\}^n \times \{0,1\}^{\tau(n)}) \subseteq \{0,1\}^{\kappa(n)}$ *(mapping transcript, randomess to key),*

(iii) $I(\{0,1\}^{\tau(n)}) \subseteq \{0,1\}$ *(co-range checking),*

(iv) $I(y) = 1$ *if and only if* $y \in Im(T)$ *(assuring co-range computability),*

(v) $\text{Prob}[K(r_A, T(r_B)) \neq K(r_B, T(r_A))] \leq 2^{-n/c}$ *(correctness w.h.p.),*

where the last probability is over the strings r_B *and* r_A *that are drawn uniformly at random from* $\{0,1\}^n$*.*

Definition 44 reflects the symmetric structure of the Diffie-Hellman key agreement protocol, i.e., Alice draws a random string r_A and sends the message $T(r_A)$; simultaneously, Bob draws a random string r_B and sends the message $T(r_B)$. Then, Alice derives her key as $K(r_A, T(r_B)$ and Bob derives his key as $K(r_B, T(r_A))$. In particular, each party sends a message which is independent from the message received from the other party. Moreover, the function that computes the message from the random string is the same for both protocol participants and so is the keying function. This fairly restricted class of key exchange protocols, of course, implies key agreement protocols with more than two messages as well as protocols with a less "symmetric" structure. Thus, as we prove a negative result, our theorems holds a fortiori for more general classes of key agreement protocols. Let us now state the main theorem of this section and show how it follows from three lemmata.

Theorem 7. *For every* $c \geq 1$*, there is an oracle* Π *such that relative to oracle* Π *and a* ***PSPACE*** *oracle, secure key agreement with a* $2^{-n/c}$*-correctness error exists and such that its transcript space is verifiable, while* $\mathcal{NP}^{\Pi,\boldsymbol{PSPACE}} \cap \boldsymbol{co}\mathcal{NP}^{\Pi,\boldsymbol{PSPACE}} \subseteq \mathcal{P}^{\Pi,\boldsymbol{PSPACE}}$*.*

Proof. The oracle Π will be instantiated by a generic oracle for the key exchange class $\mathcal{C}_{\tau,\kappa}$. We split the proof into 3 lemmata. First, we prove that $\mathcal{C}_{\tau,\kappa}$ is Γ-mergeable for an appropriate closure operator Γ (Lemma 10). Then, Lemma 8 yields that, for a generic oracle O for the class $\mathcal{C}_{\tau,\kappa}$, all problems in $\mathcal{NP}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O,\mathbf{PSPACE}}$ are in $\mathcal{P}^{O,\mathbf{PSPACE}}$. To prove Theorem 7, it remains show that infinitely many often secure key agreement exists relative to a **PSPACE** oracle and any oracle O that is generic for $\mathcal{C}_{\tau,\kappa}$ (Lemma 9). □

Lemma 9. *Relative to a* ***PSPACE*** *oracle and a generic oracle for* $\mathcal{C}_{\tau,\kappa}$*, infinitely often secure key agreement exists.*

Proof. Let O be a generic oracle for $\mathcal{C}_{\tau,\kappa}$, and assume towards contradiction that the key exchange protocol given by the oracle is not infinitely often secure. Then for all infinite, monotone sequences $(n_i)_i$, there is an adversary $\mathcal{A}$ such that $\text{Prob}[\mathcal{A}(1^{n_i}, T, k_b) = b] - \frac{1}{2}$ is not negligible. The contradiction follows by a diagonalization argument and the following observation: let $\epsilon(n) := 2^{-\frac{n}{2c}}$. By definition of genericity, for every finite number of adversaries $\mathcal{A}_1$,...,$\mathcal{A}_k$ with polynomial query-complexity, there is an infinite sequence $(n_i)_i$ such that for all $1 \leq j \leq k$, we have that $|\,\text{Prob}[\mathcal{A}(1^{n_i}, T, k_b) = b] - \frac{1}{2}| \leq \epsilon(n_i)$.

Using that Turing Machines are countable, let n_k to be the smallest security parameter such that the first k efficient adversaries have advantage less than $\epsilon(n_k)$. By construction, for all adversaries $\mathcal{A}_k$, the advantage drops below $\epsilon(n_i)$ for all $i \geq k$ and is thus negligible. □

We now define a $(2,2)$-closure operator Γ and prove that $\mathcal{C}_{\tau,\kappa}$ is Γ-mergeable. For the description of $\Gamma(v)$ on the finite oracle $v = (t,k,i) \leq (T,K,I) \in \mathcal{C}_{\tau,\kappa}$, we distinguish between adding elements to the domain of the transcript function t, the domain of key function k and the domain of the image testing function i by using the notation $\Gamma_t(v)$, $\Gamma_k(v)$ and $\Gamma_i(v)$.

Definition 45. *Let $v = (t,k,i) \in \mathcal{C}_{\tau,\kappa}$. Then, Γ is defined through the following two closure rules:*

$$(r,y) \in k \Rightarrow r \in \Gamma_t(v)$$
$$(r,y) \in t \Rightarrow y \in \Gamma_i(v).$$

Intuitively, the closure operator Γ achieves that whenever an algorithm queries the key function on randomness r, then it also queries the transcript function t on randomnesss r. Moreover, if a transcript was output by the transcript function, then the image-testing function i acknowledges this fact. We prove that $\mathcal{C}_{\tau,\kappa}$ is Γ-mergeable.

Lemma 10. *$\mathcal{C}_{\tau,\kappa}$ is Γ-mergeable for the threshold $\nu(n) := 2^{-n/c}$.*

Proof. Let (k_0,t_0,i_0) and (k_1,t_1,i_1) be two closed, consistent finite oracles that are $\mathcal{C}$-consistent up to length n and such that $|(k_0,t_0,i_0)^{\geq n} \cup (k_1,t_1,i_1)^{\geq n}|$ is smaller than $2^{\frac{n}{2c}}$. We have to show that $(k,t,i) := (k_0,t_0,i_0) \cup (k_1,t_1,i_1)$ lies in $\mathcal{C}$. Note that by locality of Γ, the finite oracle (k,t,i) is already Γ-closed. Therefore, it suffices to show that for any Γ-closed finite oracle (k,t,i) with $|(k,t,i)^{\geq n}| \leq 2^{-\frac{n}{2c}}$ that satisfies condition (i), (ii), (iii) and $i(y) = 1$, if $y \in \mathrm{Im}(t)$, there is an extension (k',t',i') that also satisfies (iv) and (v). We construct this extension in the following paragraph. The conditions on the oracles (and Γ) are only defined per every input length and thus, we only have to consider the extension within an input length. In the following paragraph, we consider n and observe that the claim holds even more for input lengths greater than n.

Let $\mathcal{T}$ be the set of transcripts that occur in the domain of $i^{=n}$, the domain of $k^{=n}$ and the image of $t^{=n}$. By definition of the size of (k,t,i) and of $\tau(n)$, we have that there is a transcript y of length $\tau(n)$ that is not in $\mathcal{T}$. On the domain of (k,t,i), we define (k',t',i') to be equal to (k,t,i). Let Y_0 be all the transcripts such that $i(y)$ is defined to be 1, although y is not in the image of t. These are at most $2^{\frac{n}{2c}}$ and thus, we can choose at most $2^{\frac{n}{2c}}$ values r that have not been assigned a value under t yet and map them onto Y_0. We define $k(r,y) := 0^{\kappa(n)}$ for all $y \in Y_0$. An r is called "old", if, so far, t' has been defined on r, and "new", if not. We choose an element y that is not in $\mathcal{T} \cup Y_0$ and such that $i(y)$ is not defined to be 0. Such a value exists by construction, as we have assigned at most $2 \cdot 2^{\frac{n}{2c}}$ transcripts and there are $2^{\frac{n}{c}}$ transcript values.

For all new r, we define $t(r) := y$ and $k(r, y') := 0^{\kappa(n)}$ for all y' as well as $k(r', y) := 0^{\kappa(n)}$ for all r'. The probability for a random r of being "old" is $2 \cdot 2^{-\frac{1}{2c}}$ and the probability for a random r of being "new" is $1 - 2 \cdot 2^{-\frac{1}{2c}}$. If out of r_A, r_B at least one is new, say r_A, then $k'(r_B, t(r_A))$ equals $0^{\kappa(n)}$, which is equal to $k'(r_A, t(r_B))$. If both, r_A and r_B are old, then there might be an error, but the probability that both random values are old is $(2 \cdot 2^{-\frac{1}{2c}})^2$, which is smaller than $2^{-n/c}$. Thus, condition (v) is satisfied.

For condition (iv), we now define $i(y)$ to be 1 if and only if $y \in \mathrm{Im}(t)$ and thereby also satisfy condition (iv). On all other values, the oracle (k', t', i') can be defined arbitrarily, for example the function k' on pairs (r, y) where $i(y) = 0$. □

7.10 On One-Way Functions from $\mathcal{NP}$-hardness

In this section, we provide an oracle, relative to which $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ equals $\mathcal{P}$, while a regular function exists that is infinitely often one-way. This gives strong evidence that if one were to prove that regular one-way functions imply hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$, then one would need to revert to non-relativizing techniques. One might interpret our result as an explanation for the difficulties in bridging the gap in the proof of Akavia et al. [AGGM06, AGGM10] that intends to show that regular one-way function (and even a larger class of one-way functions) implies hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$.

Towards this goal, we define the class of regular function oracles with exponential pre-image space.

Definition 46 (Regular Function Oracle Classes)**.** *An oracle class $\mathcal{C}$ is called a* regular function oracle class*, if there is a constant c such that*

$$\mathcal{C} = \{O | O(\{0,1\}^n) \subseteq \{0,1\}^{n/c}, |O^{-1}(y)| = 2^{|y|/c}\}.$$

We denote this class by $\mathcal{C}_c$. A function $O \in \mathcal{C}_c$ is called c-regular*.*

We will prove that, relative to a generic oracle for this class, $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ is contained in $\mathcal{P}$. An analogous version of Lemma 9 then yields that the generic oracle implements a regular function family that is infinitely many often one-way.

To prove this theorem, we first show that an analog version of Theorem 7 holds, namely that for languages $\mathcal{L}$ that lie in $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ relative to a generic oracle for a mergeable class, one can determine efficiently and deterministically a small portion of the oracle that determines whether a specific input x lies in $\mathcal{L}$ or not.

We will then extend this approach to languages in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. We will first show how to use Nisan's notion of block sensitivity [Nis91] to prove that relative to a generic oracle (for the class of all oracles), $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ is contained in $\mathcal{P}$. We will then adapt this proof to a generic oracle for the class $\mathcal{C}_c$.

Note that the regular function family, as we define it, has exponentially many pre-images for every image value. This is necessary for our proof to work—however, this is not an artifact of our proof. Interestingly, one result in Akavia et al.[AGGM06] shows that indeed, regular one-way functions with polynomially many pre-images imply hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ and even $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$.

7.10.1 $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ and Regular One-Way Functions

Theorem 8. *For a generic oracle O for $\mathcal{C}_c$, the class $\mathcal{NP}^{O,\boldsymbol{PSPACE}} \cap \boldsymbol{co}\mathcal{NP}^{O,\boldsymbol{PSPACE}}$ is contained in $\mathcal{P}^{O,\boldsymbol{PSPACE}}$ and O implements a regular function, that is infinitely many often one-way.*

Proof. As for Theorem 7, we have to prove that $\mathcal{C}_c$ is mergeable. Then, Lemma 7 yields that, relative to a generic oracle for $\mathcal{C}_c$, $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is contained in $\mathcal{P}$. Finally, we have to prove that relative to a generic oracle for $\mathcal{C}_c$, the c-regular function defined by the oracle is infinitely often one-way. This proof is analogous to the proof of Lemma 9, and we omit it. Let us now see why $\mathcal{C}_c$ is mergeable.

As the conditions on O to be in $\mathcal{C}_c$ are only length-wise, it suffices to check that length-wise, $\mathcal{C}$ is mergeable. If for the oracle w, we fix the answers to $2^{n/c}$ input values of length n, then we can still extend w to an oracle O that is c-regular as no image value has been assigned more than $2^{n/c}$ pre-images. Thus, if $v^{\leq n}$ is in $\mathcal{C}$, then for any oracle $v^{\geq n}$ with $|v^{\geq n}| \leq 2^{n/c}$ it holds that $v^{\leq n} \cup v^{\geq n} \in \mathcal{C}$. Therefore, $\mathcal{C}_c$ is mergeable for threshold $\nu(n) = 2^{n/c}$. □

7.10.2 Certificate Complexity and Block Sensitivity

The block sensitivity of a function measures how sensitive the function is to modifications of a certain number of bits of the oracle—recall that we consider an oracle as an infinite sequence of bits. Similarly, the certificate complexity measures the number of bits of the oracle we have to know to be able to determine the correct function value.

The following definitions are adapted from Nisan [Nis91], who defines these notion to meaure the decision tree complexity of boolean functions. When considering the oracle as an input to a function, we yield the same definitions as Nisan. However, for simplicity, we directly state our definitions for oracles.

Definition 47 (Certificate Complexity)**.** *Let M be an oracle machine that defines an $\mathcal{AM}$-language $\mathcal{L}^O$ for all oracles O. The certificate complexity $C(x, O)$ of a pair (x, O) is the length of the smallest finite prefix w of O such that for all extensions $O' \geq w$, one has $x \in \mathcal{L}^O$ if and only if $x \in \mathcal{L}^{O'}$. The certificate complexity $C(x)$ is the maximum of $C(x, O)$ over all O.*[3] *The certificate complexity C of M is the smallest function from $\mathbb{N}$ to $\mathbb{N}$ such that for all x, $C(x) \leq C(|x|)$. For $b \in \{0, 1\}$, the b-certificate complexity $C_b(x)$ of x is the maximum of $C(x, O)$ over all O, for which $\chi_{L^O}(x) = b$. If the set of these O is empty, we define $C_b(x) := 0$. The b-certificate complexity of M is the smallest function $C_b : \mathbb{N} \to \mathbb{N}$ such that for all x, $C_b(x) \leq C_b(|x|)$.*

If later, we consider certificate complexity with respect to two oracle $\mathbf{PSPACE}, O$, then as before, we regard **PSPACE** as fixed, i.e., the certificate complexity is only over O. The same holds for the following notion of block sensitivity.

Definition 48 (Block Sensitivity)**.** *Let M be an oracle machine that defines an $\mathcal{AM}$-language $\mathcal{L}^O$ for all oracles O. Let S be a finite set of queries and let $O_{(S)}$ be the oracle*

[3] which is well-defined as the number of queries by M is bounded.

that answers according to O, if a query is not in S, and flips the answer of O, if the query is in S. We say that a pair (x, O) is sensitive to S, if $x \in L^O$ if and only if $x \notin L^{O^{(S)}}$. The block sensitivity $sb(x, O)$ of a pair (x, O) is the maximal number of disjoint sets S_1,...,S_k such that (x, O) is sensitive to each of them. The block sensitivity $sb(x)$ is the minimum of $sb(x, O)$ over all oracles O. The block sensitivity sb is the greatest function from $\mathbb{N}$ to $\mathbb{N}$ such that for all x, $sb(x) \geq sb(|x|)$. For $b \in \{0, 1\}$, the b-block sensitivity $sb_b(x)$ of x is the minimum of $sb(x, O)$ over all those O, for which $\chi_{L^O}(x) = b$. If the set of these O is empty, we define $sb_b(x) := 0$. The b-block sensitivity of M is the greatest function $sb_b : \mathbb{N} \to \mathbb{N}$ such that for all x, $sb_b(x) \geq sb_b(|x|)$.

Sometimes, we consider different machines M and N in the same discussion. In this case, we augment the the above notations by an index M or N.

7.10.3 $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ and Generic Oracles

If no class is specified, we consider the class of all one-bit output oracles. In this section, we prove the following theorem.

Theorem 9. *Let O be a generic oracle (for the class of all oracles), then $\mathcal{AM}^{O,\boldsymbol{PSPACE}} \cap \boldsymbol{co}\mathcal{AM}^{O,\boldsymbol{PSPACE}} \subseteq \mathcal{P}^{O,\boldsymbol{PSPACE}}$, while $\mathcal{NP}^{O,\boldsymbol{PSPACE}} \not\subseteq \mathcal{P}^{O,\boldsymbol{PSPACE}}$.*

Proof. We already know that $\mathcal{NP}^{O,\mathbf{PSPACE}} \not\subseteq \mathcal{P}^{O,\mathbf{PSPACE}}$, our goal in this section is to prove that $\mathcal{AM}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O,\mathbf{PSPACE}} \subseteq \mathcal{P}^{O,\mathbf{PSPACE}}$. Again, our proof is split into proving the following two claims.

(i) If two polynomial-time oracle Turing machines $M^{O',\mathbf{PSPACE}}$ and $N^{O',\mathbf{PSPACE}}$ define a language in $\mathcal{AM}^{O',\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O',\mathbf{PSPACE}}$ for all oracles O', then there is a deterministic polynomial-time oracle Turing Machine $\mathcal{A}$ such that for all oracles O' the algorithm $\mathcal{A}^{O',\mathbf{PSPACE}}$ decides $\mathcal{L}_M^{O'}$ (and thus also $\mathcal{L}_N^{O'}$).

(ii) If $M^{O,\mathbf{PSPACE}}$ and $N^{O,\mathbf{PSPACE}}$ define a language in $\mathcal{AM}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O,\mathbf{PSPACE}}$ for the generic oracle O then there are machines M' and N' such that $\mathcal{L}_N^{O,\mathbf{PSPACE}} = \mathcal{L}_{N'}^{O,\mathbf{PSPACE}}$ and $\mathcal{L}_M^{O,\mathbf{PSPACE}} = \mathcal{L}_{M'}^{O,\mathbf{PSPACE}}$ and such that, for all oracles $O' \in \mathcal{C}$, $M'^{O,\mathbf{PSPACE}}$ and $N'^{O,\mathbf{PSPACE}}$ define a language in $\mathcal{AM}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O,\mathbf{PSPACE}}$.

As for the Main Lemma 7, from these two claims, we can derive that for the generic oracle O, the class $\mathcal{AM}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O,\mathbf{PSPACE}}$ is contained in $\mathcal{P}^{O,\mathbf{PSPACE}}$. Given two machines M and N that define a language in $\mathcal{AM}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O,\mathbf{PSPACE}}$, we use (ii) to transform them into M' and N', for which (i) yield that their respective languages are easy to decide.

Item (i) will be proved in Lemma 13, we prove item (ii) now. The proof is analogous to the proof of Claim 2. Let M and N two machines that define an language $\mathcal{L}_M^O$ in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ relative to the generic oracle O and the **PSPACE** oracle. Let T be the oracle Turing Machine that on input x, returns 1, if and only if the $\mathcal{AM}$-languages $\mathcal{L}_M^{O'}$ and $\mathcal{L}_N^{O'}$ are complementary relative to O' on input length $|x|$. Let us consider the two conditions in Lemma 4. As M and N are complementary relative to O, condition (i) is

not satisfied, i.e., there are not infinitely many x_i such that $T^O(x_i) = 0$; actually, there is not even a single such input x_i. Thus, we know that condition (ii) of Lemma 4 holds, i.e., there is a finite prefix w of O that forces T to return 1 above a certain threshold Λ. We hardcode the behaviour of M^O up to length Λ into M' and the behaviour of N^O up to length Λ unto N', and we also hardcode the prefix w into M' and N', namely, M' runs M as a subroutine on witness z and values x with $\Lambda < |x|$; and whenever M' makes a query to its oracle that is in w, then M' does not query its real oracle O', but instead, it returns the answer stored in w. We define N' in the same way. M' and N' are polynomial-time, because these operations take constant-time only. Moreoever, by construction, the $\mathcal{AM}$-languages $\mathcal{L}^{O'}_{M'}$ and $\mathcal{L}^{O'}_{N'}$ are complementary relative to any oracle O', as the prefix w forced M and N to be complementary above certain threshold and as, below the threshold Λ, M' and N' are complementary, as M^O and N^O are. Moreover, relative to O, we have that M'^O behaves like M^O and N'^O behaves like N^O. □

Lemma 11. *Let M be an oracle machine that defines an $\mathcal{AM}$-language $\mathcal{L}^O$ for* all *oracles O, then the block sensitivity sb_1 of the* 1*-instances is smaller than q, the number of queries made by M.*

Again, we require Lemma 11 in its relativized version.

Lemma 11. *(relativized) Let M be an oracle machine that defines an $\mathcal{AM}$-language $\mathcal{L}^{O,\boldsymbol{PSPACE}}$ for* all *oracles O, then the block sensitivity sb_1 of the* 1*-instances is smaller than q, the number of queries made by M.*

The proof for both version of Lemma 11 are analogous. We now prove the unrelativized version of Lemma 11.

Proof. Assume that there is an oracle O and an input x such that $\chi_{\mathcal{L}^O}(x) = 1$ and $sb(O, x) > q(|x|)$. Then, by definition, there are $k := q(|x|) + 1$ disjoint sets S_i such that (x, O) are sensitive to all of them. For each r, let z_r be a witness such that for a $1-2^{-n}$-fraction of the r-values, we have $M^O(x, r, z_r) = 1$. Assume that there is a polynomial fraction of r such that $M^O(x, r, z_r)$ does not make any query on S_i. Then, there is a polynomial fraction of r such that $M^{O(S_i)}(x, r, z_r) = M^O(x, r, z_r) = 1$ in contradiction to the $\mathcal{AM}$ requirement for 0-instances. Thus, with probability $1-2^{-|x|}$ over r, the machine $M^O(x, r, z_r)$ makes a query in S_i. As this holds for all S_i, an averaging argument yields that with probability $1 - k \cdot 2^{-|x|}$ over r, the machine $M^O(x, r, z_r)$ makes a query on all of the S_i. As there are $k = q(|x|) + 1$ disjoint sets S_i, the machine $M^O(x, r, z_r)$ makes at least $q(|x|)+1$ queries for these r, a contradiction to the bound $q(|x|)$ on the queries. □

Providing an efficient decision algorithm $\mathcal{A}$ for a language means to give an upper bound on the deterministic computation complexity of the language. We will now give an upper bounds on another complexity measure, namely the block sensitivity and will then show in Lemma 13 how to transform this bound into a bound in the deterministic computation time.

Lemma 12. *Let M and N be oracle machines that define $\mathcal{AM}$-languages $\mathcal{L}_M^O$ and $\mathcal{L}_N^O$ for all oracles O and that are complementary for all oracles O, i.e., $\mathcal{L}^O = \mathcal{L}_M^O = \overline{\mathcal{L}_N^O}$, then the block sensitivity sb is smaller than q, where q is a common upper bound in the number of queries.*

Lemma 12. *(relativized) Let M and N be oracle machines that define $\mathcal{AM}$-languages $\mathcal{L}_M^{O,\boldsymbol{PSPACE}}$ and $\mathcal{L}_N^{O,\boldsymbol{PSPACE}}$ for all oracles O and that are complementary for all oracles O, i.e., $\mathcal{L}^O = \mathcal{L}_M^O = \overline{\mathcal{L}_N^O}$, then the block sensitivity sb is smaller than q, where q is a common upper bound in the number of queries.*

Again, the proofs of the relativized version and the unrelativized version are analogous. We state the proof for the unrelativized Lemma 12.

Proof. As M and N are complementary for all oracles O, we have that if N on instance (x, O) is sensitive to a set S, then also M is sensitive to S on (x, O). In particular, if a 1-instance (x, O) of N has block sensitivity k, then the corresponding 0-instance (x, O) of M also has block sensitivity k. Therefore, by Lemma 11, the block sensitivity sb_0^M is also bounded by q and thus, sb_M is overall bounded by q. □

Now, that we have an upper bound on the complexity measure of block sensitivity, the proof of the following lemma shows us how to relate the latter into an upper bound in the computation time, which concludes the proof of Theorem 9.

Lemma 13. *If two polynomial-time oracle Turing machines $M^{O',\boldsymbol{PSPACE}}$ and $N^{O',\boldsymbol{PSPACE}}$ define a language in $\mathcal{AM}^{O',\boldsymbol{PSPACE}} \cap \boldsymbol{co}\mathcal{AM}^{O',\boldsymbol{PSPACE}}$ for all oracles O', then there is a deterministic polynomial-time oracle Turing Machine $\mathcal{A}$ such that for all oracles O' the algorithm $\mathcal{A}^{O',\boldsymbol{PSPACE}}$ decides $\mathcal{L}_M^{O'}$ (and thus also $\mathcal{L}_N^{O'}$).*

Proof. Our goal is to determine the deterministic computation time of $\mathcal{L}_M^{O'}$. Nisan (see Lemma 2.4. in [Nis91]) and Blum and Impagliazzo (see Lemma 2.2 in [BI87]) prove that, relative to a **PSPACE** oracle, the deterministic computation time of a language is at most (polynomial in) the quare of its certificate complexity. So far, we only bounded the block sensitivity of the language $\mathcal{L}_M^{O'}$. Nisan (see Lemma 2.4 in [Nis91]) proves that the square of the block sensitivity of a language upper bounds its certificate complexity, i.e., $C_M \leq (sb_M)^2$. Thus, putting both inequalities together, $\mathcal{L}_M^{O'}$ can be decided using $(C_M)^2 \leq (sb_M)^4 \leq q^4$ queries to O', where q is an upper bound on the number of queries used by M; and $\mathcal{L}_M^{O'}$ can be decided in time less than polynomial in q^4. □

7.10.4 $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$ and Generic Regular Function Oracles

In Theorem 9, we proved that for a generic oracle O for the class of all oracles, it holds that $\mathcal{AM}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O,\mathbf{PSPACE}}$ is contained in $\subseteq \mathcal{P}^{O,\mathbf{PSPACE}}$. In turn, Theorem 8 establishes that for a generic oracle O *for the class of regular functions* $\mathcal{C}_c$, the class $\mathcal{NP}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{NP}^{O,\mathbf{PSPACE}}$ is a subset of $\mathcal{P}^{O,\mathbf{PSPACE}}$. Moreover, we proved that O is an regular i.o. one-way function. We now take the best of both worlds showing that if O is the latter oracle, then it also holds that $\mathcal{AM}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O,\mathbf{PSPACE}}$ is contained in $\mathcal{P}^{O,\mathbf{PSPACE}}$.

Theorem 10. *Let O be a generic oracle O for $\mathcal{C}_c$, then the class $\mathcal{AM}^{O,\boldsymbol{PSPACE}} \cap \boldsymbol{co}\mathcal{AM}^{O,\boldsymbol{PSPACE}}$ is contained in $\mathcal{P}^{O,\boldsymbol{PSPACE}}$ and O implements a regular i.o. one-way function.*

We already established that O implements a regular i.o. one-way function. Let us show that also, $\mathcal{AM}^{O,\mathbf{PSPACE}} \cap \mathbf{co}\mathcal{AM}^{O,\mathbf{PSPACE}}$ is contained in $\mathcal{P}^{O,\mathbf{PSPACE}}$. Moreover, the proofs of Lemma 11 and Lemma 12 carry over to the case of a generic oracle for the class of regular function oracles $\mathcal{C}_c$. Moreover, Lemma 7 and Theorem 8 tell us that for mergeable oracle classes such as $\mathcal{C}_c$, the square of the certificate complexity is an upper bound on the deterministic computation time. The hardest part to show that Nisan's Lemma **2.4** [Nis91] can be adapted from the class of all oracles to the class of regular oracles. In particular, we need to adapt the notions of certificate complexity and block sensitivity for restricted oracle classes.

Definition 49 (Certificate Complexity Relative to a Class)**.** *Let M be an oracle machine that defines an $\mathcal{AM}$-language $\mathcal{L}^O$ for all oracles $O \in \mathcal{C}$. The certificate complexity $C(x, O)$ of a pair (x, O) is the length of the smallest finite prefix w of O such that for all extensions $O' \geq w$ in $\mathcal{C}$, one has $x \in \mathcal{L}^O$ if and only if $x \in \mathcal{L}^{O'}$. The certificate complexity $C(x)$ is the maximum of $C(x, O)$ over all $O \in \mathcal{C}$[4]. The certificate complexity C of M is the smallest function from $\mathbb{N}$ to $\mathbb{N}$ such that for all x, $C(x) \leq C(|x|)$. For $b \in \{0, 1\}$, the b-certificate complexity $C_b(x)$ of x is the maximum of $C(x, O)$ over all $O \in \mathcal{C}$, for which $\chi_{L^O}(x) = b$. If the set of these O is empty, we define $C_b(x) := 0$. The b-certificate complexity of M is the smallest function $C_b : \mathbb{N} \to \mathbb{N}$ such that for all x, $C_b(x) \leq C_b(|x|)$.*

Definition 50 (Block Sensitivity Relative to a Class)**.** *Let $\mathcal{C}$ be an oracle class. Let M be an oracle machine that defines an $\mathcal{AM}$-language $\mathcal{L}^O$ for all oracles $O \in \mathcal{C}$. Let S be a finite set of queries and let $O_{(S)}$ be the oracle that answers according to O, if a query is not in S, and flips the answer of O, if the query is in S. We say that a pair (x, O) is sensitive to S, if $x \in L^O$ if and only if $x \notin L^{O_{(S)}}$. The block sensitivity $sb(x, O)$ of a pair (x, O) is the maximal number of disjoint sets S_1,...,S_k such that $O_{S_i} \in \mathcal{C}$ and (x, O) is sensitive to each of them. The block sensitivity $sb(x)$ is the minimum of $sb(x, O)$ over all oracles $O \in \mathcal{C}$. The block sensitivity sb is the greatest function from $\mathbb{N}$ to $\mathbb{N}$ such that for all x, $sb(x) \geq sb(|x|)$. For $b \in \{0, 1\}$, the b-block sensitivity $sb_b(x)$ of x is the minimum of $sb(x, O)$ over all those $O \in \mathcal{C}$, for which $\chi_{L^O}(x) = b$. If the set of these O is empty, we define $sb_b(x) := 0$. The b-block sensitivity of M is the greatest function $sb_b : \mathbb{N} \to \mathbb{N}$ such that for all x, $sb_b(x) \geq sb_b(|x|)$.*

Lemma 14. *Let $\mathcal{C}$ be a regular function oracle class. Then, for each oracle machine M, it holds that the block sensitivity sb of M relative to the oracle class $\mathcal{C}$ (and a **PSPACE** oracle), is greater than $\frac{1}{3p(n)}\sqrt{C(n)}$, where C is the certificate complexity of M relative to $\mathcal{C}$ (and a **PSPACE** oracle) and p is an upper bound on the running time of M.*

[4]This number is well-defined as the number of queries by M is bounded.

Proof. Let x be a string and $O \in \mathcal{C}$ be an oracle such that (x, O) achieves the certificate complexity. Let S_1 be a minimal set such that $O_{(S_1)}$ is in $\mathcal{C}$ and such that M is sensitive to S_1. Here, minimality is defined with respect to the multi-bit-output regular function encoded by O, i.e., we want that as few as possible of the function values are modified.

We continute to pick S_2 or more generally S_i as a minimal set disjoint from all previous sets such that $O_{(S_i)}$ is in $\mathcal{C}$ and such that M is sensitive to S_i, until after picking S_t, we cannot find any other disjoint set that satisfies the condition.

The union of all S_i is a certificate for (x, O), as else, there would be another disjoint set S_{t+1} such that M is sensitive to S_{t+1} on (x, O). Therefore, we have that

$$\sum_{i=1}^{t} |S_i| \geq C(x, O).$$

Then, we can derive the following two bounds:

(i) $bs(x) \geq \frac{1}{3p} \max_i |S_i|$

(ii) $bs(x, O) \geq t$

The latter bound follows from the fact that a certificate for (x, O) has to contain at least one element from each S_i. We will see that the first bound follows from the "symmetric" structure of the S_i, namely $O_{(S_i)}$ has to be a regular function again—if one flips the response r of one query to r', there are too many pre-images for r' and too few pre-images for r. As $O_{(S_i)}$ is a regular functions, other queries have to even out the difference. From now on, we consider O as a multi-bit output oracle and the sets S_i as sets of query-response, namely all those pairs (q, r) of the regular function O, which have to be modified to obtain $O_{(S_i)}$.

If (q_0, r_0) is in O and (q_0, r_1) is in S_i, then some element has to make up for the loss of a pre-image of r_0, as the functions are regular. Simultaneously, some element (q_1, r_1) in O has to make up for the increase of pre-images of r_0 in S_i, that is, there is a q_1 such that (q_1, r_1) in O while (q_1, r_2) is in S_i. We can thus generate such a sequence of pairs (q_k, r_{k+1}) in S_i. But as all modifications have to even out to preseve regularity, it follows that there has to be an n such that (q_n, r_0) is in S_i. Thus, these elements form a cycle. We can repeat this process and thus split S_i into several cycles C_1,...,C_l. Note that $(x, O_{(S_i)})$ is sensitive to each cycle in S_i, as else, S_i would not have been minimal. We will now split up each C_j into disjoint pairs or triples to each of which S_i is sensitive. Let (q_k, r_{k+1}) denote the cycle C_j. Then, $(x, O_{(S_i)})$ is sensitive to $\{(q_0, r_2), (q_1, r_1)\}$ and to $\{(q_2, r_4), (q_3, r_3)\}$ etc.—if it were not, the choice of S_i would not have been minimal with respect to the number of queries whose answers had to be modified. Thus, each cycle C_i defines $\lfloor |C_i|/2 \rfloor$ many disjoint pairs that $(x, O_{(S_i)})$ is sensitive to. In the worst case, all C_i contain exactly 3 pairs, and then, there are $\#S_i/3$ pairs that $(x, O_{(S_i)})$ is sensitive to, where $\#S_i$ denotes the number of pairs (q, r) defined by S_i. Therefore,

$$bs(x) \geq bs(x, O_{(S_i)}) \geq \tfrac{1}{3}\#S_i \geq \tfrac{|S_i|}{3p},$$

where the last inequality follows from the fact that not more than q bits can be associated with the same query.

From (i) and (ii), we can now derive a lower bound on bs. If t is greater than $\sqrt{C}$, then (ii) yields that $bs(x) \geq bs(x, O) \geq t$. If t is smaller than $\sqrt{C}$, then at least one of the S_i has to be greater than $\sqrt{C}$. By (i), we obtain that $bs(x) \geq \frac{1}{3p}\sqrt{C}$. □

Chapter 8

Conclusion and Open Problems

We initiate the study of purely game-based composition theorems for key agreement and hope that our framework might inspire further research on game-based composition. Our results establish that forward-secure BR-secure key exchange protocols are generally composable with symmetric-key protocols, and general BR-secure key exchange protocols are composable with all single-session-reducible symmetric-key protocols. For our proofs to work, we assumed a public session matching algorithms and we showed that conversely, if a general composition theorem holds for a key exchange protocol, then the key exchange protocol admits a weak public session matching. It remains to determine the exact relationship, and it would be interesting to establish a formal connection to the results on sesssion matching established in [GR13].

While the first two theorems target BR-secure protocols such as [DF10], many popular protocols escape the BR-framework due to the implementation of explicit key confirmation, namely, key exchange protocols use a future encryption key already for encryption during the key exchange phase. While key confirmation destroys key indistinguishability, intuitively, it should not harm the security of a protocol.

We thus devise a more flexible security definition that captures this intution. Interestingly, we can show that our notion in closed under reductions, namely under key-independent reductions, a notion that we put forth. Thereby, even in this model, we provide a strategy to perform a modular analysis of key agreement protocols with a subsequent channel protocol. As an application, we prove the security of a simplified version of TLS.

Closure under reductions and composition of games are two important principles that we hope are of independent interest. In particular, we "glue" games together which seems a natural approach to consider joint attack models. Maybe, even for those, closure under reductions can be proven.

Moreover, it would be interesting to carry out a standard model analysis of TLS in our model, and moreover, other practical key exchange protocols that use key confirmation can now be analysed in our model.

Another interesting extension is a model and composition result for one-sided authentication. In particular, it seems worthy to analyse TLS in this model, as TLS is

mostly used with one-sided authentication only, as clients tend not to have certificates.

In the area of complexity-theoretic assumptions, we establish an oracle result that indicates that $\mathcal{NP} \cap \mathbf{co}\mathcal{NP}$ is not a necessary assumption for secure key agreement, if the key agreement protocol does not achieve perfect correctness—more precisely, that such a result cannot be established using relativizing techniques. It would be interesting to see, whether a similar result can be shown for perfectly correct key agreement protocols whose transcript space is not co-range verifiable. Moreover, one might be able to extend this result to $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. The most important goal would be, of course, to construct a candidate key agreement scheme that does not rely on hard problems in $\mathcal{AM} \cap \mathbf{co}\mathcal{AM}$. The aforementioned extension of our impossibility result seem a worthwhile direction to understand which loopholes to exploit. Another interesting approach would be to exploit interaction, as suggested by Rudich [Rud88]. To date, no such aproach are known.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[AD07] Miklós Ajtai and Cynthia Dwork. The first and fourth public-key cryptosystems with worst-case/average-case equivalence. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(097), 2007.

[AGGM06] Adi Akavia, Oded Goldreich, Shafi Goldwasser, and Dana Moshkovitz. On basing one-way functions on NP-hardness. In Jon M. Kleinberg, editor, *38th Annual ACM Symposium on Theory of Computing*, pages 701–710, Seattle, Washington, USA, May 21–23, 2006. ACM Press.

[AGGM10] Adi Akavia, Oded Goldreich, Shafi Goldwasser, and Dana Moshkovitz. Erratum for: on basing one-way functions on np-hardness. In *STOC*, pages 795–796. ACM, 2010.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th Annual ACM Symposium on Theory of Computing*, pages 99–108, Philadephia, Pennsylvania, USA, May 22–24, 1996. ACM Press.

[Ajt98] Miklós Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract). In *30th Annual ACM Symposium on Theory of Computing*, pages 10–19, Dallas, Texas, USA, May 23–26, 1998. ACM Press.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP TCS*, volume 1872 of *LNCS*, pages 3–22. Springer, 2000.

[AR05] Dorit Aharonov and Oded Regev. Lattice problems in np cap conp. *J. ACM*, 52(5):749–765, 2005.

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd Annual Symposium on Foundations of Computer Science*, pages 106–115, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press.

[BBF13] Paul Baecher, Christina Brzuska, and Marc Fischlin. Notions of black-box reductions, revisited. http://eprint.iacr.org/2013/101, 2013.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274, Bruges, Belgium, May 14–18, 2000. Springer, Berlin, Germany.

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.

[BCFW09] Alexandra Boldyreva, David Cash, Marc Fischlin, and Bogdan Warinschi. Foundations of non-malleable hash and one-way functions. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 524–541, Tokyo, Japan, December 6–10, 2009. Springer, Berlin, Germany.

[BDF12] Christina Brzuska, Özgür Dagdelen, and Marc Fischlin. TLS, PACE, and EAC: A cryptographic view on modern key exchange protocols. In *Sicherheit 2012*, volume 195 of *Lecture Notes in Informatics*, pages 71–82. Gesellschaft für Informatik (GI), 2012.

[BDFK12] Jens Bender, Özgür Dagdelen, Marc Fischlin, and Dennis Kügler. The PACE|AA protocol for machine readable travel documents, and its security. In *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*, pages 344–358. Springer, 2012.

[BFS+12] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: Relaxed yet composable security notions for key exchange. *IACR Cryptology ePrint Archive*, 2012:242, 2012.

[BFS+13] Christina Brzuska, Marc Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams. Less is more: Relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, ??:??–??, 2013.

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 51–62. ACM Press, 2011.

[BH13] Kfir Barhum and Thomas Holenstein. A cookbook for black-box separations and a recipe for uowhfs. In *TCC*, pages 662–679, 2013.

[BI87] Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes (extended abstract). In *FOCS*, pages 118–126. IEEE Computer Society, 1987.

[BLR] Boaz Barak, Yehuda Lindell, and Tal Rabin. Protocol initialization for the framework of universal composability. eprint: `http://eprint.iacr.org/2004/006`.

[BM03] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Kyoto, Japan, December 3–7, 2000. Springer, Berlin, Germany.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Berlin, Germany.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 03: 10th Conference on Computer and Communications Security*, pages 220–230, Washington D.C., USA, October 27–30, 2003. ACM Press.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, Santa Barbara, CA, USA, August 22–26, 1993. Springer, Berlin, Germany.

[BR95] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In *27th Annual ACM Symposium on Theory of Computing*, pages 57–66, Las Vegas, Nevada, USA, May 29 – June 1, 1995. ACM Press.

[BT03] Andrej Bogdanov and Luca Trevisan. On worst-case to average-case reductions for NP problems. In *44th Annual Symposium on Foundations of Computer Science*, pages 308–317, Cambridge, Massachusetts, USA, October 11–14, 2003. IEEE Computer Society Press.

[BV98] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.

[BWJM97] Simon Blake-Wilson, Don Johnson, and Alfred Menezes. Key agreement protocols and their security analysis. In *IMA International Conference on Cryptography and Coding*, pages 30–45. Springer, 1997.

[Cai98] Jin-Yi Cai. A relation of primal-dual lattices and the complexity of shortest lattice vector problem. *Theoretical Computer Science*, 207:105–116, 1998.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA, October 14–17, 2001. IEEE Computer Society Press.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

[CHL02] Yan-Cheng Chang, Chun-Yun Hsiao, and Chi-Jen Lu. On the impossibilities of basing one-way permutations on central cryptographic primitives. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 110–124, Queenstown, New Zealand, December 1–5, 2002. Springer, Berlin, Germany.

[CK01] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria, May 6–10, 2001. Springer, Berlin, Germany.

[CK02] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 272–287, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.

[DA06] T. Dierks and C. Allen. The TLS protocol version 1.2. In *RFC 4346*, 2006.

[DDM+05] Anupam Datta, Ante Derek, John C. Mitchell, Vitaly Shmatikov, and Mathieu Turuani. Probabilistic polynomial-time semantics for a protocol security logic (invited lecture). In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005: 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 16–29, Lisbon, Portugal, July 11–15, 2005. Springer, Berlin, Germany.

[DDMW06] Anupam Datta, Ante Derek, John C. Mitchell, and Bogdan Warinschi. Computationally sound compositional logic for key exchange protocols. In *CSFW*, pages 321–334. IEEE Computer Society, 2006.

[DF10] Özgür Dagdelen and Marc Fischlin. Security analysis of the extended access control protocol for machine readable travel documents. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC 2010: 13th International Conference on Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 54–68, Boca Raton, FL, USA, October 25–28, 2010. Springer, Berlin, Germany.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DHT12] Yevgeniy Dodis, Iftach Haitner, and Aris Tentes. On the instantiability of hash-and-sign rsa signatures. In *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 112–132. Springer, 2012.

[DKS98] Irit Dinur, Guy Kindler, and Shmuel Safra. Approximating-CVP to within almost-polynomial factors is NP-hard. In *39th Annual Symposium on Foundations of Computer Science*, pages 99–111, Palo Alto, California, USA, November 8–11, 1998. IEEE Computer Society Press.

[DOP05] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 449–466, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany.

[FF93] Joan Feigenbaum and Lance Fortnow. Random-self-reducibility of complete sets. *SIAM J. Comput.*, 22(5):994–1005, 1993.

[FF13] Marc Fischlin and Nils Fleischhacker. Limitations of the meta-reduction technique: The case of schnorr signatures. In *Eurocrypt 2013, to appear*, 2013.

[Fis99] Marc Fischlin. Pseudorandom function tribe ensembles based on one-way permutations: Improvements and applications. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 432–445, Prague, Czech Republic, May 2–6, 1999. Springer, Berlin, Germany.

[FLR+10] Marc Fischlin, Anja Lehmann, Thomas Ristenpart, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Random oracles with(out) programmability. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 303–320, Singapore, December 5–9, 2010. Springer, Berlin, Germany.

[FS10] Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 197–215, French Riviera, May 30 – June 3, 2010. Springer, Berlin, Germany.

[GG98] Oded Goldreich and Shafi Goldwasser. On the limits of non-approximability of lattice problems. In *30th Annual ACM Symposium on Theory of Computing*, pages 1–9, Dallas, Texas, USA, May 23–26, 1998. ACM Press.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, pages 365–377. ACM, 1982.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective*. Cambridge University Press, 2008.

[GR13] Wesley George and Charles Rackoff. Rethinking definitions of security for session key agreement. *IACR Cryptology ePrint Archive*, 2013:139, 2013.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108. ACM, 2011.

[HH09] Iftach Haitner and Thomas Holenstein. On the (im)possibility of key dependent encryption. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 202–219. Springer, Berlin, Germany, March 15–17, 2009.

[HHRS07] Iftach Haitner, Jonathan J. Hoch, Omer Reingold, and Gil Segev. Finding collisions in interactive protocols - a tight lower bound on the round complexity of statistically-hiding commitments. In *48th Annual Symposium on Foundations of Computer Science*, pages 669–679, Providence, USA, October 20–23, 2007. IEEE Computer Society Press.

[HMX10] Iftach Haitner, Mohammad Mahmoody, and David Xiao. A new sampling protocol and applications to basing cryptographic primitives on the hardness of np. In *IEEE Conference on Computational Complexity*, pages 76–87. IEEE Computer Society, 2010.

[HOZ13] Iftach Haitner, Eran Omri, and Hila Zarosim. Limits on the usefulness of random oracles. In *TCC*, pages 437–456, 2013.

[HR07] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 469–477, San Diego, California, USA, June 11–13, 2007. ACM Press.

[HRS09] Iftach Haitner, Alon Rosen, and Ronen Shaltiel. On the (im)possibility of Arthur-Merlin witness hiding protocols. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 220–237. Springer, Berlin, Germany, March 15–17, 2009.

[IL89] Russell Impagliazzo and Michael Luby. One-way functions are essential for complexity-based cryptography. In *30th Annual Symposium on Foundations of Computer Science*, pages 230–235, Research Triangle Park, North Carolina, October 30 – November 1, 1989. IEEE Computer Society Press.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory Conference*, pages 134–147, 1995.

[Imp11] Russell Impagliazzo. Relativized separations of worst-case and average-case complexities for np. In *IEEE Conference on Computational Complexity*, pages 104–114, 2011.

[IR90] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 8–26, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Berlin, Germany.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In *Advances in Cryptology – CRYPTO 2012*, Lecture Notes in Computer Science, pages 273–293, Santa Barbara, CA, USA, August 2012. Springer, Berlin, Germany.

[KK12] Saqib A. Kakvi and Eike Kiltz. Optimal security proofs for full domain hash, revisited. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 537–553. Springer, 2012.

[KP09] Eike Kiltz and Krzysztof Pietrzak. On the security of padding-based encryption schemes - or - why we cannot prove OAEP secure in the standard model. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 389–406, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

[KT11] Ralf Küsters and Max Tuengerthal. Composition theorems without pre-established session identifiers. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *CCS*, pages 41–50. ACM, 2011.

[Küs06] Ralf Küsters. Simulation-based security with inexhaustible interactive turing machines. In *CSFW*, pages 309–320. IEEE Computer Society, 2006.

[KY07] Jonathan Katz and Moti Yung. Scalable protocols for authenticated group key exchange. *Journal of Cryptology*, 20(1):85–113, January 2007.

[LLM06] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. eprint: `http://eprint.iacr.org/2006/073`, 2006.

[LOZ12] Yehuda Lindell, Eran Omri, and Hila Zarosim. Completeness for symmetric two-party functionalities - revisited. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2012.

[MP12] Mohammad Mahmoody and Rafael Pass. The curious case of non-interactive commitments - on the power of black-box vs. non-black-box use of primitives. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 701–718. Springer, 2012.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th Annual Symposium on Foundations of Computer Science*, pages 372–381, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[MSW10] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. The TLS handshake protocol: A modular analysis. *Journal of Cryptology*, 23(2):187–223, April 2010.

[MT10] Ueli Maurer and Björn Tackmann. On the soundness of authenticate-then-encrypt: formalizing the malleability of symmetric encryption. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 10: 17th Conference on Computer and Communications Security*, pages 505–515, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[Nis91] Noam Nisan. Crew prams and decision trees. *SIAM J. Comput.*, 20(6):999–1007, 1991.

[Pas11] Rafael Pass. Limits of provable security from standard assumptions. In *STOC*, pages 109–118. ACM, 2011.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389. Springer, 2011.

[PTV11] Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Towards non-black-box lower bounds in cryptography. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 579–596. Springer, 2011.

[PV06] Pascal Paillier and Jorge L. Villar. Trading one-wayness against chosen-ciphertext security in factoring-based encryption. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 252–266, Shanghai, China, December 3–7, 2006. Springer, Berlin, Germany.

[Reg03] Oded Regev. New lattice based cryptographic constructions. In *35th Annual ACM Symposium on Theory of Computing*, pages 407–416, San Diego, California, USA, June 9–11, 2003. ACM Press.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, Maryland, USA, May 22–24, 2005. ACM Press.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *22nd Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, Maryland, USA, May 14–16, 1990. ACM Press.

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, Cambridge, MA, USA, February 19–21, 2004. Springer, Berlin, Germany.

[Rud88] Steven Rudich. Limits on the provable consequences of one-way-functions. In *Thesis*, 1988.

[Rud92] Steven Rudich. The use of interaction in public cryptosystems (extended abstract). In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 242–251, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Berlin, Germany.

[Seu12] Yannick Seurin. On the exact security of schnorr-type signatures in the random oracle model. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2012.

[Sho99a] Victor Shoup. On formal models for secure key exchange. In *IBM Research Report RZ 3120*, 1999.

[Sho99b] Victor Shoup. On formal models for secure key exchange. eprint: `http://eprint.iacr.org/1999/012`, 1999.

[Wat10] Thomas Watson. Relativized worlds without worst-case to average-case reductions for np. In *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 752–765. Springer, 2010.

[Wil11a] Stephen C. Williams. Analysis of the ssh key exchange protocol. In *IMA Int. Conf.*, volume 7089 of *Lecture Notes in Computer Science*, pages 356–374. Springer, 2011.

[Wil11b] Stephen C. Williams. On the security of key exchange protocols. In *Thesis*, 2011.