On the Cryptographic Security of Browser-Based Protocols

Florian Kohlar
(geboren in Bochum)

Dissertation zur Erlangung des Grades eines Doktor-Ingenieurs der Fakultät für Elektrotechnik und Informationstechnik an der Ruhr-Universität Bochum

Bochum, 2014

Author Contact Information:
florian.kohlar@rub.de

Date of oral exam: 05/07/2013
Thesis Advisor: Prof. Jörg Schwenk
Prof. Dennis Hofheinz

Dedication

Although the thesis is now written, it would not be complete without a dedication to all the people who accompanied my through this four-year journey.

I want to thank Prof. Schwenk for believing in me and giving me the chance to work at his chair and all (active and former) members of the chair of network- and data security in Bochum, who were a big support all those four years and became more friends than collegues. Working with you was a pleasure!

I want to dedicate this thesis to my parents, who gave me the possibility to study in the first place and supported me the whole time, and my wife Annette, who also believed in me and raised me up whenever I was doubting myself. I also want to thank my grandparents, who took care (whether I wanted it or not) of everything I could not handle myself in that time.

Last but not least I want to thank all members of the HGI in Bochum, who are showing until today that good cooperation and teamwork can achieve greatness.

"The Road goes ever on and on
Down from the door where it began.
Now far ahead the Road has gone,
And I must follow, if I can,
Pursuing it with eager feet,
Until it joins some larger way
Where many paths and errands meet.
And whither then? I cannot say."

The Lord of the Rings, J.R.R. Tolkien

Abstract

To ensure security on the internet, so-called Authenticated Key Exchange (AKE) protocols are executed before the transmission of sensitive data. The most important of these protocols is the **S**ecure **S**ocket **L**ayer (SSL) / **T**ransport **L**ayer **S**ecurity (TLS) protocol. It is used for example to secure HTTP connections, the foundation of the World Wide Web. The cryptographic security of the TLS protocol was a long openstanding problem, so that no formal statements about the security of TLS could be made. In consideration of the wide distribution of this protocol, this deficiency is especially critical for the state of work. This thesis eradicates this flaw in three steps:

At first the construction of secure AKE protocols is analyzed. The thesis shows how an AKE protocol with high security properties can be generically constructed from key agreement protocols with low security properties.

Then a new security model for Authenticated and Confidential Channel Establishment (ACCE) protocols is given. This model allows for the first time for a proof of the complete TLS protocol, including all standardized combinations of cryptographic algorithms. The thesis covers variants for both mutual authentication and server-only authentication. All results base on universally accepted cryptographic assumptions and do not make use of so-called *Random Oracles*.

Finally, the security of the TLS renegotiation protocol is analyzed, wich enables to negotiate fresh security parameters over previously established and secure channels. Up to now it was not known how such a protocol can be modeled formally. For that reason, a new suitable security model for secure multi-phase and renegotiable (ACCE) protocols is given first. This model is then used to analyze the security of the plain TLS renegotiation protocol and a variant implementing recent countermeasure against specific attacks. Based on these results, a new countermeasure is proposed, that allows for a proof of TLS renegotiation under the strongest of the new security definitions.

Contents

List of Figures

1 Introduction

This thesis is focused on the security of real-world cryptographic protocols[1], i.e. protocols that are used to *authenticate* parties and to *establish cryptographic keys*, which are then commonly used to enable a confidential transport of messages over insecure communication channels. 'Secure' authentication in this context means essentially that no malicious party *Eve* should be able to impersonate a different party *Alice* to some other party *Bob* (i.e. all parties should only be able to authenticate themselves to other parties) and secure key exchange means that when two parties Alice and Bob negotiate a session key used for some arbitrary security context subsequently, no third party *Eve* should gain any useful information about this key.

1.1 Focus of this Thesis

In this thesis we elaborate on important questions in the field of security of real-world cryptographic protocols, especially authenticated key exchange protocols. The main questions are:

1. How can we construct secure protocols for authenticated key exchange and how can we modify a protocol secure only against weak adversaries in order to be also secure against stronger adversaries?

2. How secure is the most important protocol for authenticated key exchange, the Transport Layer Security (TLS) protocol, in terms of provable security?

3. How secure are widely-used variants of this protocol, e.g. TLS renegotiation?

We will now describe the structure of this thesis, and highlight which chapter elaborates on which of the above questions.

In the remaining introduction we first describe the notion of 'authenticated key exchange' protocols in Section 1.2. We then briefly introduce different security models in Section 1.3 and discuss the benefits and limitations of security proofs in Section 1.4.

[1]Actually, interactive two-party protocols for authentication and key exchange.

We recall basic security definitions of important primitives in Chapter 2. These definition are partially common knowledge (i.e. have frequently appeared in the literature before) and partially taken from our published results. (The PRF-Oracle Diffie Hellman (PRF-ODH) assumption described in Section 2.9 is for example taken from [JKSS12]).

In order to analyze the security of protocols, we first have to define our security model. For specific reasons (that are later explained in detail) we introduce different models. In Chapter 3 we first describe a generic security model in Section 3.1, from which we then derive specific instantions for Authenticated Key Exchange (AKE) protocols in Section 3.2.1, Authenticated and Confidential Channel Establishment (ACCE) protocol in Section 3.2.2, and multi-phase and renegotiable ACCE protocols in Section 3.2.3. The specific instantions result from [JKSS10, JKSS12, GKS13].

In Chapter 4 we are going to answer the first question, namely how a secure AKE protocol can be constructed from basic primitives. We present a generic transformation from a passively-secure key exchange protocol to a secure authenticated key exchange protocol and prove the security of this protocol. This result was joint work with Tibor Jager, Sven Schäge and Jörg Schwenk and was published at ASIACRYPT'10 [JKSS10].

Chapter 5 focuses on the second question, concerning the provable security of TLS. In order to do so, we first give a brief overview of the TLS protocol in Section 5.1, and then analyze the security of TLS Ciphersuites with Static Diffie-Hellman-based Key Exchange (TLS-SDH), TLS Ciphersuites with Ephemeral Diffie-Hellman-based Key Exchange (TLS-DHE) and TLS Ciphersuites with RSA-based Key Transport (TLS-RSA), covering both server-only authentication and mutual authentication. We start in Section 5.2 with analyzing a truncated version of TLS-DHE in the AKE security model and then show in Sections 5.3 and 5.5 that the *unmodified* TLS-DHE is secure in the ACCE model and even provides perfect forward secrecy. We then analyze TLS-RSA and TLS-SDH in the ACCE model in Sections 5.6 to 5.9, showing that they are secure under an ACCE security definition without perfect forward secrecy. The results originate from a joint research paper with Tibor Jager, Sven Schäge and Jörg Schwenk which was published at CRYPTO'12 [JKSS12] and a paper with Sven Schäge and Jörg Schwenk that is currently in submission.

In Chapter 6 we then approach the third question and analyze the security of the TLS renegotiation protocol. We start by briefly describing the protocol and highlighting a recent attack. Then we analyze both the plain protocol and an extended variant that implements countermeasures against this attack in the model for multi-phase and renegotiation protocols. We also propose a new countermeasure that enables a proof in our strongest model for renegotiable ACCE protocols in Section 6.8. This chapter is based on joint work with Florian Giesen and Douglas Stebila and was published at CCS'13 [GKS13].

We conclude in Chapter 7 with a short discussion of the achieved results.

1.2 Authentication and Key Exchange Protocols

Protocols for *Authentication and Key Exchange* are an important part of the security infrastructure of the internet that we know today. E-Commerce is heavily relying on data being sent in a secure manner from one party to another, where a party can be a single person shopping on the internet (in which case the data may consist of the contents of the shopping basket and accounting information) as well as a company transmitting important information (where the data may contain company secrets). As different the use-cases may seem, the term 'in a secure manner' can be independently translated into the following security goals:

1. Messages between two parties must not be modified (or dropped) by a third party, even if this third party has full control over the communication medium, without the parties detecting the modifications.

2. The message content is to be transported confidentially, that is, a third party with full control over the network and access to the transported messages must not be able to learn any useful information about the message content.

3. Any party Eve should not be able to send messages to a party Alice on behalf of another (different) party Bob.

We refer to the above notions as 1) **Integrity**, 2) **Confidentiality** and 3) **Authenticity**.

Not always do applications require that all three of the above security goals are fulfilled, there may be different demands for e.g. online banking, electronic shopping, electronic voting or social networking. Some security requirements are more obvious than others. For example it may at first glance seem unimportant to have a high security standard in place for social networks (as social networking is basically all about making some private information public and communicating with friends rather than discussing business matters). An argument for the contrary may be that Facebook provides a login mechanism to remote, independent webservices over Facebook Connect, so the security of Facebook can directly influence the security of connected webservices.

Authenticated Key Exchange Protocols

The most common method to establish such a secure channel between two parties, even without sharing some prior distributed secret, is to run an Authenticated Key Exchange (AKE) protocol. The messages sent in this protocol are refered to as the 'handshake' messages, messages sent after the handshake has successfully terminated are refered to as 'application data'. We will use this notation in subsequent sections.

An AKE protocol basically consists of two components: First, a key exchange protocol that negotiates secure session keys. These keys are then used to encrypt messages with some symmetric encryption scheme under this key in order to provide confidentiality. The second component of an AKE protocol is a mean to authenticate parties (and eventually ensure message integrity during the session key negotiation). Note that integrity of application data can be implicitly provided by a subsequent symmetric encryption scheme. Authentication can for example be realized through the use of signature schemes, symmetric Message Authentication Codes (MACs) or even implicitly. Quite often it is also required, that just one party provides authentication, while the other party only 'profits' from the confidentiality.

The most prominent and important protocol implemented to protect data-flows on the internet today is Secure Socket Layer (SSL) which evolved into Transport Layer Security (TLS), the most recent version being TLS 1.2 [DR08]. For that reason we later in this thesis focus on the security of this protocol.

Attacks against AKE Protocols

Finding attacks against an AKE protocol by breaking integrity, confidentiality or authentication often comprises severe ramifications in the real world. Again the prime example is TLS — a successful attack that reveals the session keys negotiated by TLS would enable the adversary to make alterations to online banking communications and similar services implying a financial impact. Some work has been published on (modeling) different types of attacks [LLM07, Kra05, CK01].

When speaking of the security of AKE protocols one has to take into account the presence of active adversaries that reside between two parties and alter, drop or inject messages in a message flow between these two parties. We call this adversary also Man-In-The-Middle (MITM). Figure 1.1 displays three different attack scenarios that may occur in the presence of such MITM adversaries.

To prevent certain attacks against key exchange protocols, one could of course apply standard cryptographic primitives to turn the protocol into an AKE using

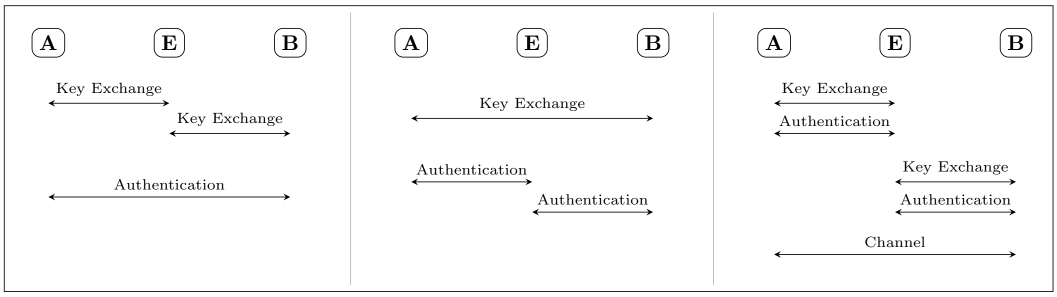

Figure 1.1: Different Man-in-the-middle attacks.
Parties Alice (**A**) and Bob (**B**) communicate in the presence of adversary Eve (**E**).

for example techniques from [BCK98, JKSS10]. When an analysis of a real-world protocol reveals such a vulnerability, altering the protocol may not be a valid choice, as e.g. in the cases cited above and also for TLS the implementation of deployed protocols can seldom be changed. Remark that failing to proof the security of a protocol in a given security model does not directly imply that this protocol is vulnerable in the real world. It may as well mean that the security model is too strong or the protocol does not fit to the security definitions. Imagine for example a three-party protocol and a security model for two-party protocols. It is obvious that the protocol can most likely not be proven secure in this model, but it is also quite clear that this ‘result’ cannot be translated in a security statement about the protocol.

1.3 An Introduction to Formal Models for Authentication and Key Exchange Protocols

We now give a brief overview of existing models for analyzing authentication and key exchange protocols. For easier comparison we concentrate on the following notions to distinguish the different models.

Partnering In the presence of passive adversaries a communication *partner* to some party Alice can be easily defined as an entity that, at the end of the protocol execution, has established a common session key. However, it is often helpful to be able to identify mutually communicating parties (i.e. partners) independent of the session key and before the protocol execution has finished.

Attacker Capabilities The strength of a model is closely tied to the strength of the adversary. The capabilities of an attacker commonly range from being able to send, modify, and drop messages exchanged between parties to learning secret values, such as negotiated session keys, long-term secrets and even intermediate session states (e.g. secret Diffie-Hellman exponents chosen during the protocol execution) and his restrictions on learning long-term keys and intermediate values.

Security goals The main goals of cryptographic protocols (especially of browser-based protocols) are to (mutually) authenticate parties and/or to negotiate 'secure' session keys. One could also say, a typical goal is to establish an authentic and confidential channel between two parties. However, such properties can be defined in many different ways, for example one can define notions for either explicit or implicit authentication of parties.

Bellare-Rogaway Model

Bellare and Rogaway basically started the line of research leading to formal models for analyzing the security of authentication and key exchange protocols [BR94a]. They are the first to define partnering via so-called 'matching conversations', i.e. secure, explicit authentication is given if the (ordered) set of received and sent messages matches for two parties engaging in a common protocol execution. Confidentiality is then defined by letting the adversary distinguish the session key of a 'fresh' session (i.e. a session in which the adversary did not actively participate) from a random key.

While this model did not allow for the adversary to learn long-term secret keys (more specifically the secret key of a public/secret key pair), Bellare and Rogaway 1995 adapted their model to the three-party case and enhanced the adversary with this capability [BR95]. However, their model was limited to three-party session-key distribution protocols, where one party distributes session keys between two other parties. Bellare, Pointcheval and Rogaway in 2000 introduced session IDs (wich could be defined similar to matching conversations by letting the session id be the concatenation of all messages sent and received) [BPR00].

Canetti-Krawczyk Model

Bellare, Canetti and Krawczyk in an independent work introduced a model that allowed for a two-step construction of secure authenticated key exchange protocols [BCK98]. They start by constructing their protocols in an idealized communication setting, i.e. in networks that for example guarantee message delivery and integrity protection. Then they describe a compiler that takes a protocol secure in the idealized setting and transforms it into a protocol secure in a more realistic setting (where an adversary may have certain control over all exchanged messages) They do so by adding so-called 'authenticators' or 'MT-authenticators'[2] (e.g. signatures). This work includes a first approach to modeling *session corruption*, i.e. to enable the adversary to learn interal values of parties specific to a session (state). This result was extended by a lot of follow-up papers, most established of wich is the work of Canetti and Krawcyzk [CK01] (which is the eponym for this class of models), which was later extended in many variants [LLM07, Oka07, Cre09, SEVB10, FSXY12].

There are two main differences between Canetti-Krawcyk-related (CK) models and Bellare-Rogaway-related (BR) models. First, CK models use session IDs to define partnering, which (dependent on the specific model) must be agreed upon **before** the protocol execution or can be arbitrarily defined. BR uses matching conversations to achieve the same. While session-ids may be more flexible, we think that the notion of matching conversations is better suited to the specific applications that we analyze in this thesis. Second, CK models do not explicitly define secure authentication. They merely define secure key exchange in unauthenticated networks, thus implicitly assuming a protection against adversarial impersonation attempts.

Univeral Composability

The Universal Composability (UC) framework described by Canetti [Can00, Can01] describes a completely modular approach to construct cryptographic protocols and analyze their security. Analyzing a primitive or protocol in this model first requires the definition of an 'ideal functionality'. This ideal functionality describes a protocol that, when executed by a trusted party in a secure environment, computes the output of the ideal functionality. The large benefit of this model is that once such a function has been proven secure, it can be used as a building block to construct more complex protocols, without needing a seperate security proof for the assembled construction. This is possible due to a 'composition theorem' described by Canetti. On the downside, the composition theorem assumes, similar to certain CK models, unique session IDs that need to be established before the protocol execution. The protocols that we focus on in this thesis, cryptographic protocols actually deployed in the real world, however typically do not use such pre-established session IDs. Hofheinz and Shoup have recently presented a similar framework to UC, called 'GNUC' (**G**enerally **N**ot **UC** [HS11]), motivated by their discovery that important definitions of the UC framework are flawed or at least problematic.[3] Although the CK model also aims at a modular construction

[2]MT stands for 'message transmission'.

[3]The original composition theorem by Canetti has been revised several times after its first publication due to such flaws.

and analysis of authentication and key exchange protocols, it is not comparable with the UC approach described below as in the UC framework corruptions of long-term secrets cannot be easily modeled.

For the analysis of complex real-world protocols, UC has some disadvantages. On the one hand, as mentioned above, globally unique session identifiers are required by the composition theorem. Although such session IDs can be established in many ways (see e.g. the compiler by Barak *et al.* [BLR04]), few protocols actually establish usable identifiers beforehand. On the other hand concurrent protocol runs cannot be analyzed in UC, if the analyzed protocols share some state value (e.g. a long-term key). When dividing a protocol into seperate ideal functionalities, this may pose a serious problem.

Remark 1. We remark, that this joint-state problem can be addressed, if a multi-session secure version of the authentication functionality (in the case of shared long-term keys) can be provided.

1.4 Benefits and Limits of Reduction-Based Security Proofs

In the following we will discuss the role of security proofs of cryptographic protocols (especially in the presence of real-world attacks).

Constructing a security proof requires 1) a model describing how protocol executions are simulated and what capabilities are given to the adversary and 2) a security definition formalizing what the adversary (given the previously described interaction capabilities) must achieve to break the protocol. The proof itself then tries to reduce **any** successful adversary against the protocol to solve an underlying problem, which is assumed to be hard.[4] If it can be shown that there exists no successful adversary against this protocol running in polynomial time, it is considered to be secure.

But even if a proof can be given, i.e. the protocol is secure under some definition, this does not guarantuee that this protocol is resilient against all future attacks. The reason is that, while trying to model the adversary as strong as possible by allowing him to learn internal states of the communicating parties, adversaries in the real world often succeed by exploiting sidechannels. Adversaries may for example exploit vulnerabilities in the implementation (see BEAST [RD11]), input or output paddings (see [Vau02]) and some real-world attacks have surfaced that exploit information about the timing of the protocol execution such as the generation of random values (see for example the attack of Jager *et al.* against XML Encryption with PKCS #1 [JSS12]). Distributing malware can also help

[4]By 'hard problem' we mean that the problem is assumed not to be solvable by any PPT adversary.

in many different ways to break the security properties of protocols, as in the example of the CRIME attack against TLS [RD12].

Standard Model Security vs. Random Oracles

Many security proofs rely on so-called 'Random Oracles', giving the communication partner access to an ideal function that (on any input) outputs a truly random output. If the same value is input twice, the same output is given (i.e. the function 'remembers' by keeping an internal database of previously asked inputs and corresponding outputs). Also, this functionality may only be accessed by two corresponding communication partners. While some results (for example most security results on RSA or RSA with Optimal Asymmetric Encryption Padding (RSA-OAEP)) only hold in the Random Oracle Model (ROM), see [Bro06], Canetti, Goldreich and Halevi have shown there exist cryptographic algorithms that are secure in the ROM, but for which any implementation of the Random Oracle with a cryptographic hash function results in insecure schemes [CGH98]. Despite this negative result, there exist examples of how Random Oracles can be instantiated for very specific applications, as for example shown by [HK09a, HJK11]

The usefulness of Random Oracles is still heavily discussed in the research community (see also [BR93, Den02, BBP04, KP09, KOS10]), so in this thesis we concentrate on the standard model security of protocols and do not make use of Random Oracles for any of our security results.

Formal Attacks against Protocols

There exist also some attacks that may not have consequences for protocols in the real world but invalidate any security proofs. For example protocols with security proofs given in models close to Bellare and Rogaway can be vulnerable to so-called Unknown Key Share Attacks (UKS), see [BWM99]. If the adversary is able to force two protocol participants into accepting the same session key while somehow violating the authentication condition, the adversay may 'query' one of the participants for the session key, thus directly breaking confidentiality. One way for the adversary to achieve this would be to remain passive, except when a certificate or a signed message is sent. The adversary then replaces the certificate/signature with his own certificate/signature, but **does not modify the signed content**. This attack is possible due to the way the security experiment is defined and cannot be explained at this point in detail. There exist several results on the limits of security proofs in existing models and Choo *et al.* gave a detailed comparison of several different models and their strengths [CBH05b, CBH05a, CBH05c].

1.5 List of Publications

In the following we give a list of my publications during my time as PhD student. All publications were joint work with the listed co-authors.

GJKS11: Florian Kohlar, Jörg Schwenk, Meiko Jensen, and Sebastian Gajek. On Cryptographically Strong Bindings of SAML Assertions to Transport Layer Security. IJMCMC, 3(4):20–35, 2011.

JKSS10: Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic Compilers for Authenticated Key Exchange. In Masayuki Abe, editor, Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings, volume 6477 of Lecture Notes in Computer Science, pages 232–249. Springer, 2010.

KSS10: Pavol Sovis, Florian Kohlar, and Jörg Schwenk. Security Analysis of OpenID. In Felix C. Freiling, editor, Sicherheit 2010: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 5. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 5.-7. Oktober 2010 in Berlin, volume 170 of LNI, pages 329–340. GI, 2010.

AKS11: Jörg Schwenk, Florian Kohlar, and Marcus Amon. The Power of Recognition: Secure Single Sign-On using TLS Channel Bindings. In Proceedings of the 7th ACM workshop on Digital identity management, DIM '11, pages 63–72, New York, NY, USA, 2011. ACM.

JKSS12: Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the Security of TLS-DHE in the Standard Model. In Reihaneh Safavi-Naini and Ran Canetti, editors, Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings, volume 7417 of Lecture Notes in Computer Science, pages 273–293. Springer, 2012.

GJKL11: Nils Gruschka, Meiko Jensen, Florian Kohlar, and Lijun Liao. On Interoperability Failures in WS-Security. In Ejub Kajan, editor, Electronic Business Interoperability: Concepts, Opportunities and Challenges, pages 615–635. IGI Global, 2011.

JKSS10*: Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic Compilers for Authenticated Key Exchange (Full Version). IACR Cryptology ePrint Archive, Report 2010/621, 2010. `http://eprint.iacr.org/2010/621`

JKSS12*: Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the Security of TLS-DHE in the Standard Model (Full Version). IACR Cryptology ePrint Archive, Report 2011/219, 2012. `http://eprint.iacr.org/2011/219`

GKS12: Florian Giesen, Florian Kohlar, and Douglas Stebila. On the Security of TLS Renegotiation. In Proceedings of the 20th ACM Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 04-08, 2013. `http://dx.doi.org/10.1145/2508859.2516694`

GKS12*: Florian Giesen, Florian Kohlar, and Douglas Stebila. On the Security of TLS Renegotiation (Full Version). IACR Cryptology ePrint Archive, Report 2012/630, 2013. `http://eprint.iacr.org/2012/630`

2 Preliminaries and Definitions

In this chapter, we present the formal definitions required to formulate our results. This chapter is not meant to be read from end to end, its main purpose is to be consulted whenever the reader encounters an unfamiliar primitive. Thus, readers with a background in provable security may skip this chapter entirely.

Notation We use $\emptyset$ to denote the empty string, and $[n] = [1, n] = \{1, \ldots, n\} \subset \mathbb{N}$ for the set of integers between 1 and n. If A is a set, then we use $a \overset{\$}{\leftarrow} A$ to denote that a is drawn uniformly random from A. In case A is a probabilistic algorithm $a \overset{\$}{\leftarrow} A$ is used to denote that A returns a when executed with fresh random coins. We use κ to denote the security parameter. $x \overset{\$}{\leftarrow} \mathcal{A}(y)$ denotes the output x of the probabilistic algorithm $\mathcal{A}$ when run on input y and randomly chosen coins.

2.1 The Decisional Diffie-Hellman Assumption

Let G be a group of prime order q (having bit-length polynomial in κ) and g a generator of G. The Decisional Diffie-Hellman (DDH) assumption states that if given (g, g^a, g^b, g^c) for $a, b, c \in \mathbb{Z}_q$ it is hard to decide whether $c = ab \bmod q$. More formally:

Definition 2.1. We say that the DDH problem is (t, ϵ)-*hard* in G, if for all adversaries $\mathcal{A}$ that run in time t it holds that

$$\left|\Pr\left[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1\right] - \Pr\left[\mathcal{A}(g, g^a, g^b, g^c) = 1\right]\right| \leq \epsilon,$$

where $a, b, c \overset{\$}{\leftarrow} \mathbb{Z}_q$.

2.2 Digital Signature Schemes

A digital signature scheme is a triple $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vfy})$, consisting of the key generation algorithm $(sk, pk) \overset{\$}{\leftarrow} \mathsf{SIG.Gen}(1^\kappa)$ generating a (public) verification key pk and a secret signing key sk on input of the security parameter κ, the signing algorithm $\sigma \overset{\$}{\leftarrow} \mathsf{SIG.Sign}(sk, m)$ generating a signature for message

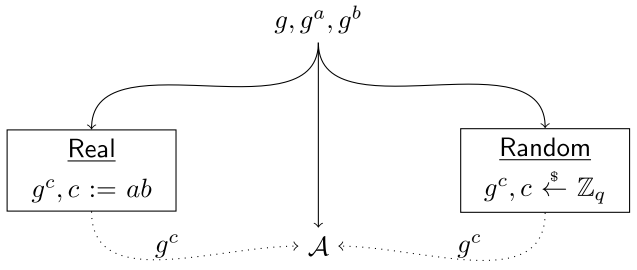

Figure 2.1: Illustration of the Decisional Diffie-Hellman Assumption

m, and the verification algorithm $\mathsf{SIG.Vfy}(pk, \sigma, m)$ returning 1, if σ is a valid signature for m under key pk, and 0 otherwise. Security is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates an asymmetric key pair $(sk, pk) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\kappa)$ and the public key pk is given to the adversary.

2. The adversary may adaptively query q messages m_i with $i \in [q]$ of his choice to the challenger. The challenger responds to each of these queries with a signature $\sigma_i = \mathsf{SIG.Sign}(sk, m_i)$ on m_i.

3. The adversary outputs a message/signature pair (m, σ).

Definition 2.2. We say that SIG is (t, ϵ, q)-*secure* against *existential forgeries under adaptive chosen-message attacks* (EUF-CMA), if for all adversaries $\mathcal{A}$ that run in time t making at most q queries it holds that

$$\Pr\left[(m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^\kappa, pk) \text{ such that } \mathsf{SIG.Vfy}(pk, m, \sigma) = 1 \wedge m \notin \{m_1, \dots, m_q\}\right] \leq \epsilon.$$

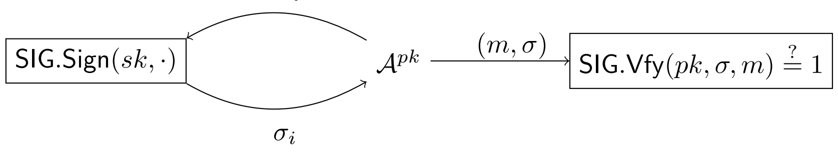
m_i
$\mathsf{SIG.Sign}(sk, \cdot)$
$\mathcal{A}^{pk}$
(m, σ)
$\mathsf{SIG.Vfy}(pk, \sigma, m) \stackrel{?}{=} 1$
σ_i

Figure 2.2: Illustration of the EUF-CMA security experiment of Definition 2.2

2.3 Symmetric Encryption Schemes

A symmetric encryption (SE) scheme is a pair $\mathsf{SE} = (\mathsf{SE.Enc}, \mathsf{SE.Dec})$, consisting of the encryption algorithm $\mathsf{SE.Enc}(k, m)$ generating a ciphertext c for message m under key k, and the deterministic decryption algorithm $\mathsf{SE.Dec}(k, c)$ returning m, if c is a valid encryption and the error symbol $\perp$ otherwise. Security is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The adversary may adaptively query the challenger for encryptions of arbitrary plaintexts m. The challenger responds to each of these queries with the output of $\mathsf{SE.Enc}(k, m)$.
2. The adversary outputs two (fresh) messages m_0 and m_1.
3. The challenger tosses coin $b \xleftarrow{\$} \{0, 1\}$. It then sets $c = \mathsf{SE.Enc}(k, m_b)$ and sends c to the adversary.
4. The adversary may again adaptively query plaintexts m of his choice, with the restriction that $m \notin \{m_0, m_1\}$. The challenger responds to each of these queries with the output of $\mathsf{SE.Enc}(k, m)$.
5. Finally the adversary outputs a bit b'.

Definition 2.3. We say that SE is (t, ϵ, q)-*secure* under Chosen-Plaintext Attacks (CPA), if all adversaries $\mathcal{A}$ that run in time t making at most q encryption queries have advantage of at most ϵ to distinguish the ciphertext of m_0 from that of m_1, i.e.

$$|\Pr[b = b'] - 1/2| \leq \epsilon.$$

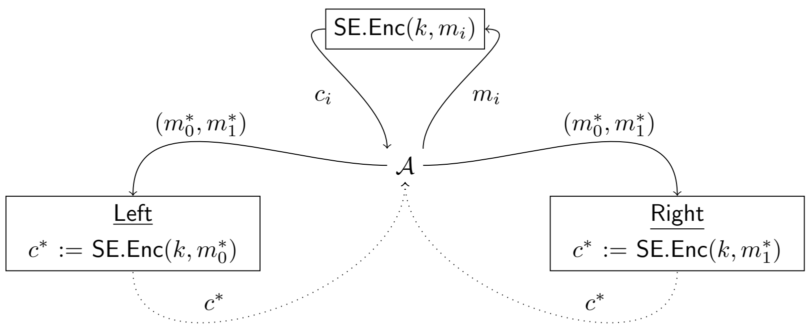
$\mathsf{SE.Enc}(k, m_i)$
c_i
m_i
(m_0^*, m_1^*)
(m_0^*, m_1^*)
$\mathcal{A}$
Left
$c^* := \mathsf{SE.Enc}(k, m_0^*)$
Right
$c^* := \mathsf{SE.Enc}(k, m_1^*)$
c^*
c^*

Figure 2.3: Illustration of the IND-CPA security experiment of Definition 2.3

2.4 Public Key Encryption Schemes

A Public-Key Encryption (PKE) scheme is a triple $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$, consisting of the key generation algorithm $(sk, pk) \xleftarrow{\$} \mathsf{PKE.Gen}(1^\kappa)$ generating a (public) encryption key pk and a secret decryption key sk on input of the security parameter κ, the probabilistic encryption algorithm $\mathsf{PKE.Enc}(pk, m)$

generating a ciphertext c for message m, and the deterministic decryption algorithm $\mathsf{PKE.Dec}(sk, c)$ returning m, if c is a valid encryption and the error symbol $\perp$ otherwise. Security is formalized in the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger generates an asymmetric key pair $(sk, pk) \overset{\$}{\leftarrow} \mathsf{PKE.Gen}(1^\kappa)$ and the public key pk is given to the adversary.

2. The adversary may adaptively query the challenger for decryptions of arbitrary ciphertexts c. The challenger responds to each of these queries with the output of $\mathsf{PKE.Dec}(sk, c)$.

3. The adversary outputs a message m^*.

4. The challenger tosses coin $b \overset{\$}{\leftarrow} \{0, 1\}$. It then sets $c_0 = \mathsf{PKE.Enc}(pk, m^*)$ and $c_1 = \mathsf{PKE.Enc}(pk, r)$ for a uniformly random message r that is of the same size as m^* and sends c_b to the adversary.

5. The adversary may again adaptively query ciphertexts c of his choice, now with the restriction that $c \neq c_b$. The challenger responds to each of these queries with the output of $\mathsf{PKE.Dec}(sk, c)$.

6. Finally the adversary outputs a bit b'.

Definition 2.4. We say that PKE is (t, ϵ, q)-*secure* under *adaptive* Chosen-Ciphertext Attacks (CCA), if all adversaries $\mathcal{A}$ that run in time t making at most q decryption queries have advantage of at most ϵ to distinguish the ciphertext of m^* from that of a truly random value, i.e.

$$|\Pr[b = b'] - 1/2| \leq \epsilon.$$

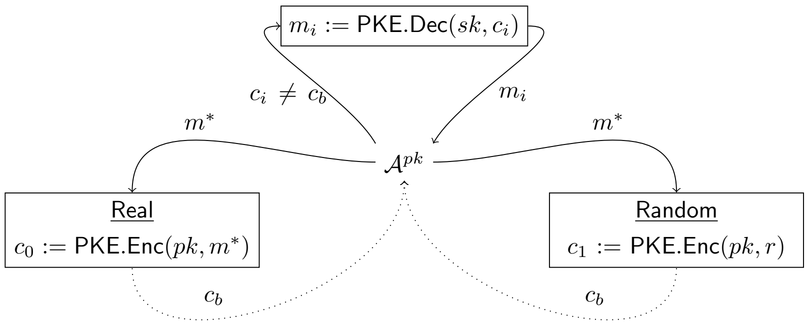
$m_i := \mathsf{PKE.Dec}(sk, c_i)$
$c_i \neq c_b$
m_i
m^*
m^*
$\mathcal{A}^{pk}$
Real
$c_0 := \mathsf{PKE.Enc}(pk, m^*)$
Random
$c_1 := \mathsf{PKE.Enc}(pk, r)$
c_b
c_b

Figure 2.4: Illustration of the IND-CCA security experiment of Definition 2.4

2.5 Pseudo-Random Functions

A Pseudo-Random Function (PRF) is a deterministic algorithm PRF which given a key $k \in \mathcal{K}_{\mathsf{PRF}}$ (with $\log(|\mathcal{K}_{\mathsf{PRF}}|)$ polynomial in κ) and a bit string x outputs a string $z = \mathsf{PRF}(k, x)$ with $z \in \{0,1\}^\mu$ (and μ being polynomial in κ). Let RF^μ be a truly random function that outputs a bitstring $z \in \{0,1\}^\mu$. Security is formulated via the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $k \xleftarrow{\$} \mathcal{K}_{\mathsf{PRF}}$ uniformly random and $b \xleftarrow{\$} \{0,1\}$.
2. The adversary may adaptively query q values x_i with $i \in [q]$ to the challenger. The challenger replies to each of these queries with either $z_i = \mathsf{PRF}(k, x_i)$ if $b = 0$ or $z_i \xleftarrow{\$} \mathsf{RF}^\mu$ if $b = 1$.
3. Finally, the adversary outputs its guess $b' \in \{0,1\}$ of b.

Definition 2.5. We say that PRF is a (t, ϵ, q)-*secure* pseudo-random function, if any adversary running in time t that makes at most q queries has an advantage of at most ϵ to distinguish the PRF from a truly random function, i.e.

$$|\Pr[b = b'] - 1/2| \leq \epsilon.$$

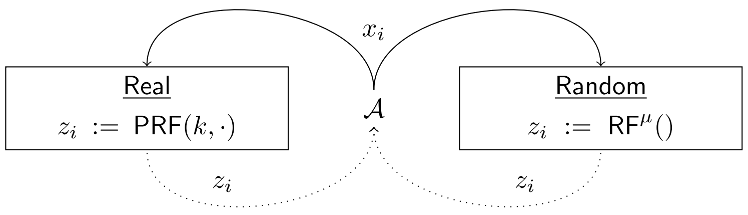

Figure 2.5: Illustration of the security experiment of Definition 2.5

Remark 2. In our security analysis we rely on the result by Fouque *et al.* [FPZ08] who showed that the PRF used in TLS 1.2 is secure with respect to the above definition if the compression function of the hash function used in the HMAC-based key-derivation function of TLS behaves like a pseudo-random function.

2.6 Message Authentication Codes

A MAC is an algorithm MAC. This algorithm implements a deterministic function $w = \mathsf{MAC}(K_{\mathsf{mac}}, m)$, taking as input a (symmetric) key $K_{\mathsf{mac}} \in \{0,1\}^\kappa$ and a message m, and returning a string w.

Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger samples $K_{\mathsf{mac}} \xleftarrow{\$} \{0,1\}^{\kappa}$ uniformly random.
2. The adversary may query arbitrary messages m_i to the challenger. The challenger replies to each query with $w_i = \mathsf{MAC}(K_{\mathsf{mac}}, m_i)$. Here i is an index, ranging between $1 \leq i \leq q$ for some $q \in \mathbb{N}$. Queries can be made adaptively.
3. Eventually, the adversary outputs a pair (m, w).

Definition 2.6. We say that MAC is a *(t, ϵ, q)-secure* message authentication code, if for all adversaries $\mathcal{A}$ that run in time t and queries at most q messages holds that

$$\Pr\left[(m,w) \xleftarrow{\$} \mathcal{A}^{\mathcal{C}}(1^{\kappa}) : w = \mathsf{MAC}(K_{\mathsf{mac}}, m) \wedge m \notin \{m_1, \ldots, m_q\}\right] \leq \epsilon_{\mathsf{MAC}}.$$

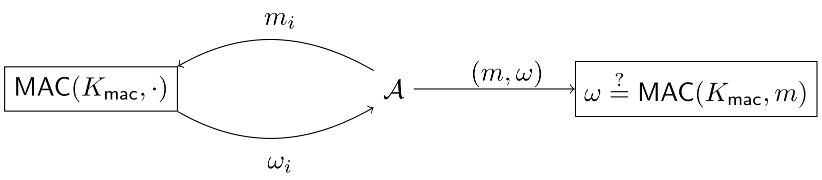
m_i
$\mathsf{MAC}(K_{\mathsf{mac}}, \cdot)$
$\mathcal{A}$
(m, ω)
$\omega \stackrel{?}{=} \mathsf{MAC}(K_{\mathsf{mac}}, m)$
ω_i

Figure 2.6: Illustration of the security experiment of Definition 2.6

2.7 Passively Secure Key Exchange Protocols

A passively secure Key Exchange (KE) protocol is a protocol KE run between two parties with the means to establish a common key k in the presence of a benign adversary. We assume, that when KE terminates, both parties have negotiated a common key k. Security is now defined by the probability of this adversary to distinguish the established key from a uniformly random key. We show this via the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The challenger executes KE between two honest parties, records the complete transcript T of messages exchanged and receives the negotiated key k.
2. The challenger samples $b \xleftarrow{\$} \{0,1\}$ uniformly random and sets $K_0 := k$ and samples $K_1 \xleftarrow{\$} \{0,1\}^{\mathcal{K}}$ uniformly at random. Then it returns K_b and the complete transcript T to the adversary.

3. Finally, the adversary outputs its guess $b' \in \{0,1\}$ of b.

Definition 2.7. We say that a key exchange protocol KE is (t,ϵ)-*secure* against passive adversaries, if any passive adversary running in time t has an advantage of at most ϵ to distinguish the established key k from a truly random key $\tilde{k}$, i.e.

$$|\Pr[b = b'] - 1/2| \leq \epsilon.$$

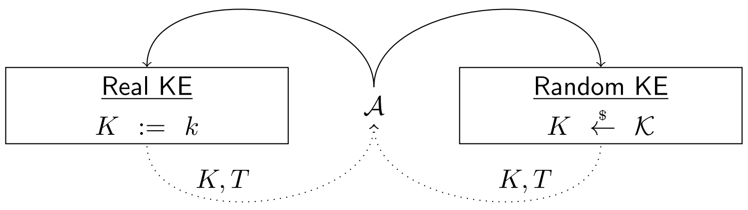

Figure 2.7: Illustration of the security experiment of Definition 2.7

2.8 Collision-Resistant Hash Functions

A *collision-resistant hash function* is a deterministic algorithm H which given a key $k \in \mathcal{K}_\mathsf{H}$ (with $\log(|\mathcal{K}_\mathsf{H}|)$ polynomial in κ) and a bit string m outputs a hash value $w = \mathsf{H}(k,x)$ in the hash space $\{0,1\}^\chi$ (with χ polynomial in κ). If k is clear from the context we write $\mathsf{H}(\cdot)$ short for $\mathsf{H}(k,\cdot)$.

Definition 2.8. We say that H is a (t,ϵ)-*secure* collision-resistant hash function, if any t-time adversary $\mathcal{A}$ that is given $k \xleftarrow{\$} \mathcal{K}_\mathsf{H}$ has an advantage of at most ϵ to compute two colliding inputs m, m' with $m \neq m'$ and $\mathsf{H}(m) = \mathsf{H}(m')$.

2.9 The PRF-Oracle-Diffie-Hellman Assumption

Let G be a group with generator g of order q' (having bit-length polynomial in κ). Let PRF be a deterministic function $z = \mathsf{PRF}(X,m)$, taking as input a key $X \in G$ and some bit string m, and returning a string $z \in \{0,1\}^\mu$. Consider the following security experiment played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The adversary $\mathcal{A}$ outputs a value m.
2. The challenger samples $u, v \xleftarrow{\$} [q]$, $z_1 \xleftarrow{\$} \{0,1\}^\mu$ uniformly random and sets $z_0 := \mathsf{PRF}(g^{uv}, m)$. Then it tosses a coin $b \in \{0,1\}$ and returns z_b, g^u and g^v to the adversary.
3. The adversary may query a pair (X, m') with $X \neq g^u$ to the challenger. The challenger replies with $\mathsf{PRF}(X^v, m')$.

4. Finally the adversary outputs a guess $b' \in \{0,1\}$.

Definition 2.9. We say that the PRF-ODH problem is *(t, ϵ)-hard* with respect to G and PRF, if for all adversaries $\mathcal{A}$ that run in time t it holds that

$$|\Pr[b = b'] - 1/2| \leq \epsilon.$$

The PRF-ODH assumption is a variant of the Oracle Diffie-Hellman (ODH) assumption introduced by Abdalla, Bellare and Rogaway in [ABR01], adopted from hash functions to PRFs and restricted to allow only a single oracle query.

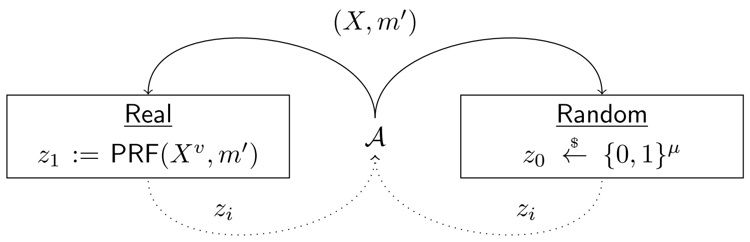
(X, m')
Real
$z_1 := \mathsf{PRF}(X^v, m')$
$\mathcal{A}$
Random
$z_0 \xleftarrow{\$} \{0,1\}^\mu$
z_i
z_i

Figure 2.8: Illustration of the security experiment of Definition 2.9

2.10 The Strong PRF-Oracle-Diffie-Hellman Assumption

Let G be a group with generator g of order q' (having bit-length polynomial in κ). Let PRF be a deterministic function $z = \mathsf{PRF}(X, m)$, taking as input a key $X \in G$ and some bit string m, and returning a string $z \in \{0,1\}^\mu$. Consider the following security game that is played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$.

1. The adversary $\mathcal{A}$ outputs a value m.
2. The challenger samples $u, v \xleftarrow{\$} [q']$ and $z_1 \xleftarrow{\$} \{0,1\}^\mu$ uniformly random and sets $z_0 := \mathsf{PRF}(g^{uv}, m)$. Then it tosses a coin $b \in \{0,1\}$ and returns z_b, g^u and g^v to the adversary.
3. The adversary may adaptively query q pairs (X_i, m'_i) with $(X_i, m'_i) \neq (g^u, m)$, $i \in [q]$ to the challenger. The challenger replies with $\mathsf{PRF}(X_i^v, m'_i)$.
4. Finally the adversary outputs a guess $b' \in \{0,1\}$.

Definition 2.10. We say that the Strong PRF-ODH problem is (t, ϵ)-*hard* for G and PRF, if for all adversaries $\mathcal{A}$ that run in time t making at most q queries in the above security game it holds that

$$|\Pr[b = b'] - 1/2| \leq \epsilon.$$

The definition of the Strong PRF-ODH problem differs in two ways. First, we allow the adversary to ask a polynomial number of queries to the oracle in contrast to only a single query. This makes our assumption more similar to the original Oracle Diffie-Hellman assumption. Second, we also allow queries of different messages m'_i with the same key g^u, as long as $m'_i \neq m$. This makes our assumption much more comparable with the classical security definition of PRFs, as now the oracle can also be queried for the challenge key more than once.

2.11 Stateful Length-Hiding Authenticated Encryption

We now define a stateful variant of the definition for length-hiding authenticated encryption (LHAE) which is attributed to Paterson *et al.* [PRS11].

A (symmetric) *stateful length-hiding authenticated encryption (sLHAE) scheme* consists of two algorithms $\mathsf{StE} = (\mathsf{StE.Enc}, \mathsf{StE.Dec})$. The (possibly probabilistic) encryption algorithm, given as $(C, st'_e) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m, st_e)$, takes the following values as input: the secret key $k \in \{0,1\}^\kappa$, the length $\mathsf{len} \in \mathbb{N}$ of the output ciphertext, header data $H \in \{0,1\}^*$, the plaintext $m \in \{0,1\}^*$, and the current state $st_e \in \{0,1\}^*$ of the encryption scheme. It outputs either a ciphertext $C \in \{0,1\}^{\mathsf{len}}$ and the updated state st'_e or the special error symbol $\perp$. The deterministic decryption algorithm $(m', st'_d) = \mathsf{StE.Dec}(k, H, C, st_d)$ processes secret key k, header data H, ciphertext C, and the current state $st_d \in \{0,1\}^*$. It returns the updated state st'_d and a value m' which is either the message encrypted in C, or a distinguished error symbol $\perp$ indicating that C is not a valid ciphertext. The encryption state st_e and decryption state st_d are initialized to the empty string $\emptyset$.

Consider the following security game that is played between challenger $\mathcal{C}$ and adversary $\mathcal{A}$.

1. The challenger draws $b \xleftarrow{\$} \{0,1\}$ and $k \xleftarrow{\$} \{0,1\}^\kappa$ and sets $st_e := \emptyset$ and $st_d := \emptyset$.
2. The adversary may adaptively query the encryption oracle $\mathsf{Encrypt}$ q_e times and the decryption oracle $\mathsf{Decrypt}$ q_d times. Figure 2.9 shows how these oracles respond to $\mathcal{A}$'s queries.
3. Finally, the adversary outputs a guess $b' \in \{0,1\}$.

Definition 2.11. We say that an Stateful Length-Hiding Authenticated Encryption (sLHAE) scheme $\mathsf{StE} = (\mathsf{StE.Init}, \mathsf{StE.Enc}, \mathsf{StE.Dec})$ is (t, ϵ)-*secure*, if for all

```
$\mathsf{Encrypt}(m_0, m_1, \mathsf{len}, H)$:
$u := u + 1$
$(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m_0, st_e)$
$(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, m_1, st_e)$
If $C^{(0)} = \perp$ or $C^{(1)} = \perp$ then return $\perp$
$(C_u, H_u, st_e) := (C^{(b)}, H, st_e^{(b)})$
Return $C_u$
```

```
$\mathsf{Decrypt}(C, H)$:
$v := v + 1$
If $b = 0$, then return $\perp$
$(m, st_d) = \mathsf{StE.Dec}(k, H, C, st_d)$
If $v > u$ or $C \neq C_v$ or $H \neq H_v$
    then $\mathsf{phase} := 1$
If $\mathsf{phase} = 1$ then return $m$
Return $\perp$
```

Figure 2.9: Encrypt and Decrypt oracles in the stateful LHAE security experiment. The values u, v and phase are all initialized to 0 at the beginning of the security game.

adversaries $\mathcal{A}$ that run in time t it holds that

$$|\Pr[b = b'] - 1/2| \leq \epsilon.$$

in the above security game. We assume that the number of encryption queries q_e and decryption queries q_d is bound by the running time of the adversary.

3 Formal Security Models and Definitions for Cryptographic Protocols

In this chapter we describe several security models for a wide range of different protocol types. Our main focus lies on the security of two-party protocols for authentication and key exchange in various instantiations in the presence of active adversarys.

An important line of research [BWJM97, CK01, LLM07] dates back to Bellare and Rogaway [BR94a], where an adversary is provided with an 'execution environment', which emulates the real-world capabilities of an active adversary. This approach has become fairly standard in the analysis of cryptographic protocols. Technically, we model the capabilities of an adversary through queries that he may ask to an execution environment implemented by a challenger. An adversary has full control over the communication network, which allows him to forward, alter, or drop any message sent by the participants, or insert new messages.

We start with defining a generic execution environment and security model in Section 3.1. We then take this security model and adopt it in Section 3.2 to give several concrete security models for different types of protocols. More specific, we give concrete security models for Authenticated Key Exchange (AKE) protocols in Section 3.2.1, Authenticated and Confidential Channel Establishment (ACCE) protocols in Section 3.2.2, and multi-phase (ACCE) protocols in Section 3.2.3.

Usually, models for two-party authentication and authenticated key exchange protocols only cover the case where both protocol participants authenticate to each other, which is called mutual authentication. In practice, however, it is fairly common, that only one party authenticates itself cryptographically during the protocol execution, while the other party, if required, later performs an out-of-band authentication (e.g. by a username/password combination over the freshly established confidential channel). We cover both cases and give definitions for mutual authentication protocols and one-sided authentication protocols.

3.1 Generic Security Model

In this section we describe a generic security model, that we use as basis for the specific instantiations in the subsequent sections. The generic security model is basically the intersection of all specific security models, defining the prime simulation set-up for all subsequently refined execution environments and the basic queries that an adversary is always provided with.

In the real world protocols are executed between many individual parties and each party may communicate in multiple sessions with an arbitrary set of other parties. To be able to make formal statements about the security of a protocol in such a distributed enviroment and in the presence of an adversary, it is common to model the adversary as a **p**robabilistic machine running in **p**olynomial **t**ime (also refered to as Probabilistic Polynomial-Time (PPT)-adversary) The adversary interacts with a single entity, the *simulator* $\mathcal{S}$, that implements all parties. We also call this simulation the *execution environment*. Note that the simulator can efficiently implement all parties (and oracles) since it also implements all states and thus all (long-term) secrets. An adversary in the real world may 'tap the wire', i.e. intercept, modify or drop messages, or use specific attacks to learn secret values (e.g. by infecting a party with malware he might learn the long-term key used by this party or the session negotiated between this party and some other party). The interaction between the adversary and the (simulated) parties is described by *queries*, that try to approximate realistic security threats in practice. Note, that the simulation represents an idealized world in which certain problems of the real world, such as (untriggered) packet losses and implementation errors, do not exist.

3.1.1 Generic Execution Environment

We now describe a generic execution environment, i.e. how the simulator implements parties and sessions. We later refine this generic environment to allow us to simulate the protocols we want to analyze.

In the following let $\phi, \eta \in \mathbb{N}$ be positive integers. We consider a set of ϕ parties $\{P_1, \ldots, P_\phi\}$. Each party $P_i \in \{P_1, \ldots, P_\phi\}$ is a (potential) protocol participant with a specific role in the protocol execution specified by variable ρ[5] that has access to some long-term secret L_i, maintained in its internal state.

Remark 3. We stress that this also models scenarios where the set of parties may

[5]The set of possible roles depends on the protocol specification.

intially consist of one party only and is step-wisely filled by the environment, since the long-term secrets are generated independently of the adversary. It makes no difference if the long-term secrets are generated at setup or on demand. We only require that the total number of parties is at most ϕ.

Remark 4. L_i might contain a single secret key corresponding to a public key/secret key pair or a set of multiple secret keys. This will later be detailed in the extended execution environments for the different protocol types.

We assume that each party P_i is associated with η oracles $\pi_i^1, \dots, \pi_i^\eta$, where each oracle π_i^s represents a process that executes one single instance of the protocol, also called a session. The oracles $\pi_i^1, \dots, \pi_i^\eta$ of party P_i all have access to the long-term secret L_i of party P_i.[6] Moreover, each oracle π_i^s maintains as internal state the following variables:

- $\Lambda \in \{\mathtt{accept}, \mathtt{reject}\}$.
- $k \in \mathcal{K}$, where $\mathcal{K}$ is the keyspace of the considered protocol.
- $\mathit{\Pi} \in [\phi]$ holding the intended communication partner, i.e. an index j that points to a globally unique party P_j.
- $\mathsf{T}^{i,s}$, the ordered transcript of all messages sent and received by the oracle.
- Some additional temporary state variable st (which may, for instance, be used to store ephemeral Diffie-Hellman exponents or counters of a stateful encryption scheme).

In the following, let $\pi_i^s.V$ denote the content of variable V stored at oracle π_i^s. The long-term secret L_i for each party is initialized depending on the protocol specification. The internal state of each oracle is initialized to $(\Lambda, k, \mathit{\Pi}, \rho, \mathsf{T}, st) = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$, where $V = \emptyset$ denotes that variable V is undefined. Variables $\mathit{\Pi}$ and ρ are set depending on the protocol specification. Furthermore, we will always assume (for simplicity) that $k = \emptyset$ if an oracle has not reached `accept`-state (yet), and contains the computed key if an oracle is in `accept`-state, so that we have

$$k \neq \emptyset \iff \Lambda = \text{`}\mathtt{accept}\text{'}. \tag{3.1}$$

Generic Adversarial Capabilities

We now describe basic interaction capabilities of an adversary $\mathcal{A}$, which we extend in the following section to capture certain protocol-specific attacks. The

[6]In our model we say for simplicity that for each oracle π_i^s of party P_i the role ρ is set to the role of P_i.

adversary can participate at any protocol execution by asking 'queries' to oracles, which exactly define and restrict the amount of interaction between the adversary and the environment. As mentioned above, we here only describe the basic queries, which are independent of the protocol type, and the internal setup of the simulator.

Informally, we want to give the adversary the ability to exchange messages between oracles, learn long-term secrets of parties, and learn session keys negotiated between two oracles. The Send query enables the adversary to initiate and run an arbitrary number of protocol instances, sequential or in parallel, and provides full control over the communication between all parties. The Reveal query may be used to learn the session keys used in previous/concurrent protocol executions. The Corrupt query allows the adversary to learn the long-term secret L_i of party P_i and may for instance be used by $\mathcal{A}$ to impersonate P_i.

Formally, an adversary may interact with the oracles described above by issuing the following queries to the execution environment.

- $\mathsf{Send}(\pi_i^s, m)$: The adversary can use this query to send message m to oracle π_i^s. The oracle will respond according to the protocol specification, depending on its internal state. If the adversary asks the first Send query to oracle π_i^s, then the oracle checks whether $m = \top$ consists of a special 'initialization' symbol $\top$. If true, then it responds with the first protocol message.

  The variables $\Lambda, k, \mathit{\Pi}, \rho, st$ are also set after some Send query. When and how depends on the considered protocol.

- $\mathsf{Reveal}(\pi_i^s)$: Oracle π_i^s responds to a Reveal query with the contents of variable k. Note that we have $k \neq \emptyset$ if and only if $\Lambda = \mathtt{accept}$, see (3.1) above.

- $\mathsf{Corrupt}(P_i, [\cdot])$: Oracle π_i^1 responds with the long-term secret L_i of party P_i.[7] The second parameter is optional and may be used to learn only parts of the long-term secret.[8]

  If $\mathsf{Corrupt}(P_i)$ is the τ-th query issued by $\mathcal{A}$ we say that P_i is *τ-corrupted.* For parties that are not corrupted we define $\tau := \infty$.

Depending on the protocol type, the adversary may have access to further queries, which are specified later in the appropriate sections. For example, to show security of Authenticated Key Exchange (AKE) protocols, the adversary is

[7]Note, that the adversary does not 'take control' of oracles corresponding to a corrupted party. But he learns the long-term secret key, and can henceforth simulate oracles of these parties.

[8]This is to simulate partial leakage. If a party has access to multiple long-term secret keys, the adversary may choose to only corrupt a specific key instead of corrupting all keys at once.

provided with a Test query which outputs a challenge that the adversary has to answer (see Section 3.2.1 for details on this query).

3.1.2 Matching Conversations

Bellare and Rogaway [BR94a] have introduced the notion of *matching conversations* in order to define correctness and security of an AKE protocol precisely. We adapt this notion and use it accordingly to define secure authentication for all following protocol types.

Recall, that $\mathsf{T}^{i,s}$ consists of all messages sent and received by π_i^s in chronological order (not including the initialization-symbol $\top$). We also say that $\mathsf{T}^{i,s}$ is the *transcript* of π_i^s. For two transcripts $\mathsf{T}^{i,s}$ and $\mathsf{T}^{j,t}$, we say that $\mathsf{T}^{i,s}$ is a *prefix* of $\mathsf{T}^{j,t}$, if $\mathsf{T}^{i,s}$ contains at least one message, and the messages in $\mathsf{T}^{i,s}$ are identical to and in the same order as the first $|\mathsf{T}^{i,s}|$ messages of $\mathsf{T}^{j,t}$.

Definition 3.1 (Matching conversations)**.** We say that π_i^s has a *matching conversation* to π_j^t, if

- $\mathsf{T}^{j,t}$ is a prefix of $\mathsf{T}^{i,s}$ and π_i^s has sent the last message(s), or
- $\mathsf{T}^{i,s} = \mathsf{T}^{j,t}$ and π_j^t has sent the last message(s).

We say that two processes π_i^s and π_j^t have *matching conversations* if π_i^s has a *matching conversation* to process π_j^t, and vice versa.

Remark 5. We remark that matching conversations in the above sense can also be seen as *post-specified session identifiers*.[9] The 'asymmetry' of the definition (i.e. the fact that we have to distinguish which party has sent the last message) is necessary, due to the fact that protocol messages are sent sequentially. For instance in the TLS Handshake protocol (see Figure 5.1 in Section 5.1) the last message of the client is the (encrypted) `ClientFinished` message fin_C. After sending this message, the client waits for the (encrypted) `ServerFinished` message fin_S before 'accepting'. In contrast, the server sends fin_S *after* receiving fin_C. Therefore the server has to 'accept' without knowing whether its last message was received by the client correctly. We have to take this into account in the definition of matching conversations, since it will later be used to define security of the protocol in presence of an active adversary that simply drops the last protocol message.

[9]The post-specified peer model was introduced by Canetti and Krawczyk, for details see [CK02].

3.2 Specific Instantiations

We now extend the above execution environment and the capabilites of an adversary for AKE protocols in Section 3.2.1, Authenticated and Confidential Channel Establishment (ACCE) protocols in Section 3.2.2, and multiphase and renegotiable ACCE protocols in Section 3.2.3.

3.2.1 Authenticated Key Exchange Protocols

In this section we refine the generic execution environment presented in the previous section to formalize the security of AKE protocols. Note, that we will only consider AKE protocols in the public key setting here, although it is trivial to modify our model to cover pre-shared symmetric keys.

Extended Execution Environment

We now say that the long-term secrets of each party $P_i \in \{P_1, \ldots, P_\phi\}$ consist of a secret key corresponding to a unique long-term key pair (pk_i, sk_i), thus $L_i = sk_i$.[10] Additionally, all oracles may assume exactly one of two roles in a protocol execution, being either Client or Server. We say that the oracle sending the first message in a protocol execution is the client and its partner oracle is the server. This means that for all oracles it holds that $\rho \in \{\mathsf{Client}, \mathsf{Server}\}$.

Remark 6. Usually, we speak of the first oracle being the *Initiator* and the second oracle being the *Responder*. However, as we later use this model to analyze the security of the TLS protocol, we will use the notation $\mathsf{Client}/\mathsf{Server}$ to better reflect the roles in this protocol.

Extended Queries The queries available to the adversary in the AKE setting are identical to the queries detailed in Section 3.1.1, except for the following modifications and additions: We now define in detail, how and when the role ρ of a party is set after a Send query, and we introduce a Test query in order to be able to define AKE security.

- $\mathsf{Send}(\pi_i^s, m)$: This query is identical to the Send query from Section 3.1.1, with the following additions:

  If this query is the first Send query to an oracle π_i^s of party P_i and $m = \top$, then P_i sets its internal variable $\rho := \mathsf{Client}$ and π_i^s responds with the first

[10]This is just for simplicity. We could easily extend our model to cover situations where each party may have more than one long-term key pair.

protocol message. Otherwise it sets $\rho :=$ Server and π_i^s responds as specified in the protocol.[11]

- Test(π_i^s): This query may be asked only once throughout the game. If π_i^s has state $\Lambda \neq$ `accept`, then it returns some failure symbol $\perp$. Otherwise it flips a fair coin b, samples an independent key $k_0 \xleftarrow{\$} \mathcal{K}$, sets $k_1 := k$ to the 'real' key computed by π_i^s, and returns k_b.

Correctness and Security Definition

Correctness of AKE protocols is in the following defined by requiring that all protocol participants, in the presence of benign adversarys, 'accept' and compute the same session key. Security of AKE protocols is then defined by requiring that (i) the protocol is a secure authentication protocol, and (ii) the protocol is a secure key-exchange protocol, thus an adversary cannot distinguish the session key k from a random key. The first requirement differs for server-only and mutual authentication, whereas it is interesting to note that the second requirement is independent of the chosen authentication mode.

AKE Game. We formally capture this notion as a game, played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The challenger implements the collection of oracles $\{\pi_i^s : i \in [\phi], s \in [\eta]\}$. At the beginning of the game, the challenger generates ϕ long-term key pairs (pk_i, sk_i) for all $i \in [\phi]$. The adversary receives the public keys $pk_1, \ldots, pk_\phi$ as input.

Now the adversary may start issuing Send, Reveal and Corrupt queries, as well as one Test query. Finally, the adversary outputs a bit b' and terminates.

Definition 3.2. Assume a 'benign' adversary $\mathcal{A}$, which picks two arbitrary oracles π_i^s and π_j^t and performs a sequence of Send queries by faithfully forwarding all messages between π_i^s and π_j^t, until the last message according to the protocol specification has been sent. Let k_i^s denote the key computed by π_i^s and let k_j^t denote the key computed by π_j^t.

We say that an AKE protocol is *correct*, if for this benign adversary and any two oracles π_i^s and π_j^t always holds that

1. both oracles have $\Lambda =$ `accept`,
2. $k_i^s = k_j^t \in \mathcal{K}$.

[11] Note that we do not include the identity of the (intended) communication partner in the Send query. Instead, we assume that the exchange of identities of communication partners (which is necessary to determine the public key used to perform authentication) is part of the protocol.

Mutual Authentication. In the following we give the definitions for mutually authenticated key exchange protocols with and without perfect forward secrecy.

In Property 1 we require that the protocol is a secure authentication protocol, i.e. for each oracle that accepts with an intended uncorrupted partner, there must exist a partner oracle of this uncorrupted party that has a matching conversation. Note that this party may still be corrupted later.

In Property 2 we require that the adversary cannot distinguish the session key k computed in a chosen 'Test session' between uncorrupted parties from a random key. For protocols with perfect forward secrecy we allow the adversary to corrupt parties *after* they have accepted.

Definition 3.3. We say that an adversary (t, ϵ)-*breaks* an AKE protocol with mutual authentication, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:

1. When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s with internal state $\Lambda = \texttt{accept}$ such that
   - π_i^s 'accepts' with intended partner $\Pi = j$ when $\mathcal{A}$ issues its τ_0-th query, and
   - P_j is τ_j-corrupted with $\tau_0 < \tau_j$,[12] and
   - there is no unique oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

   If an oracle π_i^s accepts in the above sense, then we say that π_i^s accepts *maliciously.*

2. When $\mathcal{A}$ issues a Test query to any oracle π_i^s and
   - π_i^s 'accepts' with intended partner $\Pi = j$ when $\mathcal{A}$ issues its τ_0-th query,
   - P_i and P_j are τ-corrupted with $\tau = \infty$, and
   - $\mathcal{A}$ does not issue a Reveal query to π_i^s, nor to π_j^t such that π_i^s has a matching conversation to π_j^t (if such an oracle exists),

   then the probability that $\mathcal{A}$ outputs b' which equals the bit b sampled by the Test query satisfies
   $$|\Pr[b = b'] - 1/2| \geq \epsilon.$$

   If an adversary $\mathcal{A}$ outputs b' such that $b' = b$ and the above conditions are met, then we say that $\mathcal{A}$ *answers the* Test*-challenge correctly.*

[12]That is, P_j is not corrupted when π_i^s 'accepts'. Recall that uncorrupted parties are τ-corrupted with $\tau = \infty$.

Definition 3.4. We say that an adversary (t, ϵ)-*breaks* an AKE protocol with mutual authentication and perfect forward secrecy, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:

1. When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s that accepts *maliciously* in the sense of Property 1 of Definition 3.3.
2. When $\mathcal{A}$ issues a Test query to any oracle π_i^s and
   - π_i^s 'accepts' with intended partner $\Pi = j$ when $\mathcal{A}$ issues its τ_0-th query,
   - P_j is τ_j-corrupted with $\tau_0 < \tau_j$, and
   - $\mathcal{A}$ does not issue a Reveal query to π_i^s, nor to π_j^t such that π_i^s has a matching conversation to π_j^t (if such an oracle exists),

   then the probability that $\mathcal{A}$ outputs b' which equals the bit b sampled by the Test query satisfies
   $$|\Pr[b = b'] - 1/2| \geq \epsilon.$$

Server-Only Authentication. In the following we modify the above definitions to define server-only authenticated key exchange protocols with and without perfect forward secrecy. Note, that the only difference to the above definitions with mutual authentication is that we restrict the *maliciously* accepting oracle to be a Client, i.e. to have internal state $\rho = \mathsf{Client}$.

Definition 3.5. We say that an adversary (t, ϵ)-*breaks* an AKE protocol with server-only authentication, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:

1. When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s such that π_i^s *maliciously* accepts in the sense of Property 1 of Definition 3.3, and π_i^s has internal state $\rho = \mathsf{Client}$.
2. $\mathcal{A}$ answers the Test-challenge correctly in the sense of Property 2 of Definition 3.3 and either
   - oracle π_i^s (to which $\mathcal{A}$ has issued the Test query) has internal state $\rho = \mathsf{Client}$, or
   - oracle π_i^s has internal state $\rho = \mathsf{Server}$, and there exists an oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

Definition 3.6. We say that an adversary (t, ϵ)-*breaks* an AKE protocol with server-only authentication and perfect forward secrecy, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:

1. When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s that accepts *maliciously* in the sense of Property 1 of Definition 3.5.

2. $\mathcal{A}$ answers the Test-challenge correctly in the sense of Property 2 of Definition 3.4 and either
   - oracle π_i^s (to which $\mathcal{A}$ has issued the Test query) has internal state $\rho = \mathsf{Client}$, or
   - oracle π_i^s has internal state $\rho = \mathsf{Server}$ and $\Pi = j$, and there exists an oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

We say that an AKE protocol is (t, ϵ)-*secure* (with respect to any of the above definitions), if it is correct and there exists no adversary that (t, ϵ)-*breaks* it (with respect to the same definition).

Remark 7. Note that all of the above definitions also model
acrshortkci attacks [BWJM97, Kra05], since we do not require that the client P_i is uncorrupted. Even if P_i is corrupted, it must be impossible for an adversary to impersonate party P_j to P_i.

Remark 8. Client-only security can be defined analogously by switching the internal state $\rho = \mathsf{Client}$ to $\rho = \mathsf{Server}$ in the appropriate definitions.

It is easy to see that if π_i^s has a matching conversation to π_j^t when only allowing server authentication and π_j^t has a matching conversation to π_i^s when only allowing client authentication then π_i^s and π_j^t have matching conversations to each other and the protocol is AKE secure with mutual authentication. However, the converse is not true in general.

3.2.2 Authenticated and Confidential Channel Establishment (ACCE) Protocols

Provably secure authenticated key exchange protocols as described in the previous section provide sessions keys which are indistinguishable from random strings (chosen from the same keyspace). While this approach is very appealing, due to the fact that a key derived in such a protocol execution can be used without limitations for all purposes, this notion is sometimes too strong for specific protocol variants. Namely, when the session key is already used in some manner **during** the protocol execution[13]. Formally, such a key cannot be proven indistinguishable from random — an adversary can simply take its challenge key (obtained

[13]This may not be true for all protocols that make use of the negotiated session key, e.g. for key confirmation purposes, but it will become clear that it is problematic for some interesting protocol classes, the TLS protocol being one of them.

from asking a Test query) and perform checks, if it matches the protocol execution or not. To still be able to make security statements about such protocols we introduce a new type of protocols in this section, namely *authenticated and confidential channel establishment* (ACCE) protocols.

An ACCE protocol is a protocol executed between two parties. The protocol consists of two stages, called the 'pre-accept' stage and the 'post-accept' stage.

Pre-accept stage. In this stage a 'handshake protocol' is executed. In terms of functionality this protocol is an AKE protocol as defined in Section 3.2.1, that is, at least one communication partner is authenticated, and a session key k is established. However, it need not necessarily meet the security definitions for AKE protocols. This stage ends, when both communication partners reach an `accept`-state (i.e. $\Lambda =$ '`accept`').

Post-accept stage. This stage is entered, when both communication partners reach an `accept`-state. In this stage data can be transmitted, encrypted and authenticated with key k.

The prime example of an ACCE protocol is TLS. Here, the pre-accept stage consists of the TLS Handshake protocol (including the encrypted `Finished` messages, see also Section 5.1). In the post-accept stage encrypted and authenticated data is transmitted over the TLS record layer.

To define security of ACCE protocols, we combine the security model for authenticated key exchange of Bellare and Rogaway [BR94a] transferred to the public key setting [BWJM97] with stateful length-hiding encryption in the sense of [PRS11].

Extended Execution Environment

The execution environment for ACCE protocols is identical to the extended execution environment for AKE protocols described in Section 3.2.1.

Extended Queries Compared to simulating AKE protocols, ACCE protocols are not 'finished' once the session keys are exchanged, so the security definition of ACCE protocols additionally captures the security of messages exchanged in the post-accept stage. In order to let the adversary interact with oracles in the post-accept stage, we provide the adversary with the ability to send messages over the encrypted channel (without knowing the symmetric key used for encryption) and to learn the plaintexts of messages exchanged between oracles. We also restrict the Send query to sending plaintext messages, i.e. this query is to be used only during the pre-accept stage.

Similar to the Send query, the $\mathsf{Send}^{\mathsf{pre}}$ query enables the adversary to initiate and run an arbitrary number of protocol instances, sequential or in parallel, and provides full control over the communication between all parties during the pre-accept stage. Due to this modification, the original Send query may no longer be asked. In order to allow an adversary to encrypt plaintext messages and decrypt obtained ciphertexts, we introduce two new queries. During the post-accept stage, the Encrypt query enables the adversary to transmit plaintexts and the Decrypt query enables the adversary to learn the plaintexts of messages sent.

Formally, the queries available to the adversary in the ACCE setting are identical to the queries detailed in Section 3.1.1, with the following additional queries:

- $\mathsf{Send}^{\mathsf{pre}}(\pi_i^s, m)$: This query is identical to the Send query from Section 3.2.1, except that it replies with an error symbol $\perp$ if oracle π_i^s has state $\Lambda =$ '`accept`'. (Send queries in an accept-state are handled by the Decrypt query below).

- $\mathsf{Encrypt}(\pi_i^s, m_0, m_1, \mathsf{len}, H)$: This query takes as input two messages m_0 and m_1, length parameter len, and header data H. If $\Lambda \neq$ '`accept`' then π_i^s returns $\perp$.

  Otherwise, it proceeds as depicted in Figure 3.1, depending on the random bit $b_i^s \xleftarrow{\$} \{0,1\}$ sampled by π_i^s at the beginning of the game and the internal state variables of π_i^s.

  To allow the parties to encrypt the `Finished` messages in our simulation, we allow the party itself to call this oracle even before $\Lambda =$ '`accept`'; this abuse of notation allows the party to construct encrypted protocol messages while all aspects of the security experiment remain synchronized.

- $\mathsf{Decrypt}(\pi_i^s, C, H)$: This query takes as input a ciphertext C and header data H. If π_i^s has $\Lambda \neq$ '`accept`' then π_i^s returns $\perp$. Otherwise, it proceeds as depicted in Figure 3.1.

  Similiar to above, we allow the parties to decrypted received `Finished` messages in our simulation by allowing the party itself to call this oracle even before $\Lambda =$ '`accept`'.

Remark 9. As we use this model to analyze TLS and since the TLS record layer is unidirectional (i.e. there are both encryption and decryption keys, and for most ciphersuites also MAC keys) all negotiated keys ($K_{\mathsf{enc}}^{C\to S}, K_{\mathsf{enc}}^{S\to C}, K_{\mathsf{mac}}^{C\to S}, K_{\mathsf{mac}}^{S\to C}$) are revealed by the Reveal query, though one could imagine refinements if desired.

[14]Although $\pi_i^s.\Pi$ only contains the party identifier j, in the post-accept stage every oracle π_i^s has a unique partner oracle π_j^t known to the challenger.

```
$\mathsf{Encrypt}(\pi_i^s, m_0, m_1, \mathsf{len}, H)$:
1. $(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k^{\rho}_{\mathsf{enc}}, \mathsf{len}, H, m_0, st_e)$
2. $(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k^{\rho}_{\mathsf{enc}}, \mathsf{len}, H, m_1, st_e)$
3. If $C^{(0)} = \perp$ or $C^{(1)} = \perp$ then return $\perp$
4. $u_i^s := u_i^s + 1$
5. $(C_i^s[u_i^s], H_i^s[u_i^s], st_e) := (C^{(b_i^s)}, H, st_e^{(b_i^s)})$
6. Return $C_i^s[u_i^s]$
```

```
$\mathsf{Decrypt}(\pi_i^s, C, H)$:
1. $(j,t) := \pi_i^s.\Pi^{14}$, $m' \leftarrow \perp, r' \leftarrow bot$
2. $v_i^s := v_i^s + 1$
3. $(m, st_d) = \mathsf{StE.Dec}(k^{\rho}_{\mathsf{dec}}, H, C, st_d)$
4. if $m \neq \perp$, $r' \leftarrow$ protocol response for m
5. If $v_i^s > u_j^t$ or $C \neq C_j^t[v_i^s]$
     then $\mathsf{phase} := 1$
6. If $\mathsf{phase} = 1$ and $b_i^s = 1$ then $m' \leftarrow m$,
7. Return $(m', r')$
```

Here $u_i^s, v_i^s, b_i^s, \rho, k^{\rho}_{\mathsf{enc}}, k^{\rho}_{\mathsf{dec}}$ denote the values stored in the corresponding internal variables of π_i^s.

Figure 3.1: Encrypt and Decrypt oracles in the ACCE security experiment.

Remark 10. Note that in the case of TLS, a message encrypted by some oracle π_i^s can only be decrypted by its 'partner' oracle, as different keys are used for the different communication directions (i.e. a single oracle uses different keys for encryption and decryption).

Correctness and Security Definition

Security of ACCE protocols is defined by requiring that (i) the protocol is a secure authentication protocol and (ii) in the post-accept stage all data is transmitted over an authenticated and confidential channel in the sense of sLHAE (Definition 2.11).

We now give our definitions of correct and secure ACCE protocols with mutual authentication, adapted to the modified execution environment. We also model perfect forward secrecy.

ACCE Game. Again this notion is captured by a game, played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The challenger implements the collection of oracles $\{\pi_i^s : i \in [\phi], s \in [\eta]\}$. At the beginning of the game, the challenger generates ϕ long-term key pairs (pk_i, sk_i) for all $i \in [\phi]$. The adversary receives the public keys $pk_1, \ldots, pk_\phi$ as input. Now the adversary may start issuing $\mathsf{Send}^{\mathsf{pre}}$, Reveal, $\mathsf{Corrupt}$, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$ queries. Finally, the adversary outputs a triple (i, s, b') and terminates.

Definition 3.7. Assume a 'benign' adversary $\mathcal{A}$, which picks two arbitrary oracles π_i^s and π_j^t and performs a sequence of $\mathsf{Send}^{\mathsf{pre}}$ queries by faithfully forwarding all messages of the pre-accept stage between π_i^s and π_j^t. Let $k_i^s = (k_{\mathsf{enc}}^{\mathsf{Client}}, k_{\mathsf{dec}}^{\mathsf{Client}})$ denote the key computed by π_i^s and let $k_j^t = (k_{\mathsf{dec}}^{\mathsf{Server}}, k_{\mathsf{enc}}^{\mathsf{Server}})$ denote the key computed by π_j^t.

We say that an ACCE protocol is *correct*, if for this benign adversary and any two oracles π_i^s and π_j^t always holds that

1. both oracles have $\Lambda = \texttt{accept}$,
2. $k_i^s = k_j^t \in \mathcal{K}$.

Furthermore we require that the stateful encryption scheme is correct, i.e. for all messages $m \in \{0,1\}^*$, lengths fields $\mathsf{len} \in \mathbb{N}$ that are valid for m, roles $\rho \in \{\mathsf{Client}, \mathsf{Server}\}$, keys $k = (k_{\mathsf{enc}}^\rho, k_{\mathsf{dec}}^\rho)$, headers $H \in \{0,1\}^*$, and (corresponding) encryption/decryption states $st_e, st_d \in \{0,1\}^*$ holds that

$$\mathsf{StE.Dec}(k_{\mathsf{dec}}^\rho, H, \mathsf{StE.Enc}(k_{\mathsf{enc}}^\rho, \mathsf{len}, H, m, st_e), st_d) = m.$$

Mutual Authentication. In the following we give the definitions for mutually authenticated and confidential channel establishment protocols with and without perfect forward secrecy.

Similar to the AKE security definitions we will require in Property 1 that for every oracle that accepts with a partner identifier pointing to an uncorrupted party there exists a partner oracle with matching conversations. In Property 2 we require that the adversary cannot break the security of the underlying encryption scheme, i.e. create new (valid) ciphertexts or distinguish between the encryptions of different plaintexts.

Definition 3.8. We say that an adversary *(t, ϵ)-breaks* an ACCE protocol with mutual authentication, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:

1. When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s with internal state ρ such that
   - π_i^s 'accepts' with partner $\Pi = j$ when $\mathcal{A}$ issues its τ_0-th query, and
   - P_i and P_j are τ_i, τ_j-corrupted with $\tau_i, \tau_j > \tau_0$, and
   - $\mathcal{A}$ did not issue a Reveal query to oracle π_j^t, such that π_j^t accepted while having a matching conversation to π_i^s (if such an oracle exists) and
   - there is no unique oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

   If an oracle π_i^s accepts in the above sense, then we say that π_i^s accepts *maliciously.*

2. When $\mathcal{A}$ terminates and outputs a triple (i, s, b') such that
   - π_i^s 'accepts' with intended partner $\Pi = j$ when $\mathcal{A}$ issues its τ_0-th query,
   - P_i and P_j are τ-corrupted with $\tau = \infty$, and
   - $\mathcal{A}$ did not issue a Reveal query to π_i^s, nor to π_j^t such that π_i^s has a matching conversation to π_j^t (if such an oracle exists),

then the probability that b' equals b_i^s is bounded by

$$|\Pr[b_i^s = b'] - 1/2| \geq \epsilon.$$

If an adversary $\mathcal{A}$ outputs (i, s, b') such that $b' = b_i^s$ and the above conditions are met, then we say that $\mathcal{A}$ *anwers the encryption-challenge correctly.*

Remark 11. In comparison to the AKE definitions (Section 3.2.1), we included an additional condition in the ACCE security definitions. Namely, we require that $\mathcal{A}$ did not issue a Reveal query to an oracle π_j^t, such that π_j^t accepted while having a matching conversation to π_i^s, to prevent a trivial attack, which is possible for all ACCE protocols in which the last message (of the pre-accept stage) m_x sent was (at least partially) computed by evaluating some probabilistic function $m_x \xleftarrow{\$} \mathsf{F}(k, in, r)$ under the established session key k over some publicly known input in, like for instance the transcript of all messages, and randomness r. For simplicity we assume, that some server oracle π_j^t sends the last message m_x to a client oracle π_i^s (its intended partner). The adversary can now proceed as follows:

By definition, π_j^t has to accept after sending m_x (without knowing if this message was faithfully received by π_i^s). Then (without this condition) it would be possible to ask a Reveal query to π_j^t and learn the session key k. At this point $\mathcal{A}$ may drop the message m_x, use the key k to compute a similar message m'_x by evaluating the same function with fresh randomness r' as $m'_x \xleftarrow{\$} \mathsf{F}(k, in, r')$ over the same input in that π_j^t used, and send $m'_x \neq m_x$ to π_i^s. Then π_i^s will accept without having a matching conversation π_j^t.

This issue was pointed out by our collegue Yong Li and observed independently in [BSWW13].

Definition 3.9. We say that an adversary (t, ϵ)-*breaks* an ACCE protocol with mutual authentication and perfect forward secrecy, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:

1. When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s that accepts *maliciously* in the sense of Property 1 of Definition 3.8.
2. When $\mathcal{A}$ terminates and outputs a triple (i, s, b') such that
   - π_i^s 'accepts' with intended partner $\Pi = j$ when $\mathcal{A}$ issues its τ_0-th query,
   - P_j is τ_j-corrupted with $\tau_0 < \tau_j$, and
   - $\mathcal{A}$ did not issue a Reveal query to π_i^s, nor to π_j^t such that π_i^s has a matching conversation to π_j^t (if such an oracle exists),

   then the probability that b' equals b_i^s is bounded by

$$|\Pr[b_i^s = b'] - 1/2| \geq \epsilon.$$

Remark 12. Note that we do not model Key-Compromise Impersonation Attacks (KCI) for ACCE protocols with mutual authentication. The reason is rather practical. One of the TLS ciphersuites we analyze, namely TLS with mutual authentication and static Diffie-Hellman key exchange, does not provide security against Key-Compromise Impersonation Attacks. In order to keep the model and definitions plain and simple, and to avoid giving seperate definitions for the other ciphersuites that allow for modeling KCI attacks, we leave this out-of-scope for this work. However, all definitions can be simply adjusted by removing the restriction on client corruption in Property 1 (e.g. in above Definition 3.8).

Server-Only Authentication. In the following we modify the above definitions to define server-only authenticated and confidential channel establishment protocols with and without perfect forward secrecy. Note, that similar to AKE protocols, the only difference to the above definitions with mutual authentication is that we again restrict the *maliciously* accepting oracle to be a Client, i.e. to have internal state $\rho = \mathsf{Client}$.

Definition 3.10. We say that an adversary (t, ϵ)-*breaks* an ACCE protocol with server-only authentication, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:

1. When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s such that π_i^s *maliciously* accepts in the sense of Property 1 of Definition 3.8, and π_i^s has internal state $\rho = \mathsf{Client}$.

2. $\mathcal{A}$ answers the encryption-challenge correctly in the sense of Property 2 of Definition 3.8 and either
   - oracle π_i^s has internal state $\rho = \mathsf{Client}$, or
   - oracle π_i^s has internal state $\rho = \mathsf{Server}$, and there exists an oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

Definition 3.11. We say that an adversary (t, ϵ)-*breaks* an ACCE protocol with server-only authentication and perfect forward secrecy, if $\mathcal{A}$ runs in time t, and at least one of the following two conditions holds:

1. When $\mathcal{A}$ terminates, then with probability at least ϵ there exists an oracle π_i^s that accepts *maliciously* in the sense of Property 1 of Definition 3.9, and π_i^s has internal state $\rho = \mathsf{Client}$

2. $\mathcal{A}$ answers the encryption-challenge correctly in the sense of Property 2 of Definition 3.9 and either
   - oracle π_i^s has internal state $\rho = \mathsf{Client}$, or

- oracle π_i^s has internal state $\rho = \mathsf{Server}$, and there exists an oracle π_j^t such that π_i^s has a matching conversation to π_j^t.

We say that an ACCE protocol is (t, ϵ)-*secure* (with respect to any of the above definitions), if it is correct and there exists no adversary that (t, ϵ)-*breaks* it (with respect to the same definition).

Remark 13. Also note that the above definitions for protocols that provide *perfect forward secrecy* even allow to corrupt the oracle π_i^s whose internal secret bit the adversary tries to determine (Property 2). We again allow the 'accepting' oracle to be corrupted even *before* it reaches an `accept`-state, which provides security against KCI attacks (Property 1).

Also note that by explicitly differentiating between authentication and confidentiality, we are able to model KCI attacks against protocols without perfect forward secrecy.

Relation to the AKE Security Definitions from Section 3.2.1

Note that an ACCE protocol can be constructed in a two-step approach.

1. (AKE part) First an AKE protocol is executed. This protocol guarantees the authenticity of the communication partner, and provides a cryptographically 'good' (i.e. for the adversary indistinguishable from random) session key.
2. (Symmetric part) The session key is then used in a symmetric encryption scheme providing integrity and confidentiality.

This modular approach is simple and generic, and therefore appealing. It can be shown formally that this two-step approach yields a secure ACCE protocol, if the 'AKE part' meets the security in the sense of Definitions 3.3 to 3.6, and the 'symmetric part' consists of a suitable authenticated symmetric encryption scheme.

Remark 14. Note that our ACCE security experiment is specifically tailored to model the properties of TLS and makes use of a Stateful Length-Hiding Authenticated Encryption (sLHAE) according to Definition 2.11. However, in general other authenticated symmetric encryption schemes may be suitable as well.

However, if the purpose of the protocol is the establishment of an authenticated and confidential channel, then it is not necessary that the 'AKE-part' of the protocol provides full indistinguishability of *session keys*. It actually would suffice if *encrypted messages* are indistinguishable, and *cannot be altered* by an adversary. These requirements are strictly weaker than indistinguishability of keys in the sense of Property 2 of Definitions 3.3 to 3.6, and thus easier to achieve (possibly from weaker hardness assumptions, or by more efficient protocols).

We stress that our ACCE definition is mainly motivated by the fact that security models based on key indistinguishability do not allow for a security analysis of full TLS, as detailed in the introduction. We do not want to propose ACCE as a new security notion for key exchange protocols, since it is very complex and the modular two-step approach approach seems more useful in general.

Examplary Protocol Execution Let us now review how an adversary would use the queries to carry out a normal TLS negotiation and renegotiation.

1. First the adversary sends a $\mathsf{Send}^{\mathsf{pre}}$ query to the client oracle containing the symbol $\top$. The client sets its internal variable $\rho := \mathsf{Client}$ and responds with the first protocol message, `ClientHello`. Both oracles draw a random bit and store it in b, all messages (except for `newphase` and `ready`) are recorded in the transcript T of each party.

2. The adversary delivers the first message from the client to the server by making a $\mathsf{Send}^{\mathsf{pre}}$ query to the server, which returns the next message from the server to the client (`ServerHello`, `ServerKeyExchange`, etc.).

3. The adversary delivers the messages to the client via a $\mathsf{Send}^{\mathsf{pre}}$ query. The client responds with several plaintext messages (e.g. `ClientKeyExchange`) as well as a `ChangeCipherSpec` message and stores all keys (such as the premaster secret, the master secret and the application keys) in the state variable st. It also sets its partner id Π according to the received certificate. There is one more message, the `ClientFinished` message, which the client first encrypts using an internal $\mathsf{Encrypt}$ call.

4. The plaintext messages are delivered by the adversary to the server using $\mathsf{Send}^{\mathsf{pre}}$ and the encrypted message is delivered using $\mathsf{Decrypt}$. The response by the server from $\mathsf{Send}^{\mathsf{pre}}$ will be a `ChangeCipherSpec` message and the response by the server from $\mathsf{Decrypt}$ will be the `ServerFinished` message, which the server first encrypts by internal calling $\mathsf{Encrypt}$.

5. The encrypted message is delivered by the adversary using $\mathsf{Decrypt}$. The oracles set $\Lambda = \mathtt{accept}$, copy the application keys into the variable k and are now ready to use the record layer, which the adversary can now also make use of by matching $\mathsf{Encrypt}$/$\mathsf{Decrypt}$ queries (which are answered depending on the internal bit b stored at the oracles).

3.2.3 Multi-Phase and Renegotiable ACCE Protocols

In this section we describe a model and give three different security notions for *multi-phase* and *renegotiable* ACCE protocols, differing in the strength of the 'cryptographic link' between subsequent key exchanges. Essentially, a multi-phase protocol between two parties can have many key exchanges - each called a *phase* - linked to a single session. This definition builds on the ACCE definition, described in detail in Section 3.2.2.

Since our goal is to analyze the security of TLS, we start from the ACCE model described in Section 3.2.2, rather than from the AKE security model of Section 3.2.1. The primary difference in our model for renegotiable protocols is that each party's oracle (session) can have multiple *phases*; each new phase corresponds to a renegotiation in that session, and can involve the same or different long-term keys.[15] This is qualitatively different from simply having multiple sessions, since short-term values from one phase of a session may be used in the renegotiation for the next phase, whereas multiple sessions only share long-term values. Each oracle maintains state and encryption/MAC keys for each phase. Like in TLS, our formalism allows control messages to be sent on the encrypted channel.

The basic goals of a *secure renegotiable ACCE protocol* are that (a) an adversary should not be able to read or inject messages on the encrypted channel, and (b) whenever parties successfully renegotiate, the set of all unencrypted and encrypted messages sent in all previous phases of that session should match, even when values from previous phases have been compromised. We then say that both parties *have the same view* of all previous negotiations.

Informal Descriptions of the different Security Definitions

We start by giving (informal) descriptions of the security goals of multi-phase and renegotiation protocols, which we then elaborate by describing an execution environment and the adversarial interaction capabilities and giving formal security definitions.

Secure Multi-Phase ACCE Protocols. This first security notion is a straightforward extension of the ACCE model to allow multiple, independent phases per session; notably, we require essentially no link between phases:

- An adversary breaks *(multi-phase) authentication* if a party accepts in a phase where long-term keys have not been corrupted, but no matching phase exists at the peer.
- An adversary breaks *confidentiality/integrity* if it can guess the bit b involved in a stateful length-hiding authenticated encryption-type confidentiality/integrity experiment.

[15]Note that TLS standards use different words. We say a single *session* can have multiple *phases*; the TLS standards refer to a single *connection* having multiple *sessions*.

We will later show, that the plain TLS renegotiation protocol (i.e. without any countermeasures) meets this security notion. However, TLS renegotiation has been shown to be vulnerable against certain attacks (cf. 6.1) that are not covered by this notion, so we also introduce a new, stronger notion of security.

Secure Renegotiable ACCE Protocols. This is our central security definition, which strengthens the authentication notion: parties should successfully renegotiate only when they have the (exact) same view of everything that happened before.

- An adversary breaks *renegotiation authentication* if a party accepts in a phase where long-term keys have not been corrupted, but either no matching phase exists at the peer *or some previous handshake or record layer transcript does not match.*

This strong definition requires that the views of parties match when successfully renegotiating, even when previous sessions' long-term secret keys or session keys were revealed. We then analyzed TLS renegotiation (more specifically, a version of TLS already including countermeasures against recent attacks on TLS renegotiation protocol) and found, that TLS's Signalling Ciphersuite Value (SCSV)/Renegotiation Information Extension (RIE) countermeasures do not fully protect against the case when these secret values are revealed (see Section 6.4 for details). As a result, we introduce a slightly weaker, though still quite reasonable notion of security.

Weakly-secure Renegotiable ACCE Protocols. As mentioned before, this notion is slighty weaker then secure renegotiable ACCE, but it enables us to prove (in Section 6.7) that the SCSV/RIE countermeasure for TLS generically provides it:

- An adversary breaks *weak renegotiation authentication* if a party accepts in a phase with uncorrupted long-term keys *and session keys for each earlier phase were not revealed while that phase was active*, but either no matching phase exists at the peer or some previous handshake or record layer transcript does not match.

We proceed by describing the execution environment for adversaries interacting with multi-phase ACCE protocols, then formally define the various security notions described above.

Extended Execution Environment

In the following we denote with $\texttt{phases}[\ell]$ the ℓth entry in the array $\texttt{phases}$ and $|\texttt{phases}|$ denotes the number of entries in the array. In order to formalize multi-phase protocols, we now speak of parties, sessions **and phases** (instead of only parties and sessions) and accordingly we modify the execution environment presented in Section 3.1.1 as follows:

Parties. For each party P_i the long-term secret L_i now consists of a list of γ long-term key pairs $(pk_{i,1}, sk_{i,1}), \ldots, (pk_{i,\gamma}, sk_{i,\gamma})$, instead of containing only a single keypair. We assume that each party P_i is uniquely identified by any one of its public keys $pk_{i,*}$.

Sessions. As described in Section 3.1.1, each party P_i can participate in up to η sessions, which are independent executions of the protocol and can be concurrent or subsequent; all of a party's sessions have access to the same list of its long-term key pairs, as well as a trusted list of all parties' public keys. Again, we refer to party P_i in a specific session s as oracle π_i^s.

Each oracle π_i^s records in a variable $\pi_i^s.\Pi$ the oracle corresponding to the intended communication partner, e.g. $\pi_i^s.\Pi = \pi_j^t$.[16] As well, the variable $\rho \in \{\mathsf{Client}, \mathsf{Server}\}$ records the role of the oracle. Parties can play the role of the client in some sessions and of the server in other sessions, but **their role is fixed across all phases within a session.**

Phases. Each session can now consist of up to ψ phases. Analog to ACCE protocols, each phase consists of two stages: a *pre-accept*, or 'handshake', stage, which is effectively an AKE protocol that establishes a session key and performs mutual or server-only authentication; and a *post-accept*, or 'record layer', stage, which provides a stateful communication channel with confidentiality and integrity. A list $\pi_i^s.\mathtt{phases}$ of different phase states is maintained; we sometimes use the notation $\pi_i^{s,\ell}$ for $\pi_i^s.\mathtt{phases}[\ell]$. There can be at most ψ phases per oracle. The last entry of $\pi_i^s.\mathtt{phases}$ contains the state of the current phase, which may still be in progress. Each entry $\pi_i^s.\mathtt{phases}[\ell]$ in the log contains:

- pk, the public key used by π_i^s in that phase,
- pk', the public key that π_i^s observed as being used for its peer in that phase[17],
- $\alpha \in \{0, 1\}$, denoting the authentication mode used, where $\alpha = 0$ indicates that server-only authentication is used in that phase and $\alpha = 1$ indicates mutual authentication,
- Λ, either `accept`, `reject`, or $\emptyset$ (for in-progress),

[16] As the simulator transmits all messages, it can easily determine the matching oracle. If no such oracle exists, $\pi_i^s.\Pi$ remains $\emptyset$.

[17] One of the public keys may remain empty, if no client authentication is requested.

- k, the encryption and/or MAC key(s) established by π_i^s in that phase,
- $\mathsf{T}^{i,s}$, the transcript of all (plaintext) messages sent and received by π_i^s during the pre-accept stage of that phase,
- RT_s and RT_r, the (record layer) transcripts of all ciphertexts sent and received (respectively) in the post-accept phase by π_i^s encrypted under the key established in that phase,
- b, a random bit sampled by the oracle at the beginning of the phase, and
- st, some additional temporary state (which may, for instance, be used to store ephemeral Diffie-Hellman exponents for the handshake, or state for the sLHAE scheme for the record layer).

The random bit is chosen as $b \overset{\$}{\leftarrow} \{0,1\}$ and the remaining internal state is initialized to $(\mathit{\Pi}, \rho, pk, pk', \alpha, \Lambda, k, \mathsf{T}, \mathsf{RT}_s, \mathsf{RT}_r, st) \leftarrow (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$. Again we say that $V = \emptyset$ denotes that variable V is undefined. When describing a protocol, we will enumerate the protocol messages. Once a phase of a protocol accepts (that is, an encryption key has been negotiated and authentication is believed to hold), then Λ is set to `accept`. Whenever a new handshake initialization message is received, the oracle adds a new entry to its `phases` list. The variables $\mathit{\Pi}$ and α are set at some point during (or before) the protocol execution, depending on the protocol specification. In case of TLS, the server can send the message `CertificateRequest` to request client (thus mutual) authentication (and thus specify the authentication mode α). Otherwise server-only authentication is used. Upon receipt of a certificate, the communication partner $\mathit{\Pi}$ can be determined. Application data messages sent and received encrypted under a newly established encryption key (e.g. messages sent in the TLS record layer) will be appended to variables RT_s and RT_r in the latest entry of the log. If handshake messages for the renegotiation of a new phase are encrypted under the previous phase's session key (as they are in TLS), the plaintext messages are appended to variable T in the new entry of the phase log, and ciphertexts are appended to RT in the previous phase.

Remark 15. The introduction of multiple `phases` is the main difference compared to previous AKE and ACCE models. We need to allow multiple authentications and key exchanges within one oracle to capture the functionality of renegotiation. When limited to a single phase and when each party has only one long-term key pair, our execution environment/security experiment is roughly equivalent to the ACCE model described in the previous section.

Extended Adversarial Capabilites

In the following we extend the queries defined in Section 3.1.1 to be used in conjunction with multi-phase protocols. As noted above, the security model for multi-phase protocols is closely related to the ACCE security model, so the following queries are closely related to the queries defined in Section 3.2.2. Compared to the previous models, the adversary is now additionally allowed to trigger new phases. Instead of adding a further query to model this new capability, we enhanced the $\mathsf{Send}^{\mathsf{pre}}$ query described in the ACCE section to allow for triggering new phases. Recall that in renegotiation protocols the handshake messages of all phases except for the first phase are already sent over the previously established encrypted channel. Thus, we now differentiate between (encrypted) handshake messages and ‘regular’ data by assigning a *content type* to each message and modify the Encrypt and Decrypt queries accordingly.

- $\mathsf{Send}^{\mathsf{pre}}(\pi_i^s, m)$: The adversary can use this query to send any (plaintext) message m of its choosing to (the current phase of) oracle π_i^s. The oracle will respond according to the protocol specification, depending on its internal state. Some distinguished control messages have special behaviour:
  - $m = (\texttt{newphase}, pk, \alpha)$ triggers this oracle to initiate renegotiation of a new phase (or new session if first phase). Note that the action here may vary based on the role of the party: for example, when renegotiating in TLS, a client would prepare a new `ClientHello` message, encrypt it by calling the Encrypt oracle below, and then return the ciphertext to the adversary for delivery; a server would correspondingly prepare an encrypted `HelloRequest` message.
  - $m = (\texttt{ready}, pk, \alpha)$ activates this oracle to use the public key pk in its next phase.

  For the above control messages, pk indicates the long-term public key pk the oracle should use in the phase and α indicates the authentication mode to use; the oracle returns $\perp$ if it does not hold the secret key for pk. Since the control messages do not specify the identity of the peer, this is again learned during the run of the protocol.

  Delivery of encrypted messages in the post-accept stage are handled by the Decrypt query below. For protocols such as TLS that perform renegotiation within the encrypted channel, the oracle may reply with an error symbol $\perp$ if it has at least one entry in `phases` and $m \neq (\texttt{newphase}, \cdot)$ or $(\texttt{ready}, \cdot)$.

- $\mathsf{Corrupt}(P_i[, pk])$: Oracle π_i^1 responds with the long-term secret key $sk_{i,n}$ corresponding to public key $pk = pk_{i,n}$ of party P_i, or $\perp$ if there is no n such that $pk = pk_{i,n}$. If no specific pk is queried, all long-term secret keys stored in variable L_i of party P_i are returned.

- $\mathsf{Reveal}(\pi_i^s, \ell)$: Oracle π_i^s responds with the key(s) $\pi_i^s.\mathtt{phases}[\ell].k$ used in phase ℓ, or $\emptyset$ if no such value exists.
- $\mathsf{Encrypt}(\pi_i^s, ctype, m_0, m_1, \mathsf{len}, H)$: This query depends on the random bit b sampled by π_i^s at the beginning of the current phase. It takes as input a content type $ctype$, messages m_0 and m_1, a length len, and header data H.

  Content type `control` is used for handshake messages. The adversary cannot query this oracle with $ctype = \mathtt{control}$. To allow the parties to encrypt protocol messages that must be sent encrypted (e.g. handshake messages sent in later phases), we allow the party itself to call this oracle with `control`; this abuse of notation allows the party to construct encrypted protocol messages while all aspects of the security experiment remain synchronized.

  Content type `data` is used for record layer messages; in this case, one of the two messages (chosen based on bit b) is encrypted for the adversary to distinguish. Encrypt maintains a counter u initialized to 0 and an encryption state st_e, and proceeds as depicted in Figure 3.2.
- $\mathsf{Decrypt}(\pi_i^s, C, H)$: This query takes as input a ciphertext C and header data H. If π_i^s has not accepted in the current phase, then it returns $\perp$. Decrypt maintains a counter v and a switch diverge, both initialized to 0, and a decryption state st_d, and proceeds as depicted in Figure 3.2.

  If the decryption of C contains a `control` message, then the oracle processes the message according to the protocol specification, which may include updating the state of the oracle and/or creating a new phase, and returns any protocol response message to the adversary, which may or may not be encrypted by calling Encrypt according to the protocol specification.

The behaviour of the Decrypt oracle in this combined definition for confidentiality and integrity can be somewhat difficult to understand. It extends that of stateful length-hiding authenticated encryption as defined in Section 3.2.2 by differentiating between control type (i.e. handshake) messages and arbitrary data. The diverge flag is set, when the adversary successfully injects a valid ciphertext that was not output by the Encrypt oracle before. Once both diverge flag and bit b are set to $\mathsf{diverge} = b = 1$, the Decrypt oracle subsequently outputs the decryption of queried ciphertexts.

Examplary Protocol Execution Let us now review how an adversary would use the queries to carry out a normal TLS negotiation and renegotiation (for more infor-

```
$\mathsf{Encrypt}(\pi_i^s, ctype, m_0, m_1, \mathsf{len}, H)$:
1. $u \leftarrow u + 1$
2. If $(ctype = \texttt{control})$ AND caller is not $\pi_i^s$,
     then return $\perp$
3. $(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, ctype\|m_0, st_e)$
4. $(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, \mathsf{len}, H, ctype\|m_1, st_e)$
5. If $(C^{(0)} = \perp)$ OR $(C^{(1)} = \perp)$, then return $\perp$
6. $(C_u, st_e) \leftarrow (C^{(b)}, st_e^{(b)})$
7. Return $C_u$
```

```
$\mathsf{Decrypt}(\pi_i^s, C, H)$:
1. $v \leftarrow v + 1$, $m' \leftarrow \perp$, $r' \leftarrow \perp$
2. $(ctype\|m, st_d) = \mathsf{StE.Dec}(k, H, C, st_d)$
3. If $(v > u)$ OR $(C \neq C_v)$, then $\mathsf{diverge} \leftarrow 1$
4. If $(b = 1)$ AND $(\mathsf{diverge} = 1)$, then $m' \leftarrow m$
5. If $ctype = \texttt{control}$,
     then $r' \leftarrow$ protocol response for $m$
6. Return $(m', r')$
```

where $(k, b, \mathsf{diverge}) = \pi_i^s.\texttt{phases}[\ell^*].(k, b, \mathsf{diverge})$ and ℓ^* is the last phase π_i^s accepted.

Note: k may be a 'multi-part' key with different parts for encryption, decryption, and MAC; we assume StE.Enc and StE.Dec know which parts to use.
The 'protocol response for m' may be encrypted by the party internally making an Encrypt call.

Figure 3.2: Encrypt and Decrypt oracles for the multi-phase/renegotiable ACCE security experiments.

mation about the TLS handshake, see Section 5.1).

1. First the adversary uses the $\mathsf{Send}^{\mathsf{pre}}$ query to deliver `newphase` and `ready` messages to the client and server. The client responds to the $\mathsf{Send}^{\mathsf{pre}}$ query with a `ClientHello` message and sets $\rho = \mathsf{Client}$ and pk to the public key provided by the query; the server responds with $\perp$ and sets $\rho := \mathsf{Server}$ and pk accordingly. Both oracles draw a random bit and store it in b, all messages (except for `newphase` and `ready`) are recorded in the transcript T of each party.

2. The adversary delivers the first message from the client to the server by making a $\mathsf{Send}^{\mathsf{pre}}$ query to the server, which returns the next message from the server to the client (`ServerHello`, `ServerKeyExchange`, etc.). Depending on whether the server requests client authentication, the variable α is set either to 0 or 1. The variable pk' is filled by the client (and server, if $\alpha = 1$) with the public key extracted from the received certificate.

3. The adversary delivers the messages to the client via a $\mathsf{Send}^{\mathsf{pre}}$ query. The client responds with several plaintext messages (e.g. `ClientKeyExchange`) as well as a `ChangeCipherSpec` message and stores all keys (such as the premaster secret, the master secret and the application keys) in the state variable st. There is one more message, the `ClientFinished` message, which the client first encrypts using an internal Encrypt call.

4. The plaintext messages are delivered by the adversary to the server using $\mathsf{Send}^{\mathsf{pre}}$ and the encrypted message is delivered using Decrypt. The response by the server from $\mathsf{Send}^{\mathsf{pre}}$ will be a `ChangeCipherSpec` message and the

response by the server from Decrypt will be the `ServerFinished` message, which the server first encrypts by internal calling Encrypt.

5. The encrypted message is delivered by the adversary using Decrypt. The oracles set $\Lambda = \texttt{accept}$, copy the application keys into the variable k and are now ready to use the record layer, which the adversary can now also make use of by matching Encrypt/Decrypt queries (which are answered depending on the internal bit b stored at the oracles). All messages that are now sent are recorded in the record layer transcripts RT_s of each oracle and all messages that are received in RT_r accordingly.

   When the adversary wants to trigger client-initiated renegotiation, it sends a `newphase` message via a $\mathsf{Send}^{\mathsf{pre}}$ query to the client, who responds with a `ClientHello` message encrypted via an internal Encrypt call.

6. The adversary delivers this to the server by a Decrypt call; the server responds with an encrypted protocol message, and so on. Note that the plaintext handshake messages are appended to the new phase's transcript T *and* the ciphertext handshake messages are also appended to the current existing phase's transcripts RT_s and RT_r.

7. When the oracles accept in the phase, they begin using the encryption keys for the new phase.

Correctness and Security Definitions

In the original security definition for ACCE protocols, security is defined by requiring that (i) the protocol is a secure authentication protocol, thus any party π_i^s reaches the post-accept state only if there exists another party π_j^t such that π_i^s has a matching conversation (in the sense of Definition 3.1) to π_j^t, and (ii) data transmitted in the post-accept stage over a secure channel is secure (in a sense similar to sLHAE).

We extend this notion to include security when a session has multiple phases that can be renegotiated. We will give several security definitions with different levels of security against renegotiation attacks, as described in Section 6.1.

Each security notion is formally described as a game played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, with the same overall setup but different winning conditions. In each game, the challenger implements the collection of oracles $\{\pi_i^s : i \in [\phi], s \in [\eta]\}$. At the beginning of the game, the challenger generates γ long-term key pairs $(pk_{i,1}, sk_{i,1}), \ldots, (pk_{i,\gamma}, sk_{i,\gamma})$ for each party P_i; we assume that, within a party, all public key pairs are distinct. (That distinct parties have distinct key pairs is necessary for the protocol to be secure.) The adversary receives all parties' public keys as input. The adversary may issue $\mathsf{Send}^{\mathsf{pre}}$, $\mathsf{Corrupt}$, Reveal, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$ queries to the oracles and eventually terminates.

Table 3.1 at the end of the section provides a comparative summary of the various security notions introduced in this section, as well as a summary of the results on TLS that appear in the rest of this paper.

Definition 3.12 (Correct multi-phase ACCE)**.** Assume a 'benign' adversary $\mathcal{A}$, which picks two arbitrary oracles π_i^s and π_j^t and performs a sequence of $\mathsf{Send}^{\mathsf{pre}}$ queries by faithfully forwarding all messages between π_i^s and π_j^t. We say Π is a *correct multi-phase ACCE protocol* if, for this benign adversary, any oracles π_j^t and π_i^s with destination address $\pi_i^s.\Pi = \pi_j^t$, and for all $\ell, \ell' \in [\psi]$ for which $\pi_i^s.\texttt{phases}[\ell].T$ and $\pi_j^t.\texttt{phases}[\ell'].T$ are matching conversations, it holds that

- $\pi_i^s.\texttt{phases}[\ell].\Lambda = \pi_j^t.\texttt{phases}[\ell'].\Lambda = \texttt{accept}$,
- $\pi_i^s.\texttt{phases}[\ell].\alpha = \pi_j^t.\texttt{phases}[\ell].\alpha$, and
- $\pi_i^s.\texttt{phases}[\ell].k = \pi_j^t.\texttt{phases}[\ell'].k$.

Confidentiality. All of our notions for secure multi-phase and renegotiable ACCE protocols will require confidentiality/integrity of the post-accept stage record layer in each uncorrupted phase. Intuitively, an adversary should not be able to guess the bit b used in the $\mathsf{Encrypt}/\mathsf{Decrypt}$ oracles in a phase where she has not impersonated the parties (i.e. corrupted the long-term secret keys before the phase accepted) or revealed the session key of the party or its peer.

Definition 3.13 (Confidentiality/integrity)**.** Suppose an algorithm $\mathcal{A}$ with running time τ interacts with a multi-phase ACCE protocol Π in the above execution environment and returns a tuple (i, s, ℓ, b'). If

C1. $\pi_i^s.\texttt{phases}[\ell].\Lambda = \texttt{accept}$; and

C2. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_i, \pi_i^s.\texttt{phases}[\ell].pk)$ before π_i^s accepted in phase ℓ; and

C3. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_j, \pi_i^s.\texttt{phases}[\ell].pk')$ before π_i^s accepted in phase ℓ, where $\pi_i^s.\Pi = \pi_j^t$; and

C4. $\mathcal{A}$ did not query $\mathsf{Reveal}(\pi_i^s, \ell)$; and

C5. $\mathcal{A}$ did not query $\mathsf{Reveal}(\pi_j^t, \ell')$, where $\pi_j^t = \pi_i^s.\Pi$ is π_i^s's intended communication partner, and ℓ' is any phase for which $\pi_j^t.\texttt{phases}[\ell'].T$ is a matching conversation to $\pi_i^s.\texttt{phases}[\ell].T$; and

C6. $|\Pr[\pi_i^s.\mathtt{phases}[\ell].b = b'] - 1/2| \geq \epsilon$,

then we say that $\mathcal{A}$ (t, ϵ)-*breaks confidentiality/integrity of* Π.

Secure multi-phase ACCE. First we state a straightforward extension of the ACCE model to protocols with multiple phases, but with essentially no security condition relating one phase to another. This definition captures the properties of TLS without any renegotiation countermeasures, and will be used as a stepping stone in our generic results in Chapter 6. For this simplest notion of authentication, an adversary should not be able to cause a phase to accept unless there exists a phase at the peer with a matching pre-accept handshake transcript, provided she has not impersonated the parties (i.e. corrupted long-term secret keys before the phase accepted).

In **A1** and **M** we will redefine the `NoMatch`-condition from [BR94a]. In **A2** we exclude leaking of the secret long-term keys of the accepting party (necessary for example to counter
acrshortkci attacks [Kra05]). In **A3** we exclude corruptions of the peer. In **A4** (only for server-only authentication), we ensure that the adversary only wins by making a client-oracle maliciously accept. In **A5** we exclude trivial attacks that exist for protocols with explicit key confirmation and probabilistic computations under the negotiated key.

Definition 3.14 (Secure multi-phase ACCE)**.** Suppose an algorithm $\mathcal{A}$ with running time τ interacts with a multi-phase ACCE protocol Π in the above execution environment and terminates. If, with probability at least ϵ, there exists an oracle π_i^s with $\pi_i^s.\mathit{\Pi} = \pi_j^t$ and a phase ℓ such that

A1. $\pi_i^s.\mathtt{phases}[\ell].\Lambda = \mathtt{accept}$; and

A2. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_i, \pi_i^s.\mathtt{phases}[\ell].pk)$ before π_i^s accepted in phase ℓ; and

A3. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_j, \pi_i^s.\mathtt{phases}[\ell].pk')$ before π_i^s accepted in phase ℓ; and

A4. if $\pi_i^s.\mathtt{phases}[\ell].\alpha = 0$ then $\pi_i^s.\rho = \mathsf{Client}$; and

A5. $\mathcal{A}$ did not query $\mathsf{Reveal}(\pi_j^t, \ell')$ for any ℓ' such that $\pi_j^t.\mathtt{phases}[\ell'].T$ is a matching conversation to $\pi_i^s.\mathtt{phases}[\ell].T$ before π_i^s accepted in phase ℓ; and

M. there is no ℓ' such that $\pi_j^t.\mathtt{phases}[\ell'].T$ is a matching conversation to $\pi_i^s.\mathtt{phases}[\ell].T$

then we say that $\mathcal{A}$ (t, ϵ)-*breaks authentication of* Π.

A protocol Π is a (t, ϵ)-*secure multi-phase ACCE protocol* if there exists no algorithm $\mathcal{A}$ that (t, ϵ)-breaks confidentiality/integrity (Definition 3.13) or authentication (as defined above) of Π.

The secure multi-phase ACCE definition when limited to a phase per session and a single key pair per party ($\psi = \gamma = 1$) collapses to the original ACCE definition.

Secure renegotiable ACCE. We next strengthen the authentication notion to include renegotiation. Intuitively, an adversary should not be able to cause a phase to accept unless there exists a phase at the peer with a matching pre-accept handshake transcript (M′(a)) and all previous phases' handshake and record layer transcripts match (M′(b)), provided she has not impersonated the parties in the current phase. We will show in Section 6.8 that TLS with our proposed countermeasure satisfies this definition.

Definition 3.15 (Secure renegotiable ACCE)**.** Suppose an algorithm $\mathcal{A}$ with running time τ interacts with a multi-phase ACCE protocol Π in the above execution environment and terminates. If, with probability at least ϵ, there exists an oracle π_i^s with $\pi_i^s.\mathit{\Pi} = \pi_j^t$ and a phase ℓ^* such that

A1-A5 as in Definition 3.14 with ℓ^*, and either

M′(a) $\pi_j^t.\mathtt{phases}[\ell^*].T$ is not a matching conversation to $\pi_i^s.\mathtt{phases}[\ell^*].T$ or

M′(b) for some $\ell < \ell^*$, $\pi_i^s.\mathtt{phases}[\ell].T \| \mathsf{RT}_s \| \mathsf{RT}_r \neq \pi_j^t.\mathtt{phases}[\ell].T \| \mathsf{RT}_r \| \mathsf{RT}_s$;

then we say that $\mathcal{A}$ (t, ϵ)-*breaks renegotiation authentication of* Π.

A protocol Π is a (t, ϵ)-*secure renegotiable ACCE protocol* if there exists no algorithm $\mathcal{A}$ that (t, ϵ)-breaks confidentiality/integrity (Definition 3.13) or renegotiation authentication (as defined above) of Π.

Weakly secure renegotiable ACCE. Unfortunately, TLS, when combined with SCSV/RIE countermeasures, does not satisfy Definition 3.15 because, as we will see in Section 6.4, revealing session keys in earlier phases allows the adversary to change the messages on the record layer in earlier phases, but the SCSV/RIE countermeasure will not detect this.

Of course, revealing earlier phases' session keys while that phase is active and still expecting detection when renegotiating later is a strong security property, and the lack of this property does not imply an attack in most scenarios. Our desire to characterize the renegotiable security of the SCSV/RIE countermeasure motivates a slightly weaker renegotiation notion: when previous phases' session keys are not revealed while that phase is active and the current phase's long-term secret keys are not corrupted, the adversary should not be able to cause a phase to accept unless there exists a phase at the peer with a matching pre-accept handshake transcript and all previous phases' handshake and record layer transcripts match.

Definition 3.16 (Weakly secure renegotiable ACCE)**.** Suppose an algorithm $\mathcal{A}$ with running time τ interacts with a multi-phase ACCE protocol Π in the above execution environment and terminates. If, with probability at least ϵ, there exists an oracle π_i^s with $\pi_i^s.\mathit{\Pi} = \pi_j^t$ and a phase ℓ^* such that all conditions from Definition 3.15, as well as the following additional conditions are satisfied:

A6. $\mathcal{A}$ did not issue a $\mathsf{Reveal}(\pi_i^s, \ell)$ query before π_i^s accepted in phase $\ell + 1$, for every $\ell < \ell^*$, and

A7. $\mathcal{A}$ did not issue a $\mathsf{Reveal}(\pi_j^t, \ell)$ query before π_i^s accepted in phase $\ell + 1$, for every $\ell < \ell^*$;

then we say that $\mathcal{A}$ (t, ϵ)-*breaks weak renegotiation authentication of* Π.

A protocol Π is a (t, ϵ)-*weakly secure renegotiable ACCE protocol* if there exists no algorithm $\mathcal{A}$ that (t, ϵ) breaks confidentiality/integrity (Definition 3.13) or weak renegotiation authentication (as defined above) of Π.

Remark 16. While conditions **A6** and **A7** prohibit the adversary from revealing encryption keys of previous phases while active *for the purposes of breaking authentication*, the confidentiality/integrity aspect of Definition 3.16 still places no such restriction on previous encryption keys being revealed.

Tagged-ACCE security model

In this section we introduce a variant of the ACCE model from which we can prove a generic result on the renegotiable security of ACCE protocols implementing the countermeasures discussed in Chapter 6. The *tagged-ACCE* execution environment is an extension of the ACCE security model to allow for providing arbitrary tags and is derived from the multi-phase ACCE definition (Definition 3.14) by limiting it to a single phase ($\psi = 1$) and at most one public key per party ($\gamma = 1$).

The phases log `phases` is extended with an additional per-phase variable tag.

- $\mathsf{Send}(\pi_i^s, m)$. The adversary can specify an arbitrary tag during session initialization.
  - If $m = (\texttt{newphase}, \alpha, tag)$, oracle π_i^s sets its internal variable $\rho \leftarrow \mathsf{Client}$, sets authentication mode α, stores tag, and responds with the first protocol message.

– If $m = (\texttt{ready}, \alpha, tag)$, oracle π_i^s sets $\rho \leftarrow \mathsf{Server}$, authentication mode α, stores tag, and responds with the next protocol message, if any.

The freshness and winning conditions of tagged-ACCE are unchanged from the ACCE definitions with perfect forward secrecy from Section 3.2.2.

Tagged-ACCE-fin security model

We will work with a further variant, *tagged-ACCE-fin*, which is not a fully general security model but instead is tied specifically to generic TLS protocols of the form given in Figure 6.3 (cf. page 149). It adds the following query:

- $\mathsf{RevealFin}(\pi_i^s)$: If $\Lambda = \texttt{accept}$, then return the fin_C and fin_S values sent/received by the queried oracle. Return $\emptyset$ otherwise.

The following queries are modified:

- $\mathsf{Encrypt}(\pi_i^s, ctype, m_0, m_1, \mathsf{len}, H)$: The adversary is not prevented from making queries with $ctype = \texttt{control}$.
- $\mathsf{Decrypt}(\pi_i^s, C, H)$: No semantic meaning is associated with $ctype = \texttt{control}$ messages. In other words, line 5 of Figure 3.2 is removed.

We extend the Encrypt and Decrypt queries to allow the adversary to send and receive messages on the encrypted channel with content type `control` . The freshness and winning conditions of tagged-ACCE-fin are unchanged from ACCE.

Remark 17. Revealing the `Finished` messages is very specific to the TLS protocol family and is not necessarily relevant for other protocols. Imagine, for example, a variant of the SCSV/RIE countermeasure where a separate hash of the complete transcript as it was sent over the channel is used as an authenticator. Since this value can be computed by any passive adversary, leaking this value could not affect security.

Renegotiation Protocols without Forward Secrecy.

Due to the high complexity of the model for renegotiable protocols we restrict our results to protocols with perfect forward secrecy. However, in a future work one could easily extend our definitions to allow for analyzing protocols without forward secrecy, such as protocols that use an RSA-based key transport mechanism. In the following we briefly discuss the necessary changes to our definitions.

AUTHENTICATION For renegotiation authentication, public keys used in phase ℓ cannot be corrupted while the post-accept stage of phase ℓ is still active, as it could allow the adversary to compute the session key and thus inject messages undetectably. One could add the following restrictions to Definition 3.16 to consider protocols without forward secrecy:

A7. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_i, \pi_i^s.\texttt{phases}[\ell].pk)$ before π_i^s accepted in phase $\ell + 1$, for every $\ell < \ell^*$; and

A8. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_j, \pi_i^s.\texttt{phases}[\ell].pk')$ before π_i^s accepted in phase $\ell + 1$, for every $\ell < \ell^*$, where $\pi_i^s.\Pi = \pi_j^t$.

Confidentiality. For confidentiality/integrity, public keys used in phase ℓ can never be corrupted, as it could allow the adversary to compute the session key and distinguish ciphertexts. One could replace the following restrictions in Definition 3.13 to consider protocols without forward secrecy:

C2′. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_i, \pi_i^s.\texttt{phases}[\ell].pk)$; and

C3′. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_j, \pi_i^s.\texttt{phases}[\ell].pk')$.

	Secure multi-phase ACCE (Defn. 3.14)	**Weakly secure renegotiable ACCE** (Defn. 3.16)	**Secure renegotiable ACCE** (Defn. 3.15)
Secure against Ray–Dispensa-type attack	×	✓ with query restrictions **A6**,**A7**	✓
Authentication			
A2. Corrupt *pk* before acceptance	not allowed	not allowed	not allowed
A3. Corrupt peer's *pk* before acceptance	not allowed	not allowed	not allowed
A5. Reveal session keys **during** active handshake	not allowed	not allowed	not allowed
A6. Reveal session keys of previous phases	allowed	not allowed	allowed
A7. Reveal session keys of previous phases	allowed	not allowed	allowed
M. every phase that accepts has a matching handshake transcript at *some* phase of the peer	implied		
M′(a) every phase that accepts has a matching handshake transcript at *the same* phase of the peer		implied	implied
M′(a) when a phase accepts, handshake and record layer transcripts in *all previous phases* equal those at the peer		implied	implied
Confidentiality/integrity (Defn. 3.13)	implied	implied	implied
TLS_∗ without countermeasures	—	× (Sect. 6.3)	× (Sect. 6.3)
Tagged-ACCE-fin-secure TLS_∗ with SCSV/RIE countermeasure	—	✓ (Thm. 1)	—
TLS_RSA_∗ with SCSV/RIE countermeasure	?[1]	× (Sec. 6.7.1) / ?[1]	× (Sect. 6.4)
TLS-DHE with SCSV/RIE countermeasure	✓ (Cor. 2)	✓ (Cor. 2)	× (Sect. 6.4)
Secure multi-phase TLS_∗ with new (Sect. 6.8) countermeasure	—	—	✓ (Thm. 6.4)
TLS_RSA_∗ with new (Sect. 6.8) countermeasure	?[1]	× (Sec. 6.7.1) / ?[1]	× (Sec. 6.7.1) / ?[1]
TLS-DHE with new (Sect. 6.8) countermeasure	✓ (Thm. 6.4)	✓ (Thm. 6.4)	✓ (Thm. 6.4)

[1] TLS_RSA_∗ key transport ciphersuites may be able to be shown secure under notions with suitable restrictions on forward secrecy.

Table 3.1: Summary of security notions and results on TLS

4 Generic Compiler for (Mutually) Authenticated Key Exchange Protocols

This chapter includes results from joint work with Tibor Jager, Sven Schäge and Jörg Schwenk, published at ASIACRYPT'09 [JKSS10]. In the following we describe a compiler that allow us to combine arbitrary key agreement protocols (which need only be secure against passive adversaries) with a digital signature scheme to form a Authenticated Key Exchange (AKE) protocol as defined in Section 3.2.1 without relying on Random Oracles.

The compiler allows for a modular design of new AKE protocols, using existing protocols (e.g. TLS, IPSec IKE) or new ones (e.g. zero-knowledge authentication, group signatures). The formal security proof is simplified considerably, since the security of key agreement and authentication protocols can be proven separately, and our theorems yield the security of the combined protocol.

Related Work. At CRYPTO'03, Katz and Yung presented a first scalable compiler that transforms any passively secure group key exchange protocol to an actively secure AKE [KY03]. Their compiler adds one round and constant size (per user) to the original scheme, by appending an additional signature to each message of the protocol. Our compiler, while limited to two-party key exchange protocols, only adds two signatures, independent of the number of rounds of the key exchange protocol.

There also exist several results on group key exchange for different setup conditions and security requirements, e.g. the works by Manulis *et al.*, see also [BM08a, BM08b, Man09, BBM09, MPT10].

4.1 Authenticated Key Exchange Compiler

Let us now describe our generic AKE compiler. The compiler takes as input the following building blocks (which have been defined in Chapter 2).

- A key-exchange protocol KE,
- a digital signature scheme SIG,
- a message authentication code MAC,

- and a pseudorandom function PRF.

Let $\lambda = \lambda(\kappa)$ be the length of the nonces. In the following let T_x^i denote the content of transcript T_x as received by an oracle of party P_i.

Remark 18. Note, that there is no long-term value (e.g. long-term public keys) which is used in several executions of the protocol and all parameters need to be generated freshly for each session.[18] This is mainly due to the fact, that we have to embed a challenge transcript into the Test session without knowing the (long-term) secret keys involved.

The compiled protocol proceeds (examplary between two oracles π_i^s and π_j^t) as follows (see also Figure 4.1).

1. Oracles π_i^s and π_j^t run the key exchange protocol. For instance, both oracles may run the well-known Diffie-Hellman protocol [DH76]. Throughout this protocol run, both oracles compute a key k and record a transcript that consists of the list of all messages sent and received.

2. The key k computed by KE is used to derive two distinct keys $K = \mathsf{PRF}(k, \text{'KE'})$ and $K_{\mathsf{mac}} = \mathsf{PRF}(k, \text{'MAC'})$, where 'KE' and 'MAC' are some arbitrary fixed constants such that 'KE' $\neq$ 'MAC'.[19]

3. Then π_i^s samples a random nonce $r_i \xleftarrow{\$} \{0,1\}^\lambda$, and sends it to π_j^t, π_j^t samples $r_j \xleftarrow{\$} \{0,1\}^\lambda$ and sends it to π_i^s.

4. Oracle π_i^s computes a signature $\sigma_i \xleftarrow{\$} \mathsf{SIG.Sign}(sk_i, \mathsf{T}_1^i)$ under π_i^s's secret key sk_i, where $\mathsf{T}_1^i = (\mathsf{T}_{\mathsf{KE}}^i || r_i || r_j)$ is the transcript of all messages sent and received by π_i^s so far. Then π_j^t computes a signature over the transcript $\mathsf{T}_1^j = (\mathsf{T}_{\mathsf{KE}}^j || r_i || r_j)$ of all messages sent and received by π_j^t. Let $\mathsf{T}_2^i = (\sigma_i || \sigma_j)$ denote the signatures sent and received by π_i^s, and $\mathsf{T}_2^j = (\sigma_i || \sigma_j)$ be the signatures sent and received by π_j^t.

5. π_i^s sends a MAC $t_i = \mathsf{MAC}(K_{\mathsf{mac}}, \mathsf{T}_2^i || 0)$ over transcript T_2^i using the key K_{mac} computed in 2. π_j^t replies with $t_j = \mathsf{MAC}(K_{\mathsf{mac}}, \mathsf{T}_2^j || 1)$.

6. π_i^s accepts, if $\mathsf{SIG.Vfy}(pk_j, \mathsf{T}_1^i, \sigma_j) = 1$ and $t_j = \mathsf{MAC}(K_{\mathsf{mac}}, \mathsf{T}_2^i || 1)$, that is, if σ_j is a valid signature for T_2^i under π_j^t's verification key pk_j and if w_j is a valid MAC under key K_{mac} for $\mathsf{T}_2^i || 1$. π_j^t accepts if it holds that $\mathsf{SIG.Vfy}(pk_i, \mathsf{T}_1^j, \sigma_i) = 1$ and $w_i = \mathsf{MAC}(K_{\mathsf{mac}}, \mathsf{T}_2^j || 0)$. Finally, if both oracles accept then the key K is returned.

[18] For instance, to instantiate the key exchange protocol with an encrypted key transport scheme (in a secure matter according to our model and definition), we require the used public keys to be generated freshly at the beginning of each protocol execution!

[19] Note that we assume here implicitly, that the output key space of KE matches the input key space of PRF. This fact is not only important for correctness, but also for the security proof.

Observe that the signatures and MACs are verified using the *internal* transcripts of oracles π_i^s and π_j^t. The intention behind the idea of embedding the transcripts in the protocol is to detect any changes that an active adversary makes to the messages sent by π_i^s and π_j^t. Informally, in the two-layer authentication consisting of the signature scheme and MAC, the signature is used to authenticate users and thwart MITM attacks on the key-exchange protocol, while the MAC is used as an implicit 'key confirmation' step to avoid UKS attacks [CBH05b, CBH05a]

This allows us to prove security requiring only pretty weak security properties from the utilized building blocks, namely we require that KE is secure against *passive* adversaries only, that the digital signatures are existential unforgeable under (non-adaptive) chosen-message attacks, and that the MAC and PRF meet their standard security notions.

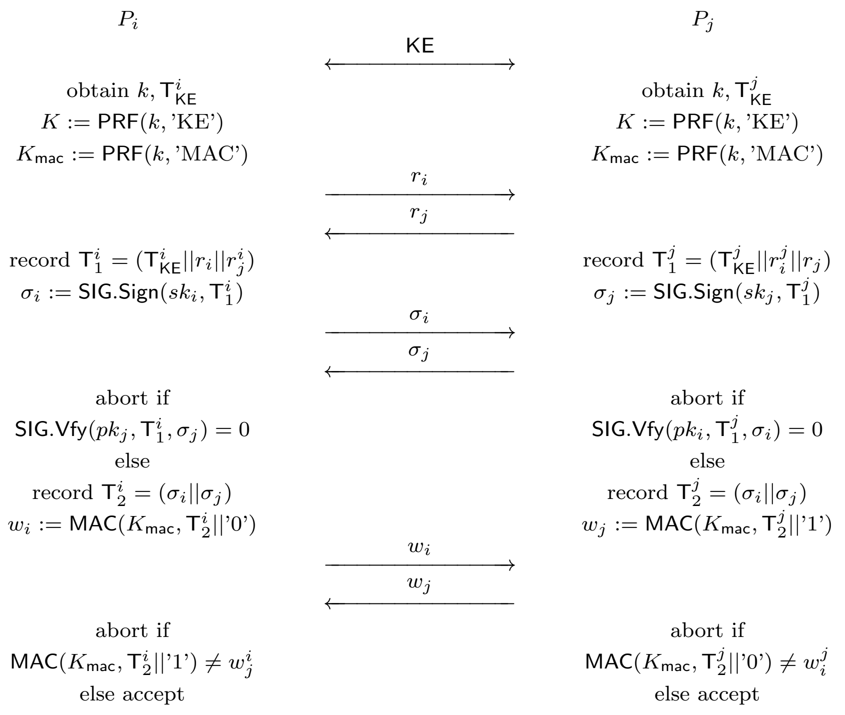
P_i
P_j
KE
obtain $k, \mathsf{T}^i_{\mathsf{KE}}$
$K := \mathsf{PRF}(k, \text{'KE'})$
$K_{\mathsf{mac}} := \mathsf{PRF}(k, \text{'MAC'})$
obtain $k, \mathsf{T}^j_{\mathsf{KE}}$
$K := \mathsf{PRF}(k, \text{'KE'})$
$K_{\mathsf{mac}} := \mathsf{PRF}(k, \text{'MAC'})$
r_i
r_j
record $\mathsf{T}^i_1 = (\mathsf{T}^i_{\mathsf{KE}}||r_i||r^i_j)$
$\sigma_i := \mathsf{SIG.Sign}(sk_i, \mathsf{T}^i_1)$
record $\mathsf{T}^j_1 = (\mathsf{T}^j_{\mathsf{KE}}||r^j_i||r_j)$
$\sigma_j := \mathsf{SIG.Sign}(sk_j, \mathsf{T}^j_1)$
σ_i
σ_j
abort if
$\mathsf{SIG.Vfy}(pk_j, \mathsf{T}^i_1, \sigma_j) = 0$
else
record $\mathsf{T}^i_2 = (\sigma_i||\sigma_j)$
$w_i := \mathsf{MAC}(K_{\mathsf{mac}}, \mathsf{T}^i_2||\text{'0'})$
abort if
$\mathsf{SIG.Vfy}(pk_i, \mathsf{T}^j_1, \sigma_i) = 0$
else
record $\mathsf{T}^j_2 = (\sigma_i||\sigma_j)$
$w_j := \mathsf{MAC}(K_{\mathsf{mac}}, \mathsf{T}^j_2||\text{'1'})$
w_i
w_j
abort if
$\mathsf{MAC}(K_{\mathsf{mac}}, \mathsf{T}^i_2||\text{'1'}) \neq w^i_j$
else accept
abort if
$\mathsf{MAC}(K_{\mathsf{mac}}, \mathsf{T}^j_2||\text{'0'}) \neq w^j_i$
else accept

Figure 4.1: Compiler for secure authenticated key exchange protocols

4.2 Security Analysis

Theorem 4.1. *Let λ be the length of the nonces r_i and r_j. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the signature scheme used to sign messages in the protocol is $(t, \epsilon_{\mathsf{sig}}, q_{\mathsf{sig}})$-secure, the message authentication code* MAC *is $(t, \epsilon_{\mathsf{mac}}, q_{\mathsf{mac}})$-secure, the key exchange protocol is $(t, \epsilon_{\mathsf{ke}})$-secure, and $q_{\mathsf{prf}}, q_{\mathsf{mac}} \leq 2$, and $q_{\mathsf{sig}} \leq \eta$.*

Then for any adversary that $(t', \epsilon_{\mathsf{AKE}})$-breaks the compiled AKE protocol in the sense of Definition 3.3 with $t \approx t'$ holds that

$$\begin{aligned}\epsilon_{\mathsf{AKE}} &\leq \epsilon_{\mathsf{A}} + \epsilon_{\mathsf{KE}} \\ &\leq 2\eta\phi\left(\frac{\eta\phi}{2^\lambda} + \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{ke}} + \epsilon_{\mathsf{mac}}\right).\end{aligned}$$

We prove the above theorem by two lemmas. Lemma 4.1 states that the AKE protocol meets property 1) of Definition 3.3, Lemma 4.2 states that it meets property 2) of Definition 3.3.

4.2.1 Authentication

Lemma 4.1. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s that accepts maliciously is at most*

$$\epsilon_{\mathsf{A}} \leq \eta\phi\left(\frac{\eta\phi}{2^\lambda} + \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{ke}} + \epsilon_{\mathsf{mac}}\right),$$

where all quantities are defined as stated in Theorem 4.1.

Proof. The proof proceeds in a *sequence of games*, following [BR06, Sho04]. The first game is the real security experiment. By assumption there exists an adversary $\mathcal{A}$ that breaks the security of the above protocol. We then describe several intermediate games that step-wisely modify the original game. Next we show that in the final security game the adversary has only negligible advantage in breaking the security of the protocol. Let $\mathsf{break}_\delta^{(a)}$ be the event that the adversary makes an oracle maliciously accept in the sense of Definition 3.3 in Game δ. In the following let $\mathsf{T}_X^{i,s}$ denote the transcript T_X as recorded by π_i^s and likewise $\mathsf{T}_X^{j,t}$ the transcript T_X as recorded by by π_j^t.

Game 0. This is the original security game. Assume an adversary $\mathcal{A}$ breaking Property 1 of Definition 3.3 with probability ϵ_{A}, i.e.

$$\Pr[\mathsf{break}_0^{(a)}] = \epsilon_{\mathsf{A}}.$$

In the sequel we will show that ϵ_{A} is negligible for any algorithm $\mathcal{A}$.

Game 1. This game proceeds exactly like the previous game, except that the simulator aborts if there exists any oracle π_i^s that chooses a random nonce r_i or r_j which is not unique. More precisely, the game is aborted if the adversary ever makes a first Send query to an oracle π_i^s, and the oracle replies with random nonce r_i or r_j such that there exists some other oracle $\pi_{i'}^{s'}$ which has previously sampled the same nonce.

In total less than $\eta\phi$ nonces r_i and r_j are sampled, each uniformly random from $\{0,1\}^\lambda$. Thus, the probability that a collision occurs is bounded by $(\eta\phi)^2 2^{-\lambda}$, which implies

$$\Pr[\mathsf{break}_0^{(a)}] \leq \Pr[\mathsf{break}_1^{(a)}] + \frac{(\eta\phi)^2}{2^\lambda}.$$

Note that now each oracle has a unique nonce r_C or r_S, which is included in the signatures. We will use this to ensure that each oracle that accepts with non-corrupted partner has a *unique* partner oracle.

Game 2. We try to guess which oracle will be the first oracle to accept maliciously. If our guess is wrong, i.e. if there is another oracle that accepts before, then we abort the game.

Technically, this game is identical to Game 1, except for the following. The challenger guesses two random indices $(i^*, s^*) \xleftarrow{\$} [\phi] \times [\eta]$. If there exists an oracle π_i^s that 'accepts' maliciously, and $(i,s) \neq (i^*, s^*)$, then the challenger aborts the game. Note that with probability $1/(\eta\phi)$ we have $(i,s) = (i^*, s^*)$, and thus

$$\Pr[\mathsf{break}_1^{(a)}] = \eta\phi \cdot \Pr[\mathsf{break}_2^{(a)}].$$

Note that in this game the adversary can only break the security of the protocol, if oracle $\pi_{i^*}^{s^*}$ is the first oracle that 'accepts' maliciously, as otherwise the game is aborted. Thus, his winning probability is reduced by a factor of $\eta\phi$.

Game 3. This game proceeds exactly like the previous game, except that the simulator aborts if π_i^s accepts and $\mathsf{T}_1^{i,s} \neq \mathsf{T}_1^{j,t}$.

Claim 3. We claim that

$$\Pr[\mathsf{break}_2^{(a)}] \leq \Pr[\mathsf{break}_3^{(a)}] + \phi \cdot \epsilon_{\mathsf{sig}}.$$

by the EUF-CMA security of the digital signature scheme. The proof of this claim exploits that the transcripts $\mathsf{T}_1^{i,s} = (\mathsf{T}_{\mathsf{KE}}^{i,s} || r_i || r_j^i)$ and $\mathsf{T}_1^{j,t} = (\mathsf{T}_{\mathsf{KE}}^{j,t} || r_i^j || r_j)$ are unique with overwhelming probability since r_i and r_j are unique due to Game 1.

Technically we first guess the party P_j, such that $\pi_{i^*}^{s^*}$ receives a signature σ_j that can be verified under pk_j, and implement the public key pk from the signature challenge into pk_j. In order to make $\pi_{i^*}^{s^*}$ accept, the adversary needs to present a signature over $\mathsf{T}_1^{i,s}$. However, the partner oracle π_j^t will output a signature over $\mathsf{T}_1^{j,t} \neq \mathsf{T}_1^{i,s}$, which the adversary cannot use. Hence, if $\pi_{i^*}^{s^*}$ accepts in this scenario, the adversary must have output a signature over $\mathsf{T}_1^{i,s}$ which was never output by π_j^t.

Game 4. This game proceeds exactly like the previous game, except that the simulator now chooses a uniformly random key $\hat{k}$ to derive K_{mac} and K as $K_{\mathsf{mac}} = \mathsf{PRF}(\hat{k},\ \text{'MAC'})$ and $K = \mathsf{PRF}(\hat{k},\ \text{'KE'})$.

Claim 4. We claim that

$$\Pr[\mathsf{break}_3^{(a)}] \leq \Pr[\mathsf{break}_4^{(a)}] + \epsilon_{\mathsf{ke}}.$$

by the security of KE against passive adversaries. Note that we must have $\mathsf{T}_1^{i,s} = \mathsf{T}_1^{j,t}$ if π_i^s or π_j^t accept, as otherwise we abort due to Game 3. This implies that the adversary forwards all messages of KE without altering anything. Technically we implement the key exchange challenger in $\pi_{i^*}^{s^*}$ and its partner oracle π_j^t. The partner oracle can be easily determined, as it uniquely maintains the same transcript $\mathsf{T}_1^{j,t} = \mathsf{T}_1^{i,s}$ due to Game 3. If the key k is derived from the key exchange, we are in Game 3, while if k is drawn randomly we are in Game 4. We can thus directly use an adversary distinguishing Game 4 from Game 3 to break the security of KE against passive adversaries.

Game 5. This game proceeds exactly like the previous game, except that the simulator now chooses a uniformly random key $\widetilde{K_{\mathsf{mac}}}$ (instead of K_{mac}, which is computed as $K_{\mathsf{mac}} := \mathsf{PRF}(k, \text{'MAC'})$) to compute w_i and w_j.

Claim 5. We claim that

$$\Pr[\mathsf{break}_4^{(a)}] \leq \Pr[\mathsf{break}_5^{(a)}] + \epsilon_{\mathsf{prf}}.$$

by the security of the pseudorandom function PRF. We exploit that we have exchanged the 'real' key k computed in KE with a 'random' key $\tilde{k}$ in Game 4. Technically we implement the PRF challenger in $\pi_{i^*}^{s^*}$ and its partner oracle $\pi_{j^*}^{t^*}$. If K_{mac} is computed as $\mathsf{PRF}(\tilde{k}, \text{'MAC'})$, we are in Game 4. If the key K_{mac} is drawn randomly, we are in Game 5. Any adversary that can distinguish between Games 5 and 4 can thus be used to break the security of the PRF.

Game 6. This game proceeds exactly like the previous game, except that the simulator aborts if π_i^s or π_j^t accepts and $\mathsf{T}_2^{i,s} \| w_i \neq \mathsf{T}_2^{j,t} \| w_j$.

Claim 6. We claim that

$$\Pr[\mathsf{break}_5^{(a)}] \leq \Pr[\mathsf{break}_6^{(a)}] + \epsilon_{\mathsf{mac}}.$$

Recall that in Game 6 we must have $\mathsf{T}_1^{i,s} = \mathsf{T}_1^{j,t}$ due to our abort condition from Game 3, and that we have replaced the key k computed in KE with a uniformly random key $\tilde{k}$ in Game 4 and the key K_{mac} used to compute the MACs in this protocol instance with a uniformly random key $\widetilde{K_{\mathsf{mac}}}$ in Game 5. Thus, if we have

$\mathsf{T}_2^{i,s} \| w_i \neq \mathsf{T}_2^{j,t} \| w_j$, then the adversary must have forged a MAC to make π_i^s or π_j^t accept. We can then use the adversary to break the security of MAC as follows. We implement a MAC challenger in π_i^s and π_j^t and use this challenger to compute MACs over $\mathsf{T}_2^{i,s}$ and $\mathsf{T}_2^{j,t}$ when necessary. Now when oracle π_i^s (or π_j^t) receives w_j (w_i respectively), it verifies if it matches the MAC it has computed itself (or as in this case learned from the MAC challenger). If the simulator aborts, we have that oracle π_i^s (or π_j^t) accepted, but $\mathsf{T}_2^{i,s} \| w_i \neq \mathsf{T}_2^{j,t} \| w_j$. As MAC is a deterministic function, it makes no difference whether $\mathsf{T}_2^{i,s} \neq \mathsf{T}_2^{j,t}$ or $w_i \neq w_j$. One oracle (in order to accept) must have received a MAC that was not computed by its partner oracle. We can then forward this MAC value to the MAC challenger and break its security.

Note that in Game 6 an oracle accepts *only if* there exists another oracle having a matching conversation, as the game is aborted otherwise. Thus, no adversary can break Property 1 of Definition 3.3 in Game 6.

Collecting probabilities from Game 0 to 6 yiels Lemma 4.1

□

4.2.2 Indistinguishability of Keys

Lemma 4.2. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the* Test*-challenge correctly is at most* $1/2 + \epsilon_{\mathsf{KE}}$ *with*

$$\epsilon_{\mathsf{KE}} \leq \epsilon_{\mathsf{A}},$$

where all quantities are defined as stated in Theorem 4.1.

Proof.(Sketch). We proceed in a sequence of games which is very similar to the sequence of games We merely add one further game. Let $\mathsf{break}_\delta^{(b)}$ be the event that the adversary answers the Test-challenge correctly in the sense of Definition 3.3 in Game δ and let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta^{(b)}] - 1/2$ denote the *advantage* of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This is the original security game. We assume an adversary $\mathcal{A}$ breaking Property 2 of Definition 3.3 with probability $1/2 + \epsilon_{\mathsf{KE}}$.

Game 1. In this game, we make the same modifications as in Games 1 to 6 in the proof of Lemma 4.1. With the same arguments as before, we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{A}},$$

Game 2. This game proceeds exactly like the previous game except that the simulator now chooses K uniformly at random from the keyspace. We exploit that we have already exchanged the function $\mathsf{PRF}(k, \cdot)$ with a uniformly random function $\tilde{F}(\cdot)$ in Game 5.

Claim 2. We claim that

$$\mathsf{Adv}_1 = \mathsf{Adv}_2.$$

In Game 2, the adversary receives a uniformly random key K. However, by collecting probabilities from Game 0 to 2 we obtain that Game 2 is indistinguishable from Game 0 (except for some negligible probability), which proves indistinguishability of 'real' from 'random' keys. Thus, the protocol meets Property 2 of Definition 3.3. □

5 On the Provable Security of TLS

This chapter includes results from joint work with Tibor Jager, Sven Schäge and Jörg Schwenk, one part published at CRYPTO'12 [JKSS12] and the other part currently in submission. It comprises a brief description of core of the TLS handshake protocol (version 1.2) and our security results on all relevant TLS ciphersuites. As previously stated we rely on results of Fouque *et al.* [FPZ08] on the security of the PRF used in TLS 1.2, so our results are restricted to versions of TLS that support suitable primitives. We first show that a truncated version of TLS, in which the `Finished` messages are sent in plain, yields a secure AKE protocol. Then we show, that the ciphersuites TLS-DHE, TLS-SDH and TLS-RSA yield secure ACCE protocols.

5.1 A Brief Introduction to Transport Layer Security (TLS)

In this section, we present the core of the current version of the TLS protocol - TLS 1.2 [DR08]. We first give a general overview of the structure of the protocol and then describe the messages sent during a protocol execution in detail.

5.1.1 Overview

The TLS protocol consists basically of two stages. A first stage in which parties authenticate to each other and establish cryptographic key material and a second stage in which the previously established keys are used to encrypt transmitted data in order to ensure confidentiality. We often refer to the first stage as handshake protocol and to the second stage as Record Layer.

The major changes from TLS 1.0 [DA99] and TLS 1.1 [DR06] to TLS 1.2 include a different error handling procedure to prevent certain attacks (e.g. TLS 1.1 introduced an explicit Initialization Vector and handled padding errors differently in order to prevent so-called Cipher Block Chaining (CBC) attacks) and, perhaps most importantly, switching from a (fixed) MD5/SHA-1 hash function combination in the PRF to ciphersuite-specific hash functions, with SHA-256 being the new standard.

A ciphersuite determines the set of algorithms to be used in the protocol execution. The two communicating parties agree on the key exchange algorithm (according to standard mostly RSA key transport or Diffie-Hellman (DH) key exchange) and parameters (e.g. key length), the MAC algorithm, the digital signature scheme and the symmetric encryption scheme and its mode of operation used to encrypt data in the second stage, i.e. after the handshake protocol.

Indistinguishability of TLS Application Keys.

The full TLS Handshake does not provide indistinguishable keys due to an interleaving of the key exchange part of TLS (the TLS Handshake protocol) and the data encryption in the TLS record layer, as the two stages are not completely seperated and independent of each other. This interleaving provides a 'check value' that allows to test whether a given key is 'real' or 'random'. More precisely, the final messages of the TLS Handshake protocol (the `Finished` messages), which are essential to provide security against active adversaries like e.g. MITM attackers, are first prepended with constant byte values (which provides us with known plaintext), then integrity protected by a MAC (which is instantiated with a pseudo-random function) and encrypted with the keys obtained from the TLS Handshake protocol. Thus, whenever an adversary receives a challenge key in response to a Test query, he can try to decrypt the `Finished` message and check validity of the MAC. If this succeeds, he will output 'real', and otherwise 'random'. Even changing the behavior of the Test query to only return the decryption keys (and not the MAC keys) does not help, since the adversary could still use the known plaintext bytes to answer the Test query successfully. Therefore it is impossible to prove the full TLS Handshake protocol secure in any security model based on indistinguishability of keys, such as the AKE model described in Section 3.2.1.

5.1.2 Related Work on the Cryptographic Security of TLS

Because of its eminent role, TLS and its building blocks have been subject to several security analyses. In 1996, Schneier and Wagner presented several minor flaws and some new active attacks against SSL 3.0 [WS96]. Starting with the famous Bleichenbacher attack [Ble98], many papers focus on various versions of the PKCS#1 standard [Kal98] that defines the encryption padding used in TLS with RSA-encrypted key transport [CJNP00, JK02, KP09, KOS10]. At Crypto'02, Johnson and Kaliski showed that a simplified version of TLS with padded RSA is Indistinguishable under Chosen-Ciphertext Attacks (IND-CCA) secure when modeling TLS as a 'tagged key-encapsulation mechanism' (TKEM) [JK02] under the strong non-standard assumption that a 'partial RSA decision oracle' is available.

Automated Proof Techniques. In an independent line of research, several works analyzed (simplified versions of) TLS using automated proof techniques in the Dolev-Yao model [DY83]. Proofs that rely on the Dolev-Yao model view cryptographic operations as deterministic operations on abstract algebras. There has been some work on simplified TLS following the theorem proving and model checking approach, i.e. Mitchell *et al.* used a finite-state enumeration tool named Murphi [Mit98] while Ogata and Futatsugi used the interactive theorem prover OTS/CafeObj [OF05]. Paulson used the inductive method and the theorem prover Isabelle [Pau99]. Unfortunately it is not known if these proofs are actually cryptographically sound. Bhargavan *et al.* [BFCZ08] go two steps farther: First, they automatically derive their formal model from the source code of an TLS implementation, and second they try to automatize computational proofs using the CryptoVerif tool. Chaki and Datta [CD09] also use source code of TLS, automatically find a weakness in OpenSSL 0.9.6c, and claim that SSL 3.0 is correct.

Universal Composability Approach. In 2008, Gajek *et al.* presented the first security analysis of the complete TLS protocol, combining Handshake and record layer, in the UC framework [Can01] for all three key exchange protocols static Diffie-Hellman, ephemeral signed Diffie-Hellman, and encrypted key transport [GMP$^+$08a]. The nonces r_C and r_S exchanged between client and server can be seen as an instantiation of the protocol of Barak *et al.* [BLR04] to agree on a globally unique session ID. However, the ideal functionalities described in this paper are strictly weaker than the security guarantees we expect from TLS: For the Handshake part, only unauthenticated key exchange is modelled ($\mathcal{F}_{KE}$), and thus the secure communication channel functionality ($\mathcal{F}_{SCS}$) only guarantees confidentiality, not authenticity of endpoints. The paper further assumes that RSA-OAEP is used for encrypted key transport, which is not the case for current versions of TLS.

Küsters and Tuengerthal [KT11] claim to prove composable security for TLS assuming only local session identifiers, but leave out all details of the proof and only point to [GMP$^+$08a].

Proofs in the Random Oracle Model. Morissey *et al.* [MSW08] analyzed, in a paper that is closest to our results, the security of the truncated TLS Handshake protocol (cf. Section 5.2) in the Random Oracle model and provided a modular proof of security for the established application keys. They make extensive use of the Random Oracle model to separate the three layers they define in the TLS Handshake, and to switch from computational to indistinguishability based security models. The proof of Morissey *et al.* proceeds in three steps, and the order of messages of the TLS Handshake is slightly changed to better separate these three steps. They first consider a very weak class of passively secure key exchange protocols where the session key cannot *be computed* from the session transcript. As an example, when considering encrypted key transport (of the premaster secret) this requirement can easily be fulfilled if the employed public key encryption scheme is One-Wayness under Chosen-Plaintext Attacks (OW-CPA) secure. Next they define a slightly stronger security notion that additionally protects against UKS attacks and show that it applies to the master secret key exchange of TLS. As before security of the key is defined in a one-way sense. In the last step they show that the 'application keys' (i.e. the encryption keys and MAC keys) produced by TLS fulfill the standard notion of security, namely *indistinguishability* from random values. The use of the ROM is justified by the authors by the fact that it seems impossible to prove the PKCS#1 v1.5 based ciphersuites of TLS secure in the standard model. This argumentation does not affect our work, since we only consider Diffie-Hellman-based ciphersuites.

The modular proof strategy used in this paper is essentially bound to the ROM, since secure protocols for the premaster phase only yield secure protocols for the master phase if the master secret is derived from the premaster secret by evaluating a Random Oracle. Thus the ROM is used not only to allow for a security proof for TLS-RSA ciphersuites, but also enables a modular proof.

However, this model has been criticized fundamentally in several works starting with [CGH98]. (In this context we note that the informal argumentation for the necessity of the ROM, namely that a security proof in the standard model would imply an encryption scheme under the RSA assumption secure against Chosen-Ciphertext Attacks (CCA), what was believed to be a hard, long-standing open problem at the time of publication of [MSW08], has been invalidated recently by Hofheinz and Kiltz which presented a CCA secure encryption scheme that is secure under the factoring assumption which is weaker than the RSA assumption [HK09b].)

Security of the TLS Symmetric Encryption Scheme. In a very recent work, Paterson, Ristenpart, and Shrimpton [PRS11] introduce the notion of length-hiding authenticated encryption, which aims to capture the properties from the TLS record layer. Most importantly, they were able to show that CBC-based ciphersuites of TLS 1.1 and 1.2 meet this security notion. This work matches nicely our results on the TLS Handshake protocol. Their paper extends the seminal work of Bellare and Namprempre [BN00, BN08] on authenticated encryption, and on the analysis of different Mac-then-Encode-then-Encrypt (MEE) schemes analyzed by Krawczyk [Kra01] and Maurer and Tackmann [MT10].

This is an important building block for our work, since it allows to capture the precise notion of a TLS-based Authenticated and Confidential Channel Establishment (ACCE). TLS-ACCE is used implicitly in many security applications (e.g. the Same Origin Policy of webbrowsers), and explicitely stating the security guarantees offered by TLS-ACCE is an important step towards a future anaysis of these protocols.

Recent Developments. In 2011 Brzuska *et al.* [BFWW11] presented new security definitions for key exchange protocols which are weaker than AKE but still generally composable with symmetric primitives. Their proof does essentially rely on Random Oracles to model how TLS derives the master key and the application keys. Technically, this allows for example to reduce the security of TLS-DHE to the Computational Diffie-Hellman (CDH) assumption, as opposed to requiring the stronger Decisional Diffie-Hellman and PRF-ODH assumptions.[20] In contrast to our work, they do not cover server-only authentication (and it is not clear if their composability guarantees also hold in the server-only setting).

Also, Brzuska *et al.* [BFS$^+$12] proposed relaxed game-based security notions for key exchange. This approach may serve as an alternative to our ACCE-based approach to circumvent the impossibility of proving the TLS Handshake secure in a key-indistinguishability-based security model.

5.1.3 The TLS Handshake Protocol

In this section we now describe the message flow of TLS in detail. The basic protocol flow in TLS is fixed (and the same) for all ciphersuites, differing only in (i) the content of the messages sent and (ii) the set of messages sent depending on the ciphersuite and the type of authentication (server-only or mutual).

Figure 5.1 depicts the message flow of TLS if the server additionally requests client authentication (left side) and the message flow if only server-authentication is required (right side). Depending on whether client authentication is requested, the number of messages sent in the TLS Handshake Protocol ranges from 9 to 13 messages. The messages can contain cryptographic information as Diffie-Hellman group parameters and public keys as well as constant byte values. In the following,

[20]When using Random Oracles to derive keys, the output of the Random Oracle is indistinguishable from random if only a single input, for example the premaster secret Diffie-Hellman value g^{cs}, where g^c is the client and g^s the server share, is hard to compute for the adversary. When using a PRF the output is only distributed indistinguishable from random if the key g^{cs} is indistinguishable from random in the first place. This allows to use the CDH assumption in contrast to the DDH assumption in the security proof when relying on Random Oracles.

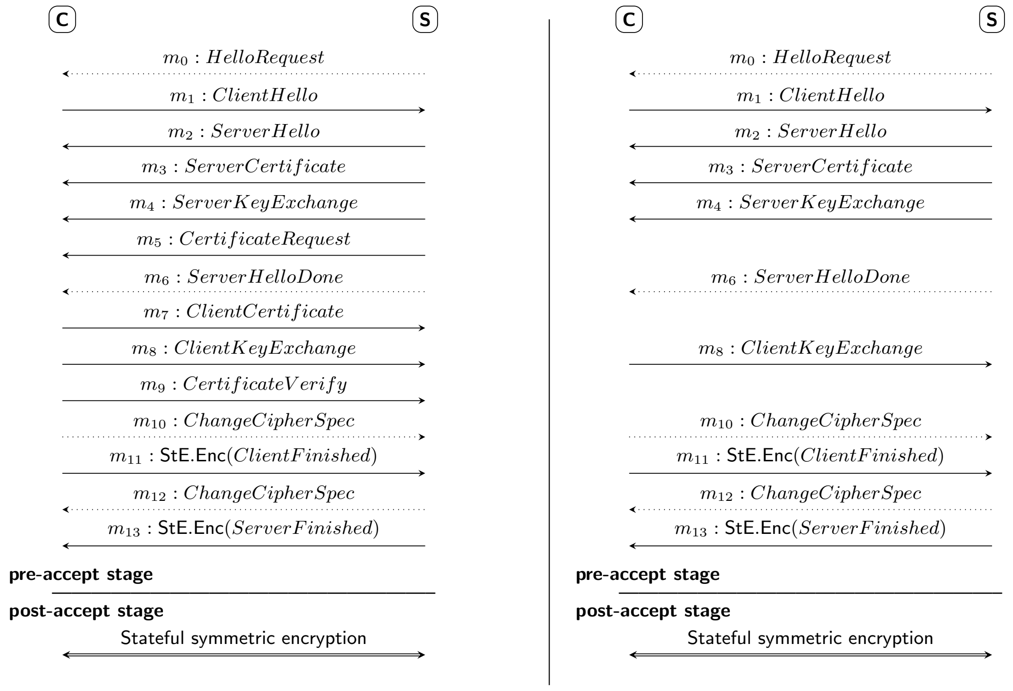
C
S
m_0 : HelloRequest
m_1 : ClientHello
m_2 : ServerHello
m_3 : ServerCertificate
m_4 : ServerKeyExchange
m_5 : CertificateRequest
m_6 : ServerHelloDone
m_7 : ClientCertificate
m_8 : ClientKeyExchange
m_9 : CertificateVerify
m_{10} : ChangeCipherSpec
m_{11} : StE.Enc(ClientFinished)
m_{12} : ChangeCipherSpec
m_{13} : StE.Enc(ServerFinished)
pre-accept stage
post-accept stage
Stateful symmetric encryption
C
S
m_0 : HelloRequest
m_1 : ClientHello
m_2 : ServerHello
m_3 : ServerCertificate
m_4 : ServerKeyExchange
m_6 : ServerHelloDone
m_8 : ClientKeyExchange
m_{10} : ChangeCipherSpec
m_{11} : StE.Enc(ClientFinished)
m_{12} : ChangeCipherSpec
m_{13} : StE.Enc(ServerFinished)
pre-accept stage
post-accept stage
Stateful symmetric encryption

Figure 5.1: TLS handshake with mutual (left) and server-only (right) authentication. Dotted lines mark control messages that contain no cryptographic information. $\mathsf{StE.Enc}(m)$ denotes a stateful symmetric encryption of message m.

we list all messages sent during a TLS protocol run with mutual authentication and explain their explicit content and function.

m_0 `HelloRequest.` This message can be sent by the server to initiate a (new) TLS negotiation. This message does not contain any information (it only contains the value '0') and is not included in the computation of signatures or `Finished` messages. Each message of the handshake protocol is prefixed with the handshake type that allows to distinguish them from other (non-handshake) messages.

m_1
m_2 `ClientHello/ServerHello.` The Hello messages contain a random 32 byte value (nonce) that is used in a later stage to derive the master secret. The nonce sent by the client is denoted as r_C, the one sent by the server as r_S. Note that for the client only 28 bytes are chosen completely at random while 4 bytes are derived from the local time of the client. The Client Hello message also includes a list of ciphersuites supported by the client. The Server Hello message contains the server's choice of the ciphersuite that is then used by both parties. The Hello messages may also contain additional information such as the compression method applied to the messages before sending and possible protocol extensions. Since this additional information is not cryptographically relevant we omit details on this.

m_3 `ServerCertificate.` This message contains the server certificate $cert_S$ (or a certificate chain), which binds a public key to the owner of the certificate (i.e. the server). Besides the server identity, the certificate may contain a subset of the following information, depending on the ciphersuite used:

- public parameters for a Diffie-Hellman based key exchange (a prime number p, a generator g for a prime-order q subgroup of $\mathbb{Z}_p^*$ and a public Diffie-Hellman key (or 'share') g^s (where $s \overset{\$}{\leftarrow} \mathbb{Z}_q$ is the corresponding secret key) in TLS-SDH,
- a public key pk_S (consisting of RSA-modulus N and exponent e with $gcd(e, \varphi(N)) = 1$) used for encrypted key transport in TLS-RSA or
- a public key pk_S of a signature scheme in TLS-DHE.

Note that the client learns the identity of its communication partner not before receiving this message. A certificate may contain additional information, which is out-of-scope for our security analysis. We also omit details on Public-Key Infrastructures (PKIs).

m_4 `ServerKeyExchange.` This message is commonly sent for TLS-DHE and contains public parameters for a DH based key exchange (as defined in the previous message) along with a signature covering both nonces r_C, r_S and the public parameters. It is not sent in TLS-RSA and TLS-SDH ciphersuites, if the server certificate contains valid key material.

m_5 `CertificateRequest.` By sending this message the server requests the client to provide a certificate. The message contains a list of valid certificate types and hash/signature algorithm pairs that the client may use. This message may also contain a list of certificate authorities trusted by the server and *is only sent when client authentication is required.*

m_6 `ServerHelloDone.` Sending this message, the server tells the client to proceed with the next phase of the protocol and check the validity of the server certificate received in m_3.

m_7 `ClientCertificate.` This message contains the client certificate $cert_C$ (or a certificate chain). The certificate contains the client identity and a public key pk_C for a signature scheme. For TLS-SDH ciphersuites, the certificate also contains a fixed Diffie-Hellman public key g^c used for computation of the premaster secret (pms) (where the g is the genenerator specified in the server certificate and $c \overset{\$}{\leftarrow} \mathbb{Z}_q$ is the corresponding secret key). If the client does not have access to a suitable certificate matching the server's requirements, it must send a certificate message containing no certificates.[21] Note that the server learns the identity of its communication partner not before receiving this message. *This message is only sent when client authentication is required.*

m_8 `ClientKeyExchange.` The client always sends this message, the content depends on the negotiated ciphersuite:

- a public DH key g^c (which as before must match the parameters received by the server) in TLS-DHE (and TLS-SDH with server-only authentication),
- a 48-byte (premaster) secret encrypted under the public key of the server in TLS-RSA, or
- an empty string in TLS-SDH, if the client certificate contains a static DH key.

m_9 `CertificateVerify.` Here, the client sends a signature σ_C computed on the concatenation of all messages exchanged so far (that is m_1 to m_8). *This message is only sent when client authentication is required and the client certificate has signing capabilities, i.e. it does not contain static Diffie-Hellman keys.*

m_{10}
m_{12} `ChangeCipherSpec.` This message consists of a single byte of value '1' and indicates, that subsequent messages will be encrypted under the newly established keys. *This message is considered a seperate protocol and not belong a part of the handshake protocol.*

m_{11}
m_{13} `ClientFinished/ServerFinished.` The `Finished` messages contain a stateful encryption of the following information:

$$\mathsf{fin}_C := \mathsf{PRF}(ms, label\|\mathsf{H}(m_1, \ldots, m_{10})), \quad \text{and}$$

$$\mathsf{fin}_S := \mathsf{PRF}(ms, label\|\mathsf{H}(m_1, \ldots, m_{10}, \mathsf{fin}_C{}^{22}, m_{12}))$$

where $label$ ='client finished' for the `ClientFinished` message fin_C and $label$ ='server finished' for the `ServerFinished` message fin_S, and H denotes a collision-resistant hash function. That is message $m_{11} = \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Client}}, \mathsf{len}, H, \mathsf{fin}_C, st_e)$ and message $m_{13} = \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Server}}, \mathsf{len}, H, \mathsf{fin}_S, st_e)$, where $k_{\mathsf{enc}}^{\mathsf{Client}}$, $k_{\mathsf{enc}}^{\mathsf{Server}}$ and the master secret (ms) are defined and computed as below.

[21] In that case the server may then decide to continue without client authentication or to abort.

[22] Note, that m_{11} contains an **encryption** of fin_C, and that fin_S is computed over plaintext handshake messages only.

	mutual TLS-RSA	server-only TLS-RSA	mutual TLS-SDH	server-only TLS-SDH	server-only TLS-DHE
`ClientHello`	r_C	r_C	r_C	r_C	r_C
`ServerHello`	r_S	r_S	r_S	r_S	r_S
`ServerCertificate`	$cert_S$	$cert_S$	$cert_S, g^s$	$cert_S, g^s$	$cert_S$
`ServerKeyExchange`	-[23]	-	-	-	g^s
`ClientCertificate`	$cert_C$	-	$cert_C, g^c$	-	-
`ClientKeyExchange`	$\mathsf{PKE.Enc}(pms)$	$\mathsf{PKE.Enc}(pms)$	-	g^c	g^c

Figure 5.2: Ciphersuite-dependent TLS messages

In the following we will explain, how intermediate keys and application keys are derived from the messages exchanged between the two parties.

Computing the premaster secret. The premaster secret pms is either chosen by the client (in TLS-RSA) or computed as g^{cs} for DH-based ciphersuites, where g is a generator for some prime-order subgroup chosen by the server, g^s is the public DH key of the server and g^c the public DH key of the client.

Computing the master secret. The master secret ms is computed by applying the PRF of TLS, keyed with the premaster secret pms, to a fixed label $label_1$ and the nonces exchanged between the parties r_C, r_S.

$$ms := \mathsf{PRF}(pms, label_1 || r_C || r_S)$$

The master secret will then be used to derive application keys and to compute the `Finished` messages.

Computing the application keys. The four application keys (encryption and MAC keys for each direction) are also computed using the PRF of TLS, where the inputs are now the master secret ms, another fixed label $label_2$ and (again) the random nonces r_C, r_S.

$$K_{\mathsf{enc}}^{C\to S} || K_{\mathsf{enc}}^{S\to C} || K_{\mathsf{mac}}^{C\to S} || K_{\mathsf{mac}}^{S\to C} := \mathsf{PRF}(ms, label_2 || r_C || r_S)$$

We also define

$$k_{\mathsf{enc}}^{\mathsf{Client}} = k_{\mathsf{dec}}^{\mathsf{Server}} := K_{\mathsf{enc}}^{C\to S} || K_{\mathsf{mac}}^{C\to S}, \quad \text{and}$$

$$k_{\mathsf{enc}}^{\mathsf{Server}} = k_{\mathsf{dec}}^{\mathsf{Client}} := K_{\mathsf{enc}}^{S\to C} || K_{\mathsf{mac}}^{S\to C}.$$

We refer to these application keys as **encryption keys**. In the following we use k_S short for the encryption key(s) $k_{\mathsf{enc}}^{\mathsf{Server}} = k_{\mathsf{dec}}^{\mathsf{Client}}$ and k_C short for the encryption key(s) $k_{\mathsf{enc}}^{\mathsf{Client}} = k_{\mathsf{dec}}^{\mathsf{Server}}$. These encryption keys are then used to key a stateful length-hiding authenticated encryption scheme as defined in Definition 2.11.

[23]'-' means that the message is not sent (or empty) in this setting

Figure 5.3 exemplary depicts the full message flow for the TLS ciphersuite with mutual authentication and ephemeral Diffie-Hellman key exchange in detail.

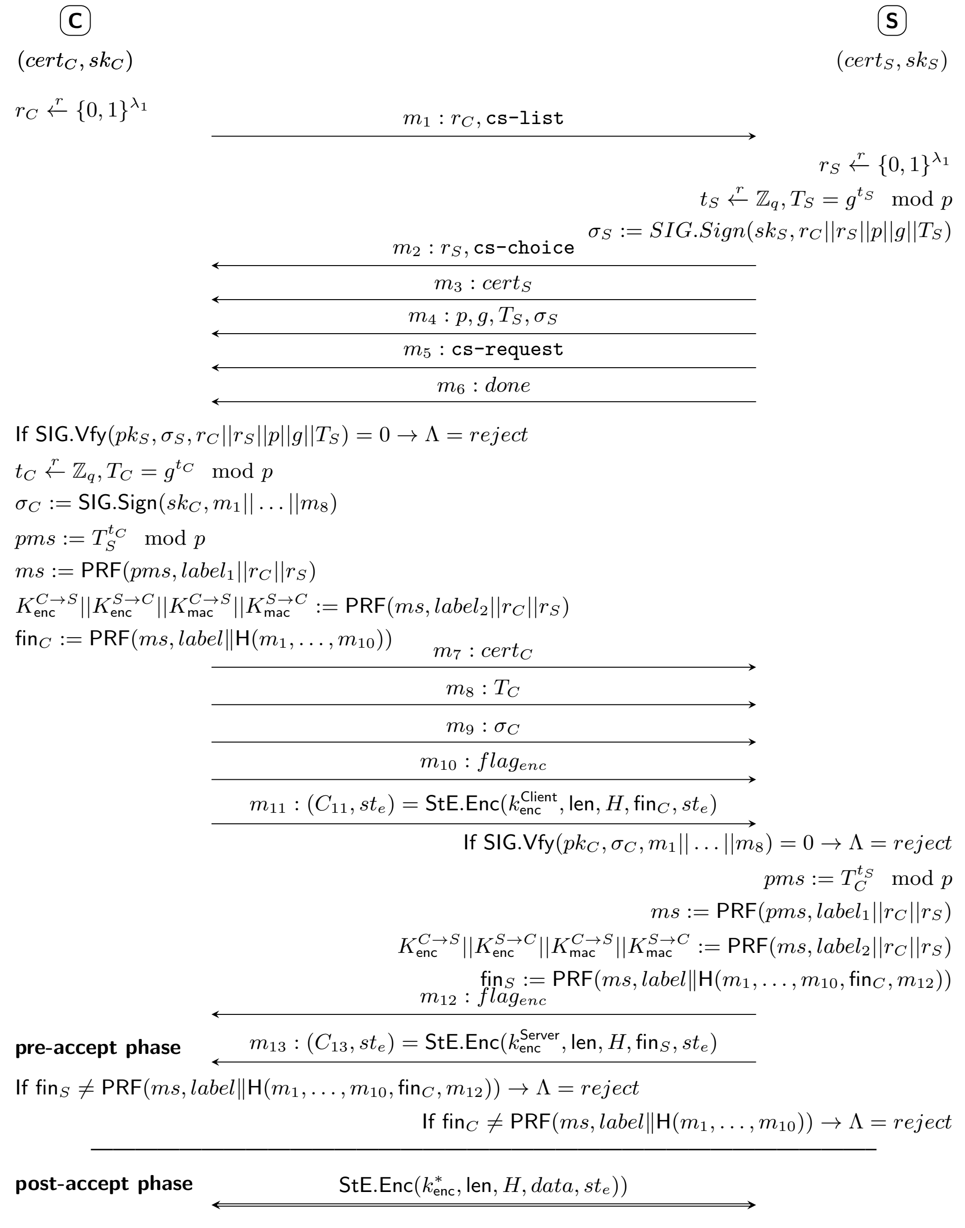
C
S
$(cert_C, sk_C)$
$(cert_S, sk_S)$
$r_C \xleftarrow{r} \{0,1\}^{\lambda_1}$
$m_1 : r_C,$ cs-list
$r_S \xleftarrow{r} \{0,1\}^{\lambda_1}$
$t_S \xleftarrow{r} \mathbb{Z}_q, T_S = g^{t_S} \mod p$
$\sigma_S := SIG.Sign(sk_S, r_C||r_S||p||g||T_S)$
$m_2 : r_S,$ cs-choice
$m_3 : cert_S$
$m_4 : p, g, T_S, \sigma_S$
$m_5 :$ cs-request
$m_6 : done$
If $\mathsf{SIG.Vfy}(pk_S, \sigma_S, r_C||r_S||p||g||T_S) = 0 \to \Lambda = reject$
$t_C \xleftarrow{r} \mathbb{Z}_q, T_C = g^{t_C} \mod p$
$\sigma_C := \mathsf{SIG.Sign}(sk_C, m_1||\dots||m_8)$
$pms := T_S^{t_C} \mod p$
$ms := \mathsf{PRF}(pms, label_1||r_C||r_S)$
$K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C} := \mathsf{PRF}(ms, label_2||r_C||r_S)$
$\mathsf{fin}_C := \mathsf{PRF}(ms, label\|\mathsf{H}(m_1, \dots, m_{10}))$
$m_7 : cert_C$
$m_8 : T_C$
$m_9 : \sigma_C$
$m_{10} : flag_{enc}$
$m_{11} : (C_{11}, st_e) = \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Client}}, \mathsf{len}, H, \mathsf{fin}_C, st_e)$
If $\mathsf{SIG.Vfy}(pk_C, \sigma_C, m_1||\dots||m_8) = 0 \to \Lambda = reject$
$pms := T_C^{t_S} \mod p$
$ms := \mathsf{PRF}(pms, label_1||r_C||r_S)$
$K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C} := \mathsf{PRF}(ms, label_2||r_C||r_S)$
$\mathsf{fin}_S := \mathsf{PRF}(ms, label\|\mathsf{H}(m_1, \dots, m_{10}, \mathsf{fin}_C, m_{12}))$
$m_{12} : flag_{enc}$
pre-accept phase
$m_{13} : (C_{13}, st_e) = \mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Server}}, \mathsf{len}, H, \mathsf{fin}_S, st_e)$
If $\mathsf{fin}_S \neq \mathsf{PRF}(ms, label\|\mathsf{H}(m_1, \dots, m_{10}, \mathsf{fin}_C, m_{12})) \to \Lambda = reject$
If $\mathsf{fin}_C \neq \mathsf{PRF}(ms, label\|\mathsf{H}(m_1, \dots, m_{10})) \to \Lambda = reject$
post-accept phase
$\mathsf{StE.Enc}(k_{\mathsf{enc}}^*, \mathsf{len}, H, data, st_e))$

Figure 5.3: TLS handshake for TLS-DHE with mutual authentication

Attack Focus. We do neither consider abbreviated TLS Handshakes[24], nor side-channel attacks (such as the Bleichenbacher attack against PKCS #1.5 [Ble98] or the BEAST (Browser Exploit Against SSL/TLS) attack[25] discussed in [Bar04, Bar06]). Moreover, we do not consider attacks based on side-channels such as error messages, or implementation issues (e.g. the cross-protocol attack by Schneier

[24]Note that the server can always enforce a full TLS Handshake
[25]This vulnerability was fixed in TLS 1.1

and Wagner [WS96]) and for simplicity also assume that TLS compression is not used, excluding attacks like CRIME [RD12].

5.2 Truncated TLS-DHE with Mutual Authentication is a Secure AKE Protocol

In this section we prove the security of a modified version of the TLS Handshake protocol. As previously discussed in Section 5.1.2, it is impossible to prove the full TLS Handshake protocol secure in any security model based on key-indistinguishability, like the model from Section 3.2.1, because the encryption and MAC of the `Finished` messages provide a 'check value', that can be exploited by an adversary to determine the bit b chosen by the Test query.

Therefore we consider a 'truncated TLS' protocol as in [MSW08, MSW10]. In this truncated version, we assume that the `Finished` messages are sent in clear, that is, neither encrypted nor authenticated by a MAC. More precisely, we modify the TLS protocol depicted in Figure 5.1 such that

- message m_{11} contains only fin_C (instead of $\mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Client}}, \mathsf{len}, H, \mathsf{fin}_C, st_e)$), and
- message m_{13} contains only fin_S (instead of $\mathsf{StE.Enc}(k_{\mathsf{enc}}^{\mathsf{Server}}, \mathsf{len}, H, \mathsf{fin}_S, st_e)$).

This simple modification allows to prove security in the key-indisinguishability-based security model from Section 3.2.1.

Introducing the PRF-ODH Assumption

At first sight, a more direct approach to give a security proof of TLS-DHE might be to reduce the security of the protocol (at least of the key exchange part) to the DDH problem. However, this is not possible in a security model that allows the adversary to corrupt parties, i.e. learn long-term secret keys. The problem that denies such a proof occurs when embedding the DDH challenge. An adversary can always test the simulator's knowledge of the secret exponents corresponding to the (public) Diffie-Hellman shares sent during the key exchange. To this end the adversary proceeds as follows. Note that the simulator has to embed (part of) the DDH challenge in the server's key exchange message without (cryptographically) knowing the identity of the client. So when the adversary intercepts a `ServerKeyExchange` message, it always chooses a fresh random exponent from the corresponding DH group, corrupts the client to learn the client's signing key and responds to the server with the modified `ClientKeyExchange` message. The server cannot compute the resulting session key, because it knows neither the secret exponent of the DH share it has sent (as it was taken from the

DDH challenge) nor the secret exponent from the client's share (as it was chosen by the adversary).

It is important to note, that we may have to deal with a successfull adversary, which is able to break some primitive later in the protocol execution[26]. Thus, to be able to use such an adversary to break the security of this primitive, we first have to efficiently simulate all oracles to the adversary, which we cannot do in the above mentioned case. We cannot compute the DH key and thus cannot compute the `ServerFinished` message.

Theorem 5.1. *Let μ be the output length of* PRF *and let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the signature scheme used to sign messages in TLS is $(t, \epsilon_{\mathsf{sig}}, q_{\mathsf{sig}})$-secure, the hash function* H *is $(t, \epsilon_{\mathsf{H}})$-collision resistant, the PRF-ODH-problem is $(t, \epsilon_{\mathsf{prfodh}})$-hard with respect to G and* PRF*, the DDH-problem is $(t, \epsilon_{\mathsf{ddh}})$-hard in the group G used to compute the TLS premaster secret, and $q_{\mathsf{prf}} \leq \eta$ and $q_{\mathsf{sig}} \geq \eta$.*

Then for any adversary that $(t', \epsilon_{\mathsf{ttls}})$-breaks the truncated ephemeral Diffie-Hellman TLS Handshake protocol in the sense of Definition 3.4 with $t \approx t'$ holds that

$$\epsilon_{\mathsf{ttls}} \leq 4 \cdot \eta\phi \left(\frac{\eta\phi}{2^{\lambda}} + \phi \cdot \epsilon_{\mathsf{sig}} + \frac{5}{4} \cdot \epsilon_{\mathsf{ddh}} + \frac{5}{2} \cdot \epsilon_{\mathsf{prf}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \frac{1}{2^{\mu}} \right) \right).$$

We consider three types of adversaries:

1. Adversaries that succeed in making an oracle accept maliciously, such that the first oracle that does so is a Client-oracle (i.e. an oracle with $\rho = \mathsf{Client}$). We call such an adversary a Client-adversary.

2. Adversaries that succeed in making an oracle accept maliciously, such that the first oracle that does so is a Server-oracle (i.e. an oracle with $\rho = \mathsf{Server}$). We call such an adversary a Server-adversary.

3. Adversaries that do not succeed in making any oracle accept maliciously, but which answer the Test-challenge. We call such an adversary a Test-adversary.

We prove Theorem 5.1 by proving three lemmas. Lemma 5.1 bounds the probability $\epsilon_{\mathsf{client}}$ that a Client-adversary succeeds, Lemma 5.2 bounds the probability $\epsilon_{\mathsf{server}}$ that a Server-adversary succeeds, and Lemma 5.3 bounds the success probability ϵ_{enc} of a Test-adversary. Then we have

$$\epsilon_{\mathsf{ttls}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} + \epsilon_{\mathsf{enc}}.$$

[26]Take as example an adversary that on input of a valid `ServerFinished` message can output a different, but colliding`ServerFinished` message, that will be accepted by the client.

5.2.1 Authentication

Lemma 5.1. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s with $\rho =$ Client that accepts maliciously is at most*

$$\epsilon_{\mathsf{client}} \leq \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^\mu} \right) \right)$$

where all quantities are defined as stated in Theorem 5.1.

PROOF. The proof proceeds in a *sequence of games*, following [BR06, Sho04]. The first game is the real security experiment. We then describe several intermediate games that modify the original game step-by-step, and argue that our complexity assumptions imply that each game is computationally indistinguishable from the previous one. We end up in the final game, where no adversary can break the security of the protocol.

Let $\mathsf{break}_\delta^{(1)}$ be the event that occurs when the first oracle accepts maliciously in the sense of Definition 3.4 with $\rho =$ Client in Game δ.

Game 0. This game equals the AKE security experiment described in Section 3.2.1. Thus, for some $\epsilon_{\mathsf{client}}$ we have

$$\Pr[\mathsf{break}_0^{(1)}] = \epsilon_{\mathsf{client}}.$$

Game 1. In this game we add an abort rule. The challenger aborts, if there exists any oracle π_i^s that chooses a random nonce r_C or r_S which is not unique. More precisely, the game is aborted if the adversary ever makes a first Send query to an oracle π_i^s, and the oracle replies with random nonce r_C or r_S such that there exists some other oracle $\pi_{i'}^{s'}$ which has previously sampled the same nonce.

In total less than $\eta\phi$ nonces r_C and r_S are sampled, each uniformly random from $\{0,1\}^\lambda$. Thus, the probability that a collision occurs is bounded by $(\eta\phi)^2 2^{-\lambda}$, which implies

$$\Pr[\mathsf{break}_0^{(1)}] \leq \Pr[\mathsf{break}_1^{(1)}] + \frac{(\eta\phi)^2}{2^\lambda}.$$

Note that now each oracle has a unique nonce r_C or r_S, which is included in the signatures. We will use this to ensure that each oracle that accepts with non-corrupted partner has a *unique* partner oracle.

Game 2. We try to guess which client oracle will be the first oracle to accept maliciously. If our guess is wrong, i.e. if there is another (Client or Server) oracle that accepts before, then we abort the game.

Technically, this game is identical to Game 1, except for the following. The challenger guesses two random indices $(i^*, s^*) \xleftarrow{\$} [\phi] \times [\eta]$. If there exists an oracle π_i^s that 'accepts' maliciously before, and $(i, s) \neq (i^*, s^*)$ and π_i^s has $\rho \neq \mathsf{Client}$, then the challenger aborts the game. With probability $1/(\eta\phi)$ we have $(i, s) = (i^*, s^*)$, and thus

$$\Pr[\mathsf{break}_1^{(1)}] = \eta\phi \cdot \Pr[\mathsf{break}_2^{(1)}].$$

Note that in this game the adversary can only break the security of the protocol, if oracle $\pi_{i^*}^{s^*}$ is the first oracle that 'accepts' maliciously and has $\rho = \mathsf{Client}$, as otherwise the game is aborted.

Game 3. Again the challenger proceeds as before, but we add an abort rule. We want to make sure that $\pi_{i^*}^{s^*}$ receives as input exactly the Diffie-Hellman value T_S that was selected by some other uncorrupted oracle that received the nonce r_C chosen by $\pi_{i^*}^{s^*}$ as first input (note that there may be several such oracles, since the adversary may send copies of r_C to many oracles).

Technically, we abort and raise event $\mathsf{abort}_{\mathsf{sig}}$, if oracle $\pi_{i^*}^{s^*}$ ever receives as input a message $m_3 = cert_S$ indicating intended partner $\Pi = j$ and message $m_4 = (p, g, T_S, \sigma_S)$ such that σ_S is a valid signature over $r_C||r_S||p||g||T_S$ that verifies under $cert_S$, but there exists no oracle π_j^t which has previously output σ_S. Clearly we have

$$Pr[\mathsf{break}_2^{(1)}] \leq \Pr[\mathsf{break}_3^{(1)}] + \Pr[\mathsf{abort}_{\mathsf{sig}}].$$

Note that the experiment is aborted, if $\pi_{i^*}^{s^*}$ does not accept *maliciously*, due to Game 2. This means that party P_j must be τ_j-corrupted with $\tau_j = \infty$ (i.e. not corrupted) when $\pi_{i^*}^{s^*}$ accepts (as otherwise $\pi_{i^*}^{s^*}$ does not accept *maliciously*). To show that $\Pr[\mathsf{abort}_{\mathsf{sig}}] \leq \phi \cdot \epsilon_{\mathsf{sig}}$, we construct a signature forger as follows. The forger receives as input a public key pk^* and simulates the challenger for $\mathcal{A}$. It guesses an index $\phi \xleftarrow{\$} [\phi]$, sets $pk_\phi = pk^*$, and generates all long-term public/secret keys as before. Then it proceeds as the challenger in Game 3, except that it uses its chosen-message oracle to generate a signature under pk_ϕ when necessary.

If $\phi = j$, which happens with probability $1/\phi$, then the forger can use the signature received by $\pi_{i^*}^{s^*}$ to break the EUF-CMA security of the signature scheme with success probability ϵ_{sig}, so $\Pr[\mathsf{abort}_{\mathsf{sig}}]/\phi \leq \epsilon_{\mathsf{sig}}$. Therefore if $\Pr[\mathsf{abort}_{\mathsf{sig}}]$ is not negligible, then ϵ_{sig} is not negligible as well and we have

$$Pr[\mathsf{break}_2^{(1)}] \leq \Pr[\mathsf{break}_3^{(1)}] + \phi \cdot \epsilon_{\mathsf{sig}}.$$

Note that in Game 3 oracle $\pi_{i^*}^{s^*}$ receives as input a Diffie-Hellman value T_S such that T_S was chosen by another oracle, but not by the adversary. Note also that there may be multiple oracles that issued a signature σ_S containing r_C, since the adversary may have sent several copies of r_C to several oracles.

Game 4. In this game we want to make sure that we know *which* oracle π_j^t will issue the signature σ_S that $\pi_{i^*}^{s^*}$ receives. Note that this signature includes the random nonce r_S, which is unique due to Game 1. Therefore the challanger in this game proceeds as before, but additionally guesses two indices $(j^*, t^*) \xleftarrow{\$} [\phi] \times [\eta]$. It aborts, if the adversary does *not* make a Send query containing r_C to $\pi_{j^*}^{t^*}$ or if $\pi_{j^*}^{t^*}$ responds with messages containing σ_S and σ_S is not forwarded to $\pi_{i^*}^{s^*}$.

We know that there must exists at least one oracle that outputs σ_S such that σ_S is forwarded to $\pi_{i^*}^{s^*}$, due to Game 3. Thus we have

$$\Pr[\mathsf{break}_3^{(1)}] \leq \eta\phi \cdot \Pr[\mathsf{break}_4^{(1)}].$$

Note that in this game we know exactly that oracle $\pi_{j^*}^{t^*}$ chooses the Diffie-Hellman share T_S that $\pi_{i^*}^{s^*}$ uses to compute its premaster secret.

Game 5. Recall that $\pi_{i^*}^{s^*}$ computes the master secret as $ms = \mathsf{PRF}(T_S^{t_c}, label_1||r_C||r_S)$, where T_S denotes the Diffie-Hellman share received from $\pi_{j^*}^{t^*}$, and t_c denotes the Diffie-Hellman exponent chosen by $\pi_{i^*}^{s^*}$. In this game we replace the master secret ms computed by $\pi_{i^*}^{s^*}$ with an independent random value $\widetilde{ms}$. Moreover, if $\pi_{j^*}^{t^*}$ receives as input the same Diffie-Hellman share T_C that was sent from $\pi_{i^*}^{s^*}$, then we set the master secret of $\pi_{j^*}^{t^*}$ equal to $\widetilde{ms}$. Otherwise we compute the master secret as specified in the protocol. We claim that

$$Pr[\mathsf{break}_4^{(1)}] \leq \Pr[\mathsf{break}_5^{(1)}] + \epsilon_{\mathsf{prfodh}}.$$

Suppose there exists an adversary $\mathcal{A}$ that distinguishes Game 5 from Game 4. We show that this implies an adversary $\mathcal{B}$ that solves the PRF-ODH problem.

Adversary $\mathcal{B}$ outputs $(label_1||r_C||r_S)$ to its oracle and receives in response (g, g^u, g^v, R), where either $R = \mathsf{PRF}(g^{uv}, label_1||r_C||r_S)$ or $R \xleftarrow{\$} \{0,1\}^\mu$. It runs $\mathcal{A}$ by implementing the challenger for $\mathcal{A}$, and embeds (g^u, g^v) as follows. Instead of letting $\pi_{i^*}^{s^*}$ choose $T_C = g^{t_C}$ for random $t_C \xleftarrow{\$} \mathbb{Z}_q$, $\mathcal{B}$ defines $T_C := g^u$. Similarly, the Diffie-Hellman share T_S of $\pi_{j^*}^{t^*}$ is defined as $T_S := g^v$. Finally, the master secret of $\pi_{i^*}^{s^*}$ is set equal to R.

Note that $\pi_{i^*}^{s^*}$ computes the master secret after receiving T_S from $\pi_{j^*}^{t^*}$, and then it sends $m_8 = T_C$. If the adversary decides to forward m_8 to $\pi_{j^*}^{t^*}$, then the master secret of $\pi_{j^*}^{t^*}$ is set equal to R. If $\pi_{j^*}^{t^*}$ receives $T_{C'} \neq T_C$, then $\mathcal{B}$ queries its oracle to compute $ms' = \mathsf{PRF}(T_{C'}^v, label_1||r_C||r_S)$, and sets the master secret of $\pi_{j^*}^{t^*}$ equal to ms'.

Note that in any case algorithm $\mathcal{B}$ 'knows' the master secret of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, and thus is able to compute all further protocol messages (in particular the finished messages fin_C and fin_S) and answer a potential Reveal query to $\pi_{j^*}^{t^*}$ as required (note that there is no Reveal query to $\pi_{i^*}^{s^*}$, as otherwise the experiment is aborted, due to Game 2). If $R = \mathsf{PRF}(g^{uv}, label_1||r_C||r_S)$, then the view of $\mathcal{A}$ is identical to Game 4, while if $R \xleftarrow{\$} \{0,1\}^\mu$ then it is identical to Game 5, which yields the above claim.

Game 6. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ with a PRF challenge. If $\pi_{j^*}^{t^*}$ uses the same master secret $\widetilde{ms}$ as $\pi_{i^*}^{s^*}$ (cf. Game 5), then the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{j^*}^{t^*}$ is replaced as well. Of course the same random function is used for both oracles sharing the same $\widetilde{ms}$. In particular, this function is used to compute the `Finished` messages by both partner oracles.

Distinguishing Game 6 from Game 5 implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\Pr[\mathsf{break}_5^{(1)}] \leq \Pr[\mathsf{break}_6^{(1)}] + \epsilon_{\mathsf{prf}}$$

Game 7. In Game 6 we have replaced the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ with a random function. Thus, the `ServerFinished` message expected by $\pi_{i^*}^{s^*}$ is

$$\mathsf{fin}_S^* = F_{\widetilde{ms}}(label_4||\mathsf{H}(m_1||\cdots||m_{12})),$$

where $m_1||\cdots||m_{12}$ denotes the transcript of all messages sent and received by $\pi_{i^*}^{s^*}$. In the next game we would like to argue, that the adversary is not able to predict fin_S^*, unless there is an oracle $\pi_{j^*}^{t^*}$ having a matching conversation to $\pi_{i^*}^{s^*}$, because $F_{\widetilde{ms}}$ is random.

Before we can do so, we need to make sure that oracle $\pi_{j^*}^{t^*}$ (the only other oracle potentially having access to $F_{\widetilde{ms}}$, due to Game 6) never evaluates $F_{\widetilde{ms}}$ on any input $label_4||\mathsf{H}(m')$ with

$$m' \neq m_1||\cdots||m_{12} \quad \text{and} \quad \mathsf{H}(m') = \mathsf{H}(m_1||\cdots||m_{12}). \tag{5.1}$$

Therefore we add another abort condition. We abort the game, if oracle $\pi_{j^*}^{t^*}$ ever evaluates the random function $F_{\widetilde{ms}}$ on an input m' such that (5.1) holds. Since (5.1) implies that a collision for H is found, we have

$$\Pr[\mathsf{break}_6^{(1)}] \leq \Pr[\mathsf{break}_7^{(1)}] + \epsilon_{\mathsf{H}}$$

Game 8. Finally we use that the *unique* (due to Game 7) hash of the full transcript of all messages sent and received is used to compute the `Finished` messages, and that `Finished` messages are computed by evaluating a truly random function that is only accessible to $\pi_{i^*}^{s^*}$ and (possibly) $\pi_{j^*}^{t^*}$ due to Game 6. This allows to show that any adversary has probability at most $\frac{1}{2^\mu}$ of making oracle $\pi_{i^*}^{s^*}$ accept without having a matching conversation to $\pi_{j^*}^{t^*}$.

The `Finished` messages are computed by evaluating a truly random function $F_{\widetilde{ms}}$. This function is only accessible to oracles sharing $\widetilde{ms}$, and evaluated on a unique hash value derived from the full transcript containing all previous messages. Thus, if there is no oracle having a matching conversation to $\pi_{i^*}^{s^*}$, the adversary receives no information about $F_{\widetilde{ms}}(label_4||\mathsf{H}(m_1||\cdots||m_{12}))$. Therefore we have $\Pr[\mathsf{break}_8^{(1)}] = 0$ and

$$\Pr[\mathsf{break}_7^{(1)}] \leq \Pr[\mathsf{break}_8^{(1)}] + \frac{1}{2^\mu} = \frac{1}{2^\mu}.$$

Collecting probabilities from Game 0 to Game 8 yields Lemma 5.1. □

Lemma 5.2. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s with $\rho = \mathsf{Server}$ that accepts maliciously is at most*

$$\epsilon_{\mathsf{server}} \leq \eta\phi\left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^\mu}\right)$$

where all quantities are defined as stated in Theorem 5.1.

Proof. Let $\mathsf{break}_\delta^{(2)}$ be the event that occurs when the first oracle accepts maliciously in the sense of Definition 3.4 with $\rho = \mathsf{Server}$ in Game δ.

Game 0. This game equals the AKE security experiment described in Section 3.2.1. Thus, for some $\epsilon_{\mathsf{server}}$ we have

$$\Pr[\mathsf{break}_0^{(2)}] = \epsilon_{\mathsf{server}}.$$

Game 1. In this game we add an abort rule. The challenger aborts, if there exists any oracle π_i^s that chooses a random nonce r_C or r_S which is not unique. With the same arguments as in Game 1 from the proof of Lemma 5.1 we have

$$\Pr[\mathsf{break}_0^{(2)}] \leq \Pr[\mathsf{break}_1^{(2)}] + \frac{(\eta\phi)^2}{2^\lambda}.$$

Game 2. This game is identical, except for the following. The challenger guesses two random indices $(i^*, s^*) \xleftarrow{\$} [\phi] \times [\eta]$. If there exists an oracle π_i^s that 'accepts' maliciously, and $(i, s) \neq (i^*, s^*)$ and π_i^s has $\rho \neq \mathsf{Server}$, then the challenger aborts the game. Note that if the first oracle π_i^s that 'accepts' maliciously has $\rho = \mathsf{Server}$, then with probability $1/(\eta\phi)$ we have $(i, s) = (i^*, s^*)$, and thus

$$\Pr[\mathsf{break}_1^{(2)}] = \eta\phi \cdot \Pr[\mathsf{break}_2^{(2)}].$$

Note that in this game the adversary can only break the security of the protocol, if oracle $\pi_{i^*}^{s^*}$ is the first oracle that 'accepts' maliciously and has $\rho = \mathsf{Server}$, as otherwise the game is aborted.

Game 3. The challenger proceeds as before, but we add an abort rule. We want to make sure that $\pi_{i^*}^{s^*}$ receives as input exactly the Diffie-Hellman value $m_8 = T_C$ that was selected by some other uncorrupted oracle.

Technically, we abort and raise event $\mathsf{abort}_{\mathsf{sig}}$, if oracle $\pi_{i^*}^{s^*}$ ever receives as input a message $m_7 = cert_C$ indicating intended partner $\Pi = j$ and message $m_9 = \sigma_C$ such that σ_C is valid, i.e. $\mathsf{SIG.Vfy}(pk_C, \sigma_C, m_1||\ldots||m_8) = 1$, but there exists no oracle π_j^t which has previously output σ_C. Clearly we have

$$Pr[\mathsf{break}_2^{(2)}] \leq \Pr[\mathsf{break}_3^{(2)}] + \Pr[\mathsf{abort}_{\mathsf{sig}}].$$

Note that the experiment is aborted, if $\pi_{i^*}^{s^*}$ does not accept *maliciously*, due to Game 2. This means that party P_j must be τ_j-corrupted with $\tau_j = \infty$ (i.e. not corrupted) when $\pi_{i^*}^{s^*}$ accepts. To show that $\Pr[\mathsf{abort}_{\mathsf{sig}}] \leq \phi \cdot \epsilon_{\mathsf{sig}}$, we construct a signature forger as follows. The forger receives as input a public key pk^* and simulates the challenger for $\mathcal{A}$. It guesses an index $\phi \xleftarrow{\$} [\phi]$, sets $pk_\phi = pk^*$, and generates all long-term public/secret keys as before. Then it proceeds as the challenger in Game 3, except that it uses its chosen-message oracle to generate a signature under pk_ϕ when necessary.

If $\phi = j$, which happens with probability $1/\phi$, then the forger can use the signature received by $\pi_{i^*}^{s^*}$ to break the EUF-CMA security of the signature scheme with success probability ϵ_{sig}, so $\Pr[\mathsf{abort}_{\mathsf{sig}}]/\phi \leq \epsilon_{\mathsf{sig}}$. Therefore if $\Pr[\mathsf{abort}_{\mathsf{sig}}]$ is not negligible, then ϵ_{sig} is not negligible as well and we have

$$Pr[\mathsf{break}_2^{(2)}] \leq \Pr[\mathsf{break}_3^{(2)}] + \phi \cdot \epsilon_{\mathsf{sig}}.$$

Note that in Game 3 oracle $\pi_{i^*}^{s^*}$ receives as input a Diffie-Hellman value T_C such that T_C was chosen by another oracle, but not by the adversary. Note also that this oracle is unique, since the signature includes the client nonce r_C, which is unique due to Game 1. From now on we denote this unique oracle with $\pi_{j^*}^{t^*}$.

Note also that $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ share a premaster secret $pms = T_C^{t_S} = T_S^{t_C}$, where $T_C = g^{t_C}$ and $T_S = g^{t_S}$ for random exponents t_S and t_C chosen by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively.

Game 4. In this game, we replace the premaster secret $pms = g^{t_C t_S}$ shared by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random value g^r, $r \overset{\$}{\leftarrow} \mathbb{Z}_q$. The fact that the challenger has full control over the Diffie-Hellman shares T_C and T_S exchanged between $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, due to the modifications introduced in the previous games, provides us with the leverage to prove indistinguishability under the Decisional Diffie-Hellman assumption.

Technically, the challenger in Game 4 proceeds as before, but when $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ compute the premaster secret as $pms = g^{t_C t_S}$, the challenger replaces this value with a uniformly random value $\widetilde{pms} = g^r, r \overset{\$}{\leftarrow} \mathbb{Z}_p^*$, which is in the following used by both partner oracles. Suppose there exists an algorithm distinguishing Game 4 from Game 3. Then we can construct an algorithm $\mathcal{B}$ solving the DDH problem as follows. Algorithm $\mathcal{B}$ receives as input a DDH challenge (g, g^u, g^v, g^w). The challenger defines $T_C := g^u$ and $T_S := g^v$ for the Diffie-Hellman shares chosen by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively. Instead of computing the Diffie-Hellman key as in Game 3, it sets $pms = g^w$ both for the 'client' and the 'server' oracle. Now if $w = uv$, then this game proceeds exactly like Game 3, while if w is random than this game proceeds exactly like Game 4. The DDH assumption therefore implies that

$$\Pr[\mathsf{break}_3^{(2)}] \leq \Pr[\mathsf{break}_4^{(2)}] + \epsilon_{\mathsf{ddh}}.$$

Note that in Game 4 the premaster secret of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is uniformly random, and independent of T_C and T_S. This will provide us with the leverage to replace the function $\mathsf{PRF}(\widetilde{pms}, \cdot)$ with a truly random function in the next game.

Game 5. In Game 5 we make use of the fact that the premaster secret $\widetilde{pms}$ of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is chosen uniformly random, and independent of T_C and T_S. We thus replace the value $ms = \mathsf{PRF}(\widetilde{pms}, label_1 || r_C || r_S)$ with a random value $\widetilde{ms}$.

Distinguishing Game 5 from Game 4 implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\Pr[\mathsf{break}_4^{(2)}] \leq \Pr[\mathsf{break}_5^{(2)}] + \epsilon_{\mathsf{prf}}.$$

Game 6. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random function. Of course the same random function is used for both oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$. In particular, this function is used to compute the `Finished` messages by both partner oracles.

Distinguishing Game 6 from Game 5 again implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\Pr[\mathsf{break}_5^{(2)}] \leq \Pr[\mathsf{break}_6^{(2)}] + \epsilon_{\mathsf{prf}}.$$

Game 7. In Game 6 we have replaced the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ with a random function. Thus, the `ClientFinished` message expected by $\pi_{i^*}^{s^*}$ is

$$\mathsf{fin}_C^* = F_{\widetilde{ms}}(label_3 || \mathsf{H}(m_1 || \cdots || m_{10})),$$

where $m_1 || \cdots || m_{10}$ denotes the transcript of all messages sent and received by $\pi_{i^*}^{s^*}$. Again we would like to argue that the adversary is not able to predict fin_C^*, unless there is an oracle $\pi_{j^*}^{t^*}$ having a matching conversation to $\pi_{i^*}^{s^*}$, because $F_{\widetilde{ms}}$ is random.

Before we can do so, we need to make sure that oracle $\pi_{j^*}^{t^*}$ (the only other oracle potentially having access to $F_{\widetilde{ms}}$, due to Game 6) never evaluates $F_{\widetilde{ms}}$ on any input $label_3 || \mathsf{H}(m')$ with

$$m' \neq m_1 || \cdots || m_{10} \quad \text{and} \quad \mathsf{H}(m') = \mathsf{H}(m_1 || \cdots || m_{10}). \tag{5.2}$$

Therefore we add another abort condition. We abort the game, if oracle $\pi_{j^*}^{t^*}$ ever evaluates the random function $F_{\widetilde{ms}}$ on an input m' such that (5.2) holds. Since (5.2) implies that a collision for H is found, we have

$$\Pr[\mathsf{break}_6^{(2)}] \leq \Pr[\mathsf{break}_7^{(2)}] + \epsilon_{\mathsf{H}}$$

Game 8. Finally we use that the unique hash of the full transcript of all messages sent and received by $\pi_{i^*}^{s^*}$ is used to compute the `Finished` messages, and that `Finished` messages are computed by evaluating a truly random function that is only accessible to $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ due to Game 7. This allows to show that any adversary has probability at most $\frac{1}{2^\mu}$ of making oracle $\pi_{i^*}^{s^*}$ accept without having a matching conversation to $\pi_{j^*}^{t^*}$.

The `Finished` messages are computed by evaluating a truly random function $F_{\widetilde{ms}}$, which is only accessible to oracles sharing $\widetilde{ms}$, and the full transcript containing all previous messages is used to compute the `Finished` messages. If there is no oracle having a matching conversation to $\pi_{i^*}^{s^*}$, the adversary receives no information about $F_{\widetilde{ms}}(label_3 || m_1 || \cdots || m_{10})$. Thus we have

$$\Pr[\mathsf{break}_7^{(2)}] \leq \Pr[\mathsf{break}_8^{(2)}] + \frac{1}{2^\mu} = \frac{1}{2^\mu}.$$

Collecting probabilities from Game 0 to Game 8 yields Lemma 5.2. □

5.2.2 Indistinguishability of Keys

Lemma 5.3. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the* Test*-challenge correctly is at most $1/2 + \epsilon_{\mathsf{enc}}$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} + \eta\phi \cdot (\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}}) .$$

where $\epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}$ is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 3.4 (cf. Lemmas 5.1 and 5.2) and all other quantities are defined as stated in Theorem 5.1.

Proof. Assume without loss of generality that the $\mathcal{A}$ always asks a Test query such that all conditions in Property 2 of Definition 3.4 are satisfied. Let $\mathsf{break}_\delta^{(3)}$ denote the event that $b' = b$ in Game δ, where b is the random bit sampled by the Test query, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta^{(3)}] - 1/2$ denote the *advantage* of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This game equals the AKE security experiment described in Section 3.2.1. For some ϵ_{enc} we have

$$\Pr[\mathsf{break}_0^{(3)}] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = \frac{1}{2} + \mathsf{Adv}_0.$$

Game 1. The challenger in this game proceeds as before, but it aborts and chooses b' uniformly random, if there exists any oracle that accepts maliciously in the sense of Definition 3.4. Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}},$$

where $\epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}$ is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 3.4 (cf. Lemmas 5.1 and 5.2).

Recall that we assume that $\mathcal{A}$ always asks a Test query such that all conditions in Property 2 of Definition 3.4 are satisfied. In particular it asks a Test query to an oracle π_i^s that 'accepts' after the τ_0-th query of $\mathcal{A}$ with intended partner $\Pi = j$, such that P_j is τ_j-corrupted with $\tau_j > \tau_0$. Note that in Game 1 for any such oracle π_i^s there exists a unique 'partner oracle' π_j^t such that π_i^s has a matching conversation to π_j^t, as the game is aborted otherwise.

Game 2. The challenger in this game proceeds as before, but in addition guesses indices $(i^*, s^*) \xleftarrow{\$} [\phi] \times [\eta]$. It aborts and chooses b' at random, if the adversary issues a $\mathsf{Test}(\pi_i^s)$ query with $(i,s) \neq (i^*, s^*)$. With probability $1/(\eta\phi)$ we have $(i,s) = (i^*, s^*)$, and thus

$$\mathsf{Adv}_1 \leq \eta\phi \cdot \mathsf{Adv}_2.$$

Note that in Game 2 we know that $\mathcal{A}$ will issue a Test query to oracle $\pi_{i^*}^{s^*}$. Note also that $\pi_{i^*}^{s^*}$ has a unique 'partner' due to Game 1. In the sequel we denote with $\pi_{j^*}^{t^*}$ the unique oracle such that $\pi_{i^*}^{s^*}$ has a matching conversation to $\pi_{j^*}^{t^*}$, and say that $\pi_{j^*}^{t^*}$ is the *partner* of $\pi_{i^*}^{s^*}$.

Game 3. Let $\mathsf{T}^{i^*,s^*} = g^u$ denote the Diffie-Hellman share chosen by $\pi_{i^*}^{s^*}$, and let $\mathsf{T}^{j^*,t^*} = g^v$ denote the share chosen by its partner $\pi_{j^*}^{t^*}$. Thus, both oracles compute the premaster secret as $pms = g^{uv}$.

The challenger in this game proceeds as before, but replaces the premaster secret pms of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random group element $\widetilde{pms} = g^w$, $w \xleftarrow{\$} \mathbb{Z}_q$. Note that both g^u and g^v are chosen by oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively, as otherwise $\pi_{i^*}^{s^*}$ would not have a matching conversation to $\pi_{j^*}^{t^*}$ and the game would be aborted.

Suppose that there exists an algorithm $\mathcal{A}$ distinguishing Game 3 from Game 2. Then we can construct an algorithm $\mathcal{B}$ solving the DDH problem as follows. $\mathcal{B}$ receives as input (g, g^u, g^v, g^w). It implements the challenger for $\mathcal{A}$ as in Game 2, except that it sets $\mathsf{T}^{i^*,s^*} := g^u$ and $\mathsf{T}^{j^*,t^*} := g^v$, and the premaster secret of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ equal to $pms := g^w$. Note that $\mathcal{B}$ can simulate all messages exchanged between $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ properly, in particular the finished messages using knowledge of $pms = g^w$. Since all other oracles are not modified, $\mathcal{B}$ can simulate these oracles properly as well.

If $w = uv$, then the view of $\mathcal{A}$ when interacting with $\mathcal{B}$ is identical to Game 2, while if $w \xleftarrow{\$} \mathbb{Z}_q$ then it is identical to Game 3. Thus, the DDH assumption implies that

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{ddh}}.$$

Game 4. In Game 4 we make use of the fact that the premaster secret $\widetilde{pms}$ of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is chosen uniformly random. We thus replace the value $ms = \mathsf{PRF}(\widetilde{pms}, label_1 || r_C || r_S)$ with a random value $\widetilde{ms}$.

Distinguishing Game 4 from Game 3 implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{prf}}.$$

Game 5. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random function $F_{\widetilde{ms}}$. Of course the same random function is used for both oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$. In particular, this function is used to compute the key material as

$$K_{\mathsf{enc}}^{C \to S} || K_{\mathsf{enc}}^{S \to C} || K_{\mathsf{mac}}^{C \to S} || K_{\mathsf{mac}}^{S \to C} := F_{\widetilde{ms}}(label_2 || r_C || r_S)$$

Distinguishing Game 5 from Game 4 again implies an algorithm breaking the security of the pseudo-random function PRF. Moreover, in Game 5 the adversary always receives a random key in response to a Test query, thus receives no information about b', which implies $\mathsf{Adv}_5 = 0$ and

$$\mathsf{Adv}_4 \le \mathsf{Adv}_5 + \epsilon_{\mathsf{prf}} = \epsilon_{\mathsf{prf}}.$$

Collecting probabilities from Game 0 to Game 5 yields Lemma 5.3. □

Summing up probabilities from Lemmas 5.1 to 5.3, we obtain that

$$\begin{aligned}
\epsilon_{\mathsf{ttls}} &\le \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} + \epsilon_{\mathsf{enc}} \\
&\le 2 \cdot (\epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}) + \eta\phi \cdot (\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}}) \\
&\le 4 \cdot \max\{\epsilon_{\mathsf{client}}, \epsilon_{\mathsf{server}}\} + \eta\phi \cdot (\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}}) \\
&\le 4 \cdot \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^\mu} \right) \right) \\
&\quad + \eta\phi \cdot (\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}}) \\
&= 4 \cdot \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \frac{5}{4} \cdot \epsilon_{\mathsf{ddh}} + \frac{5}{2} \cdot \epsilon_{\mathsf{prf}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^\mu} \right) \right),
\end{aligned}$$

which yields Theorem 5.1.

5.3 TLS-DHE with Mutual Authentication is a Secure ACCE Protocol

We now adapt the proof of truncated TLS-DHE given in the previous section and additionally reduce to the security of the symmetric encryption scheme used to encrypt the `Finished` messages. We then receive a proof of the full TLS handshake protocol.

Theorem 5.2. *Let μ be the output length of* PRF *and let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the signature scheme used to sign messages in TLS is $(t, \epsilon_{\mathsf{sig}}, q_{\mathsf{sig}})$-secure, the DDH-problem is $(t, \epsilon_{\mathsf{ddh}})$-hard in the group G used to compute the TLS premaster secret, the hash function* H *is $(t, \epsilon_{\mathsf{H}})$-collision resistant, the PRF-ODH-*

problem is $(t, \epsilon_{\mathsf{prfodh}})$-hard with respect to G and PRF, and $q_{\mathsf{prf}} \leq \eta$ and $\phi \leq q_{\mathsf{sig}}$. Suppose that the stateful symmetric encryption scheme is $(t, \epsilon_{\mathsf{sLHAE}})$-secure.

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the ephemeral Diffie-Hellman TLS protocol in the sense of Definition 3.9 with $t \approx t'$ holds that

$$\epsilon_{\mathsf{tls}} \leq 4\eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi\epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + \eta\phi\epsilon_{\mathsf{prfodh}} + (\eta\phi + 2) \cdot \left(\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^\mu} \right) \right).$$

To prove Theorem 5.2, we divide the set of all adversaries into two categories:

1. Adversaries that succeed in making an oracle accept maliciously. We call such an adversary an authentication-adversary.

2. Adversaries that do not succeed in making any oracle accept maliciously, but which answer the encryption-challenge. We call such an adversary an encryption-adversary.

We prove Theorem 5.2 by the following two lemmas.

Lemma 5.4 bounds the probability ϵ_{auth} that an authentication-adversary succeeds and Lemma 5.5 bounds the probability ϵ_{enc} that an encryption-adversary succeeds. Then we have

$$\epsilon_{\mathsf{tls}} \leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}}.$$

Lemma 5.4. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s that accepts maliciously is at most*

$$\epsilon_{\mathsf{auth}} \leq 2 \cdot \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + (\eta\phi + 2) \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^\mu} \right) \right)$$

where all quantities are defined as stated in Theorem 5.2.

Note that $\epsilon_{\mathsf{auth}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}$, where $\epsilon_{\mathsf{client}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Client}$ that accepts maliciously in the sense of Definition 3.9, and $\epsilon_{\mathsf{server}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Server}$ that accepts maliciously. We claim that

$$\epsilon_{\mathsf{client}} \leq \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^\mu} \right) \right)$$
$$\epsilon_{\mathsf{server}} \leq \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^\mu} \right)$$

and thus

$$\begin{aligned}
\epsilon_{\mathsf{auth}} &\leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} \\
&\leq \eta\phi \left(\frac{\eta\phi}{2^{\lambda-1}} + 2\phi\epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + \eta\phi\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} \right) \\
&\qquad + \eta\phi\,(\eta\phi + 1) \cdot \left(\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^{\mu}} \right) \\
&\leq \eta\phi \left(\frac{\eta\phi}{2^{\lambda-1}} + 2\phi\epsilon_{\mathsf{sig}} + \eta\phi\epsilon_{\mathsf{prfodh}} + (\eta\phi + 2) \cdot \left(\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^{\mu}} \right) \right).
\end{aligned}$$

The bounds on $\epsilon_{\mathsf{client}}$ and $\epsilon_{\mathsf{server}}$ are derived similar to the proofs of Lemma 5.1 and Lemma 5.2 in Section 5.2.1. We only extend the proofs by one game-hop that exploits the sLHAE-security of the encryption scheme. This is necessary, as an adversary can violate the matching conversations definition, and thus make an oracle malicously accept, by creating a new, valid encryption of fin_C (or fin_S), which is distinct from the ciphertext output by the corresponding client (or server) oracle.

Lemma 5.5. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ anwers the encryption-challenge correctly is at most $1/2 + \epsilon_{\mathsf{enc}}$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + \eta\phi\,(\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}})\,.$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 3.9 (cf. Lemma 5.4) and all other quantities are defined as stated in Theorem 5.2.

The proof of this lemma again extends the proof of Lemma 5.3 by one game-hop that exploits the sLHAE-security of the encryption scheme.

Proof. Assume without loss of generality that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 3.9 are satisfied. Let $\mathsf{break}_\delta^{(4)}$ denote the event that $b' = b_i^s$ in Game δ, where b_i^s is the random bit sampled by π_i^s, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta^{(4)}] - 1/2$ denote the *advantage* of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This game equals the ACCE security experiment described in Section 3.2.2. For some ϵ_{enc} we have

$$\Pr[\mathsf{break}_0^{(4)}] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = \frac{1}{2} + \mathsf{Adv}_0.$$

Game 1. The challenger in this game proceeds as before, but it aborts and chooses b' uniformly random, if there exists any oracle that accepts maliciously in the sense of Definition 3.9. Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}},$$

where ϵ_{auth} an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 3.9 with mutual authentication (cf. Lemma 5.4).

Recall that we assume that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 3.9 are satisfied. In particular it outputs (i, s, b') such that π_i^s 'accepts' after the τ_0-th query of $\mathcal{A}$ with intended partner $\Pi = j$, and P_j is τ_j-corrupted with $\tau_j > \tau_0$. Note that in Game 1 for any such oracle π_i^s there exists a unique 'partner oracle' π_j^t such that π_i^s has a matching conversation to π_j^t, as the game is aborted otherwise.

Game 2. The challenger in this game proceeds as before, but in addition guesses indices $(i^*, s^*) \xleftarrow{\$} [\phi] \times [\eta]$. It aborts and chooses b' at random, if the adversary outputs (i, s, b') with $(i, s) \neq (i^*, s^*)$. With probability $1/(\eta\phi)$ we have $(i, s) = (i^*, s^*)$, and thus

$$\mathsf{Adv}_1 \leq \eta\phi \cdot \mathsf{Adv}_2.$$

Note that in Game 2 we know that $\mathcal{A}$ will output (i^*, s^*, b'). Note also that $\pi_{i^*}^{s^*}$ has a unique 'partner' due to Game 1. In the sequel we denote with $\pi_{j^*}^{t^*}$ the unique oracle such that $\pi_{i^*}^{s^*}$ has a matching conversation to $\pi_{j^*}^{t^*}$, and say that $\pi_{j^*}^{t^*}$ is the *partner* of $\pi_{i^*}^{s^*}$.

Game 3. The challenger in this game proceeds as before, but replaces the premaster secret pms of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random group element $\widetilde{pms} = g^w$, $w \xleftarrow{\$} \mathbb{Z}_q$. Note that both g^u and g^v are chosen by oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively, as otherwise $\pi_{i^*}^{s^*}$ would not have a matching conversation to $\pi_{j^*}^{t^*}$ and the game would be aborted.

With the same arguments as in Game 3 in the proof of Lemma 5.3, we have

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{ddh}}.$$

Game 4. As in Game 4 in the proof of Lemma 5.3, we now make use of the fact that the premaster secret $\widetilde{pms}$ of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is chosen uniformly random. We thus replace the value $ms = \mathsf{PRF}(\widetilde{pms}, label_1 || r_C || r_S)$ with a random value $\widetilde{ms}$.

Distinguishing Game 4 from Game 3 implies an algorithm breaking the security of the pseudo-random function PRF, thus

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{prf}}.$$

Game 5. As in Game 5 in the proof of Lemma 5.3, we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random function $F_{\widetilde{ms}}$. Of course the same random function is used for both oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$. In particular, this function is used to compute the key material as

$$K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C} := F_{\widetilde{ms}}(label_2||r_C||r_S)$$

Distinguishing Game 5 from Game 4 again implies an algorithm breaking the security of the pseudo-random function PRF, thus we have

$$\mathsf{Adv}_4 \leq \mathsf{Adv}_5 + \epsilon_{\mathsf{prf}}.$$

Note that in Game 5 the key material $K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C}$ of oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is uniformly random and independent of all TLS Handshake messages exchanged in the pre-accept stage.

Game 6. Now we use that the key material $K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C}$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ in the stateful symmetric encryption scheme uniformly at random and independent of all TLS Handshake messages.

In this game we construct a simulator $\mathcal{B}$ that uses a successful ACCE adversary $\mathcal{A}$ to break the security of the underlying sLHAE secure symmetric encryption scheme (Definition 2.11). By assumption, the simulator $\mathcal{B}$ is given access to an encryption oracle $\mathsf{Encrypt}$ and a decryption oracle $\mathsf{Decrypt}$. $\mathcal{B}$ embeds the sLHAE experiment by simply forwarding all $\mathsf{Encrypt}(\pi_{i^*}^{s^*}, \cdot)$ queries to $\mathsf{Encrypt}$, and all $\mathsf{Decrypt}(\pi_{j^*}^{t^*}, \cdot)$ queries to $\mathsf{Decrypt}$. Otherwise it proceeds as the challenger in Game 5.

Observe that the values generated in this game are exactly distributed as in the previous game. We thus have

$$\mathsf{Adv}_5 = \mathsf{Adv}_6.$$

If $\mathcal{A}$ outputs a triple (i^*, s^*, b'), then $\mathcal{B}$ forwards b' to the sLHAE challenger. Otherwise it outputs a random bit. Since the simulator essentially relays all messages it is easy to see that an adversary $\mathcal{A}$ having advantage ϵ' yields an adversary $\mathcal{B}$ against the sLHAE security of the encryption scheme with success probability at least $1/2 + \epsilon'$.

Since by assumption any adversary has advantage at most $\epsilon_{\mathsf{sLHAE}}$ in breaking the sLHAE security of the symmetric encryption scheme, we have

$$\mathsf{Adv}_6 \leq 1/2 + \epsilon_{\mathsf{sLHAE}}.$$

□

Addig up probabilities from Lemmas 5.4 and 5.5, we obtain that

$$\begin{aligned}\epsilon_{\mathsf{tls}} &\leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}} \\ &\leq 2 \cdot \epsilon_{\mathsf{auth}} + \eta\phi\left(\epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}\right) \\ &\leq \eta\phi\left(\frac{\eta\phi}{2^{\lambda-2}} + 4\phi\epsilon_{\mathsf{sig}} + 3\epsilon_{\mathsf{ddh}} + 2\eta\phi\epsilon_{\mathsf{prfodh}} + 4\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}\right) \\ &\qquad + 2\eta\phi\left(\eta\phi + 1\right) \cdot \left(\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^{\mu}}\right)\end{aligned}$$

which yields Theorem 5.2.

5.4 General Proof Idea for Subsequent Proofs

For simplicity let us concentrate on the proof idea for server-only authentication. In the following let $k_S := k_{\mathsf{enc}}^{\mathsf{Server}}$ denote the symmetric encryption key computed by the server and $k_C := k_{\mathsf{enc}}^{\mathsf{Client}}(= k_{\mathsf{dec}}^{\mathsf{Server}})$ denote the encryption key computed by the client. Let $k_S^{(C)} := k_{\mathsf{dec}}^{\mathsf{Client}} = k_{\mathsf{enc}}^{\mathsf{Server}}$ denote the symmetric key that is computed by the client oracle to decrypt messages encrypted by the server, let $k_S^{(S)}$ be the corresponding key computed by the server, and let $k_C^{(C)}$ and $k_C^{(S)}$ be the encryption keys of the client computed by client and server, respectively. Similarly, we let $m^{(S)}$ be a message sent or received or a value computed by the server, and $m^{(C)}$ by a client respectively. For authentication we consider an adversary that makes the client oracle maliciously accept, i.e. it accepts although there is no matching conversation with some other uncorrupted oracle. We consider several cases and subcases.

In **Case 1** the adversary does not modify the client Diffie-Hellman share (in TLS-DHE and TLS-SDH) or the encrypted premaster secret (in TLS-RSA), whereas in **Case 2** it does so. In **Case 1** we now consider two subcases. In **Case 1.1** the adversary does not modify any of the random nonces r_C, r_S. In **Case 1.2** it does make such a modification. In **Case 1.1** we again consider two subcases. In **Case 1.1.1** the adversary does not modify any of the remaining messages m_1 to m_{11} exchanged between client oracle and server oracle, whereas in **Case 1.1.2**, it does so. We now reduce each of these cases to the security of the stateful encryption scheme. We always embed the sLHAE challenge key into $k_S^{(C)}$, the symmetric key that is computed by the client oracle to decrypt the encryption of fin_S. In **Case 1.1.1** and **Case 1.1.2** we have that $k_S^{(C)} = k_S^{(S)}$, i.e. client oracle and server oracle compute the same key for k_S and the queries granted in the sHLAE security game are used to compute the output m_{13} of the server oracle.

We have to show that the adversary *even with the help of the server oracle* cannot make the client oracle accept without breaking one of the underlying security assumptions.

In **Case 1.1.1**, the master secrets, the encryption keys and the fin_S messages computed by client and server oracle are equal and all indistinguishable from random from the adversary's point of view. By the definition of security (cf. p. 36 and 38), the adversary must thus have computed a *new* encryption m'_{12} of fin_S which is distinct from the message m_{13} that is computed by the server oracle to make the client oracle accept. If we embed the sLHAE challenge key into $k_S^{(C)} = k_S^{(S)}$, and use the encryption queries granted in the sLHAE security game to compute m_{13} we can directly break the security of the sLHAE scheme. In **Case 1.1.2** the master secret and encryption keys computed by client and server oracle are equal and indistinguishable from random. However, when client and server oracle compute the `Finished` messages they use distinct transcripts. First, by the security of the hash function the hash values of these transcripts differ as well. Second, by the security of the PRF and since the input values to the PRF are distinct, the values for fin_S computed by the client oracle and server oracle, $\mathsf{fin}_S^{(C)}$ and $\mathsf{fin}_S^{(S)}$, must differ as well for the two oracles with overwhelming probability $2^{-\mu}$. In this case the adversary cannot use the output of the server oracle to make the client oracle accept because due to the correctness of the sLHAE encryption system there cannot be two distinct plaintext for one ciphertext. Instead it has to compute a new encryption of $\mathsf{fin}_S^{(C)}$ on its own. Such an adversary can directly be used to break the security of the sLHAE scheme.

In **Case 1.2** the adversary modifies (at least) one of the nonces r_C, r_S exchanged between client and server oracle. In **Case 2** we have that the adversary modifies the message m_8 sent by the client, such that the premaster secrets computed by client and server are distinct with overwhelming probability. The proofs for **Case 1.2** and **Case 2** are similar. In **Case 1.2** we first have to show for each ciphersuite that the premaster secret pms exchanged between client and server oracle is indistinguishable from a random value to the adversary. In **Case 2**, we need to show that $pms^{(C)}$ is indistinguishable from a random value to the adversary and that it is distinct from (and actually independent of) $pms^{(S)}$ (with probability 2^{ν} where ν is the bitsize of pms) because of the adversary's modifications to m_8. Next we have to show that the master secret computed by the client oracle $ms^{(C)}$ is indistinguishable from random and distinct to $ms^{(S)}$. In **Case 2** this is because the key to thePRF, $pms^{(C)}$, is already indistinguishable from random and distinct from $pms^{(S)}$. In **Case 1.2**, this is because client and server use distinct nonces r_S, r_C as input to the PRF to compute the master secrets while $pms^{(C)} = pms^{(S)}$ is indistinguishable from random to the adversary. So by the security of thePRF, with probability $2^{-\mu}$ $ms^{(C)}$ is distinct from $ms^{(S)}$. We can now substitute the output values $k_C^{(C)}, k_S^{(C)}, \mathsf{fin}_C^{(C)}, \mathsf{fin}_S^{(C)}$ of the second application of the PRF with truly random values. By the security of the PRF the adversary cannot recognize this modification. Now, since a) $k_S^{(C)}$ is drawn uniformly at random and b) $k_S^{(C)}$ is never used at any point before, we can draw $k_S^{(C)}$ also after the client oracle has received the last protocol message m_{13} (and independently of the adversary and all the computations within the server oracle). This shows that no information about the random key $k_S^{(C)}$ is leaked. We can thus embed the sLHAE challenge key in $k_S^{(C)}$. If the adversary makes the client oracle accept we can directly break the security of the sLHAE scheme. In **Case 1.2** and **Case 2**, we do not have to exploit any of the $\mathsf{Encrypt}$ queries granted by the sLHAE security game. This shows authentication.

To show that the adversary cannot break the encryption challenge we observe that *in TLS the client only accepts, if all the values used for the computation of the encryption keys are unmodified.* In particular, **Case 1.1.2** also rules out that the adversary modifies one of the remaining messages which are sent from the client oracle to the server oracle. If the client oracle accepts, both client and server oracle thus have computed the same encryption keys. We can now directly plugin the sLHAE challenge either in $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. Any adversary that breaks the encryption challenge can be used to break the sLHAE security.

5.5 TLS-DHE with Server-only Authentication is ACCE Secure

Theorem 5.3. *Let μ be the output length of the PRF and λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-*

secure, the hash function is $(t, \epsilon_{\mathsf{H}})$-secure, the signature scheme used to sign messages in TLS is $(t, \epsilon_{\mathsf{sig}}, q_{\mathsf{sig}})$-secure, the sLHAE scheme is $(t, \epsilon_{\mathsf{sLHAE}})$-secure, and the PRF-ODH-problem is $(t, \epsilon_{\mathsf{prfodh}})$-hard with respect to G and PRF.

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the TLS-DHE Handshake protocol with server-only authentication in the sense of Definition 3.11 (with perfect forward secrecy) with $t \approx t'$ and $q_{\mathsf{prf}} \geq 1$, and $q_{\mathsf{sig}} \geq \eta$, it holds that

$$\epsilon_{\mathsf{tls}} \leq 2 \cdot \frac{(\eta\phi)^2}{2^{\lambda-1}} + 2\phi \cdot \epsilon_{\mathsf{sig}} + 8(\eta\phi)^2 \cdot \left(2^{-\mu} + 2^{-\nu} + \epsilon_{\mathsf{H}}\right) + 9(\eta\phi)^2 \cdot (\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{sLHAE}}) .$$

When proving server-only authentication we consider two types of adversaries.

Type 1. The adversary breaks the server-only authentication property of Definition 3.11 for a client oracle (Property 1).

Type 2. The adversary answers the encryption challenge correctly for clients that do not accept maliciously (Property 2).

To prove that no adversary of **Type 1** exists (again except for some negligible error probability) we proceed as follows. At first we exclude certificate forgeries and ensure uniqueness of the random nonces. Then we exclude modifications of the random nonces and the server share by reducing security to that of the signature scheme, since the server signature is computed over all these values. To do so, technically we first guess the server party j for which the adversary outputs a signature forgery with probability $\frac{1}{\phi}$. We replace the public key pk_j of this party with the challenge public key pk' of the signature security game. For simulation of oracles of party P_j we use the signature oracle granted in the signature security game. We then guess the server oracle π_j^t and its partner (client) oracle π_i^s with probability $1/(d^2\phi)$. (Note that we have already guessed the server party holding the server oracle.)

We now consider two cases: either the client Diffie-Hellman share has been modified by the adversary (**Case 2**) or not (**Case 1**). In **Case 1** we directly substitute the master secret ms that is computed between oracles π_i^s and π_j^t by a truly random value. Any adversary that can recognize our modification can be used to break the PRF-ODH assumption.[27] To this end consider that we send the concatenation of the fixed label $label_1$ and the random nonces to the PRF-ODH challenger who responds with z_b, g^u and g^v. (For details see Definition 2.9.) Now we first embed the g^v share from the PRF-ODH challenge in the message m_4 sent by π_j^t. Then we set g^u to be the Diffie-Hellman share of π_i^s included in message m_8 and use z_b as the master secret for the client and server oracle. Note that g^u and g^v are distributed exactly as before, since they are chosen at random. Also we now

[27]Due to the nature of the PRF-ODH assumption we do not replace the premaster secret pms separately.

use the PRF-ODH oracle to simulate that π_j^t has access to the secret key v (in case the adversary modifies m_8). Now if z_b is the real output of PRF this corresponds to the situation where ms is computed according to the protocol specification. However if z_b is random this exactly corresponds to the situation where ms is drawn uniformly random. So under the PRF-ODH assumption no adversary can notice our substitution of the master secret with a truly random value in **Case 1**. The remaining part of the proof is similar to that of TLS-RSA with server-only authentication: we now again consider two subcases of **Case 1**, **Case 1.1.1** and **Case 1.1.2**. Observe that in this proof we do not require two additional subcases **Case 1.1** and **Case 1.2** since the random nonces are protected by adversarial modifications via the signatures. Either one of the messages m_1 to m_{11} have been modified in transit (**Case 1.1.2**) or not (**Case 1.1.1**). In **Case 1.1.1** we thus assume that only the message m_{13} has been modified. In this case the adversary has to compute a new encryption of $\mathsf{fin}_S^{(C)} = \mathsf{fin}_S^{(S)}$ to make the client accept which breaks the security of the sLHAE scheme. If the adversary modifies m_1 to m_{11}, the `Finished` messages fin_S (i.e. $\mathsf{fin}_S^{(C)}$ and $\mathsf{fin}_S^{(S)}$), can, exploiting the security of the hash function and thePRF, be substituted with random values that are distinct with probability at least $1 - 2^{-\mu}$. So in **Case 1.1.2**, $\mathsf{fin}_S^{(S)}$ is independent from $\mathsf{fin}_S^{(C)}$ and the adversary cannot use the output of the server oracle to compute m_{13} such that the client accepts and has to compute this message on its own. However, this breaks the security of the sLHAE scheme. In **Case 1.1.1** and **Case 1.1.2** we use the $\mathsf{Encrypt}$ queries to compute the output of the server oracle m_{13}.

The **Cases 2** can be reduced to the security of the sLHAE scheme. We embed the sLHAE challenge in the key $k_S^{(C)}$ of the client oracle that is used to decrypt server messages. As in previous proofs $k_S^{(C)}$ is independent of the adversary and all computations made by the server oracle. However, now any message that makes the client accept breaks the security of the sLHAE scheme.

To prove that no adversary of **Type 2** exists (again except for some negligible error probability), we exploit that in the previous proof we technically also showed, that the client will not accept if any of the messages sent by the client will be modified. Since any message that is used to derive the encryption keys will not be modified, the encryption keys of client and server oracle are equal. With the same arguments as before, we can now substitute the encryption keys by random values and directly plugin the sLHAE challenge in either $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the $\mathsf{Encrypt}$ query of the sLHAE game. Similarly the Decrypt query is used to decrypt all these queries on behalf of the receiving oracle. Since both DH shares are drawn randomly in each session we also have perfect forward secrecy.

5.6 TLS-RSA with Server-Only Authentication is ACCE Secure

In this section we show that TLS-RSA with server-only authentication is ACCE secure under Definition 3.10.

Theorem 5.4. *Let $\mu = \mu(\kappa)$ be the output length of thePRF, $\nu = \nu(\kappa)$ be the length of the premaster secret, and $\lambda = \lambda(\kappa)$ be the length of the nonces r_C and r_S. Assume that the pseudo-random function is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the hash function is $(t, \epsilon_{\mathsf{H}})$-secure, the public key encryption scheme is $(t, \epsilon_{\mathsf{pke}}, q_{\mathsf{pke}})$-secure, and the sLHAE scheme is $(t, \epsilon_{\mathsf{sLHAE}})$-secure. Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the server-only authenticated TLS-RSA Handshake protocol in the sense of Definition 3.10 with $t \approx t'$ and $q_{\mathsf{pke}} \geq \eta$, and $q_{\mathsf{prf}} \geq 2$, it holds that*

$$\epsilon_{\mathsf{tls}} \leq \frac{(\eta\phi)^2}{2^{\lambda-1}} + 8\eta\phi^2(2^{-\nu} + 2^{-\mu} + \epsilon_{\mathsf{H}}) + (8\eta\phi^2 + \eta\phi)(\epsilon_{\mathsf{pke}} + 2\epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{sLHAE}}).$$

We prove Theorem 5.4 via two lemmas. Lemma 5.6 bounds the probability ϵ_{auth} that an adversary succeeds in making the client oracle accept maliciously. Lemma 5.7 bounds the probability ϵ_{enc} that an adversary answers the encryption challenge correctly while not making client oracle accept maliciously. Then we have

$$\epsilon_{\mathsf{tls}} \leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}}.$$

5.6.1 Server-Only Authentication

Lemma 5.6. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists a client oracle π_i^s that accepts maliciously with respect to Definition 3.10 is at most*

$$\epsilon_{\mathsf{auth}} \leq \frac{(\eta\phi)^2}{2^{\lambda}} + 4\eta\phi^2(\epsilon_{\mathsf{sLHAE}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + 2^{-\nu} + 2^{-\mu} + \epsilon_{\mathsf{pke}}).$$

where all quantities are defined as stated in Theorem 5.4.

Proof. The proof proceeds in a *sequence of games*, following [BR06, Sho04]. The first game is the real security experiment. We then describe several intermediate games that modify the original game step-by-step, and argue that our complexity assumptions imply that each game is computationally indistinguishable from the previous one. In particular, the difference of the success probability of the adversary in two subsequent game is negligible. We end up in the final game, where no adversary can break the security of the protocol.

Let $\mathsf{break}_\delta^{(1)}$ be the event that occurs when a client oracle accepts maliciously in the sense of Definition 3.10 in Game δ. If required, we use $x^{(S)}$ to denote the value of x that is computed by a server oracle and $x^{(C)}$ the value of x that is computed by a client oracle (e.g. $pms^{(S)}$ denotes the premaster secret computed by the server).

Game 0. This game equals the ACCE security experiment described in Section 3.2.2. Thus, for some ϵ_{auth} we have

$$\Pr[\mathsf{break}_0^{(1)}] = \epsilon_{\mathsf{auth}}.$$

Game 1. In this game we add an abort rule. The challenger aborts, if there exists any oracle π_i^s that chooses a random nonce r_C or r_S which is not unique. More precisely, the game is aborted if the adversary ever makes a first Send query to an oracle π_i^s, and the oracle replies with random nonce r_C or r_S such that there exists some other oracle $\pi_{i'}^{s'}$ which has previously sampled the same nonce.

In total less than $\eta\phi$ nonces r_C and r_S are sampled, each uniformly random from $\{0,1\}^\lambda$. Thus, the probability that a collision occurs is bounded by $(\eta\phi)^2 \cdot 2^{-\lambda}$, which implies

$$\Pr[\mathsf{break}_0^{(1)}] \leq \Pr[\mathsf{break}_1^{(1)}] + \frac{(\eta\phi)^2}{2^\lambda}.$$

Note that now each oracle has a unique nonce r_C or r_S, which will be used to compute the master secret.

Game 2. We try to guess which client oracle will be the first to accept maliciously. If our guess is wrong, we abort the game.

Technically, this game is identical to the previous, except for the following. The challenger guesses two random indices $(i^*, s^*) \overset{\$}{\leftarrow} [\phi] \times [\eta]$. If π_i^s is the first client oracle that 'accepts' maliciously and $(i,s) \neq (i^*, s^*)$, then the challenger aborts the game. Since there are at most $\eta\phi$ oracles the probability to guess correctly, i.e. we have $(i,s) = (i^*, s^*)$, is $\geq 1/(\eta\phi)$, and thus

$$\Pr[\mathsf{break}_1^{(1)}] \leq \eta\phi \cdot \Pr[\mathsf{break}_2^{(1)}].$$

Note that in this game the adversary can only break the security of the protocol, if oracle $\pi_{i^*}^{s^*}$ is the first client oracle that 'accepts' maliciously, as otherwise the game is aborted.

Game 3. In this game we guess the party that holds the communication partner of the client oracle. Technically we draw $j^* \xleftarrow{\$} [\phi]$. If $\Pi \neq j^*$ for client oracle $\pi_{i^*}^{s^*}$ we abort.

We get

$$\Pr[\mathsf{break}_2^{(1)}] \leq \phi \cdot \Pr[\mathsf{break}_3^{(1)}].$$

In the following we will consider two distinct cases. In the first case, **Case 1**, the adversary does not modify the ciphertext containing the premaster secret, message m_8, which is sent from the client oracle to the server oracle. In the second case, **Case 2**, it does so.

Case 1:

Game 4. In this game we exchange the ciphertext c of the premaster secret sent by $\pi_{i^*}^{s^*}$, message m_8, by an encryption c' of a truly random message r (and independently of pms). However, both client and server continue to use pms as computed by the client. Since the challenger implements all server oracles it can, whenever the ciphertext c' is received by any server oracle of P_{j^*}, set the premaster secret to pms. We show that this modification is indistinguishable from the previous game when the Public-Key Encryption (PKE) scheme is secure. Any adversary that can distinguish these two games can be used to break the security of the public key scheme as follows.

First, we embed the challenge public key of the PKE challenger in pk_{j^*}. Next we draw a random pms for oracle $\pi_{i^*}^{s^*}$ and send it to the PKE challenger. The challenge in turn responds with an encryption c^* of either pms or an independently drawn random value r. Let $\pi_{j^*}^{t}$ be the server oracle where we exchange m_8 with c^* but continue to use the original pms. For all other oracles $\pi_{j^*}^{t}$ of P_{j^*} with $t \in [\eta]$ and $t \neq t^*$ we use the decryption queries granted in the PKE security game to decrypt m_8 and set the pms of the server oracle as given in the plaintext of m_8 unless the server oracle receives c^*. Whenever c^* is received by any server oracle we simply set the premaster secret of that oracle to pms. Observe that if c^* is an encryption of pms we are in the previous game. However, if c^* encrypts an independently drawn random message we are in the current game. So any adversary that distinguishes these two games can directly be used to break the security of the PKE scheme.

We have

$$\Pr[\mathsf{break}_3^{(1)}] \leq \Pr[\mathsf{break}_4^{(1)}] + \epsilon_{\mathsf{pke}}.$$

We note that in this game the adversary does not obtain any information on pms from m_8.

We now consider two subcases. Either the adversary does not modify any of the messages r_C or r_S in transit (**Case 1.1**) or it does so (**Case 1.2**).

<u>**Case 1.1:**</u>

Game 5. In this game, we substitute the master secret generated by the client oracle and the server oracle by a single truly random value. Any adversary that can distinguish between this and the previous game can be used to break the security of the function PRF.

To show this, observe that since pms is drawn uniformly at random and no information about pms is revealed to the adversary via m_8 due the previous game we can directly embed the security challenge of the PRF challenger: we use a PRF query (granted in the PRF security game) to compute the master secret. If the answer has been computed with PRF we are in the previous game. If the answer is a truly random value, we are in the current game. So any adversary that distinguishes this game from the previous can directly be used to break the security of the PRF.

We have,

$$\Pr[\mathsf{break}_4^{(1)}] \leq \Pr[\mathsf{break}_5^{(1)}] + \epsilon_{\mathsf{prf}}.$$

Recall that neither r_S nor r_C have been modified by the adversary. We are now in a game where the master secret ms of client and server oracle are equal and the truly random function used for computing ms can only be accessed by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$.
At this point we define another two subcases. Either the adversary does not modified any of the (remaining) messages of m_1 to m_{11}, **Case 1.1.1**, or it does, **Case 1.1.2**.

<u>**Case 1.1.1:**</u>

In this case, the adversary must by definition modify m_{13} to make the client oracle accept maliciously.

Game 6. However, we now guarantee, that the adversary has no knowledge of the encryption keys generated by the client oracle and the server oracle. We do so by first replacing the PRF used to compute these keys (and fin_S) by a truly random function, and then substituting fin_S and the encryption keys k_S and k_C with truly random values. Any adversary that can detect this modification can be used to break the security of the PRF.

To show this, observe that since ms is drawn uniformly at random we can directly embed the security challenge of the PRF challenger: we use a PRF query (granted in the PRF security game) to compute the encryption keys and fin_S. If these values have been computed with thePRF, we are in the previous game. If these values are truly random, we are in the current game. So any adversary that distinguishes this game from the previous can directly be used to break the security of the PRF.

We have

$$\Pr[\mathsf{break}_5^{(1)}] \leq \Pr[\mathsf{break}_6^{(1)}] + \epsilon_{\mathsf{prf}}.$$

Game 7. In this game we show that any successfull adversary can be used to break the security of the encryption system. We know that the adversary outputs a message $m'_{12} \neq m_{13}$ which makes the client adversary accept. This can only happen if m'_{12} is a distinct, valid encryption of m_{13}. However, then m'_{12} can be used to break the security of the sLHAE encryption scheme as follows: since the encryption keys are random we can directly plug-in the sLHAE challenge (in k_S) that is used to encrypt and decrypt messages sent from the server oracle to the client oracle. When we need to generate m_{13} that is computed by the server oracle we use one $\mathsf{Encrypt}$ query as granted by the sLHAE challenger. Since the adversary outputs a new ciphertext m'_{12} that has not been generated by $\mathsf{Encrypt}$, this directly breaks the security of the sLHAE game.

$$\Pr[\mathsf{break}_6^{(1)}] \leq \epsilon_{\mathsf{sLHAE}}.$$

Collecting probabilities we get

$$\epsilon_{\mathsf{auth}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \eta\phi^2(\epsilon_{\mathsf{sLHAE}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{pke}}).$$

In the following, we will only explain the differences to the proof of **Case 1.1.1**.

Case 1.1.2:

In this case, the adversary has modified any of the remaining messages m_1 to m_{11} on transit. We now substitute the output values of the pseudo-random function keyed with ms by truly random values. Since the encryption keys do only depend on r_C and r_S, we can substitute them by the same random keys. However, the computation of fin_S does also depend on the messages m_1 to m_{11}. By assumption, the values m_1 to m_{11} computed and received by the two oracles do differ at some point. Thus, when hashing the concatenated messages, the corresponding outputs can only be equal with probability at most ϵ_{H} – otherwise we can use the adversary to produce a collision and break a security of the hash function. Since the inputs to PRF are now distinct with overwhelming probability we can substitute $\mathsf{fin}_S^{(S)}$

and $\mathsf{fin}_S^{(C)}$[28] by independently drawn random values. With the same argument as before, our modification can only be detected by the adversary with probability ϵ_{prf}. We now abort if the output values of the PRF are equal which only happens with probability $2^{-\mu}$.

At this point the server oracle cannot help the adversary to compute a message m_{13} which will make the client oracle accept. Thus, to make the client oracle accept, the adversary must compute m'_{12} to be an encryption of $\mathsf{fin}_S^{(C)}$. However, since $\mathsf{fin}_S^{(S)}$ is distinct from $\mathsf{fin}_S^{(C)}$ the adversary cannot use m_{13}.

We now embed the sLHAE challenge in $k_S^{(C)}$ which is used by the client to decrypt messages sent by the server. Then the adversary against server-only authentication outputs a new ciphertext m'_{12} that has not been generated by Encrypt which breaks the security of the sLHAE game.

Collecting probabilities we get

$$\epsilon_{\mathsf{auth}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \eta\phi^2(\epsilon_{\mathsf{sLHAE}} + 2\epsilon_{\mathsf{prf}} + 2^{-\mu} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{pke}}).$$

Case 1.2:

In this case the adversary has modified either r_C or r_S in transit. Since these values are input to the PRF when computing the master secret we can substitute $ms^{(S)}$ and $ms^{(C)}$ by independently drawn random values. We abort if these values are equal which only happens with probability $2^{-\mu}$. In the next step we replace $\mathsf{fin}_C^{(C)}$ and $\mathsf{fin}_S^{(C)}$ and the encryption keys $k_S^{(C)}$ $k_C^{(C)}$ with independently drawn truly random values. Due to the security of the PRF these modifications cannot be detected by the adversary. Since $k_S^{(C)}$ is never used at any point before we can also draw this value after the client oracle receives the encryption of fin_S and independently of all the computations of the adversary and the server oracle. We now embed the sLHAE challenge into $k_S^{(C)}$.

To make the client oracle accept the adversary must compute m'_{12} to be an encryption of $\mathsf{fin}_S^{(C)}$ under $k_S^{(C)}$. Thus, if the adversary outputs a new ciphertext m'_{12}, it directly breaks the security of the sLHAE game as before. Observe that to simulate the server oracle correctly, we do not have to query Encrypt to generate m_{13}.

Collecting probabilities we get

$$\epsilon_{\mathsf{auth}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \eta\phi^2(\epsilon_{\mathsf{sLHAE}} + 2\epsilon_{\mathsf{prf}} + 2^{-\mu} + \epsilon_{\mathsf{pke}}).$$

[28]Recall that $\mathsf{fin}_S^{(C)}$ refers to the `ServerFinished` message that is re-computed by the client (for verification)

Case 2:

We now have that the adversary modifies message m_8 which is sent from the client oracle to the server oracle and contains the premaster secret pms. We first substitute the ciphertext c (i.e. the encrypted premaster secret) that is sent by the client oracle, by an encryption of a truly random message r independently drawn from pms. If r is equal to pms which happens with negligible probability $2^{-\nu}$ we abort, otherwise we continue. The client will continue to use the randomly drawn pms. The server oracle will use whatever it receives as m_8. We show that our modification is indistinguishable from the previous game when the PKE is secure. Any adversary that can distinguish these two games can be used to break the security of the public key scheme as follows.

We embed the challenge public key of the PKE challenger in pk_{j^*}. For all other oracles $\pi^t_{j^*}$ of P_{j^*} with $t \in [\eta]$ and $t \neq t^*$ we use the decryption queries granted by the PKE challenger to decrypt m_8 messages. We let the client oracle draw the random pms and then sent it to the PKE challenger who returns a ciphertext c^* which either encrypts pms or a truly random value. Next we send c^* to the server oracle $\pi^{t^*}_{j^*}$. Observe that if c^* is an encryption of pms we are in the previous game. However, if c^* encrypts an independently drawn random message we are in the current game. So any adversary that distinguishes these two games can directly be used to break the security of the PKE scheme.

We now have that $pms^{(C)}$ and $pms^{(S)}$ are independent (while $pms^{(C)}$ is uniformly random). Observe that no information about $pms^{(C)}$ is revealed neither to the adversary nor to the server oracle. In the next step we replace the master secret $ms^{(C)}$ by a truly random value. Then with the same arguments as above we replace the `Finished` messages $\mathsf{fin}^{(C)}_C$ and $\mathsf{fin}^{(C)}_S$ and the encryption keys $k^{(C)}_S$ $k^{(C)}_C$ by independently drawn truly random values. These modifications are all indistinguishable by the adversary by the security of the PRF. Observe that now $k^{(C)}_S$ can also be drawn after the client oracle receives m_{13} (and independently of all previous values) since it is never used before. Now we can embed the sLHAE challenge key into $k^{(C)}_S$ and any message that makes the client oracle accept can be used to break the sLHAE scheme. We do not have to use any Encrypt query granted by the sLHAE security game.

Collecting probabilities we get

$$\epsilon_{\mathsf{auth}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \eta\phi^2(\epsilon_{\mathsf{sLHAE}} + 2\epsilon_{\mathsf{prf}} + 2^{-\nu} + \epsilon_{\mathsf{pke}}).$$

□

5.6.2 Indistinguishability of Ciphertexts

Lemma 5.7. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the encryption-challenge correctly is at most $1/2 + \epsilon_{\mathsf{enc}}$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + \eta\phi(\epsilon_{\mathsf{pke}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}).$$

where ϵ_{auth} is an upper bound on the probability that there exists a client oracle that accepts maliciously in the sense of Definition 3.10 with server-only authentication (Lemmas 5.6) and all other quantities are defined as stated in Theorem 5.4.

Proof. Assume without loss of generality that the adversary $\mathcal{A}$ always outputs a triple such that all conditions in Property 2 of Definition 3.10 are satisfied. Let $\mathsf{break}_\delta^{(2)}$ denote the event that $b' = b_i^s$ in Game δ, where b_i^s is the random bit sampled by the client oracle π_i^s, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta^{(2)}] - 1/2$ denote the *advantage* of $\mathcal{A}$ in Game δ.

Consider the following sequence of games.

Game 0. This game equals the ACCE security experiment described in Section 3.2.2. For some ϵ_{enc} we have

$$\Pr[\mathsf{break}_0^{(2)}] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = \frac{1}{2} + \mathsf{Adv}_0.$$

Game 1. The challenger in this game proceeds as before, but it aborts and chooses b' uniformly random, if there exists any oracle that accepts maliciously in the sense of the ACCE definition.

Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}},$$

where ϵ_{auth} is an upper bound on the probability that there exists a client oracle that accepts maliciously in the sense of Definition 3.10 (cf. Lemma 5.6).

Recall that we now assume that $\mathcal{A}$ outputs a triple (i, s, b') such that the oracle π_i^s 'accepts' with a unique partner oracle π_j^t, such that π_i^s has a matching conversation to π_j^t, as the game is aborted otherwise.

Game 2. The challenger in this game proceeds as before, but in addition guesses indices $(i^*, s^*) \xleftarrow{\$} [\phi] \times [\eta]$. It aborts and chooses b' at random, if the adversary outputs (i, s, b') with $(i, s) \neq (i^*, s^*)$. With probability $1/(\eta\phi)$ we have $(i, s) = (i^*, s^*)$, and thus

$$\mathsf{Adv}_1 \leq \eta\phi \cdot \mathsf{Adv}_2.$$

Note that in Game 2 we know that $\mathcal{A}$ will output (i^*, s^*, b'). Note also that $\pi_{i^*}^{s^*}$ has a unique 'partner' due to the previous game. In the sequel we denote with $\pi_{j^*}^{t^*}$ the unique oracle such that $\pi_{i^*}^{s^*}$ has a matching conversation to $\pi_{j^*}^{t^*}$, and say that $\pi_{j^*}^{t^*}$ is the *partner* of $\pi_{i^*}^{s^*}$.

Subsequently, we only consider the case that the adversary does not modify any of the messages of the pre-accept stage.

Due to proof of Lemma 5.6, we can subsequently only deal with adversaries that do not modify messages exchanged between client $\pi_{i^*}^{s^*}$ and server oracle. Then we can use all the game-hops of the previous proof to the position where both the server and the client oracle compute the keys used for the sLHAE scheme as uniformly random keys. In this way we end up in Game 6 of the previous proof.

Game 3. In this game we can directly plug-in the sLHAE challenge. We randomly decide to embed the sLHAE challenge key into $k_C^{(C)} = k_C^{(S)}$ or into $k_S^{(C)} = k_S^{(S)}$. With probability $\geq 1/2$ our choice is correct (i.e. the adversary attacks the sLHAE ciphertexts generated under this key). We have

$$\mathsf{Adv}_2 \leq 2\phi \cdot \mathsf{Adv}_3.$$

This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the Encrypt query of the sLHAE game. Similarly the Decrypt query is used to decrypt all the queries on behalf of the receiving oracle. If $\mathcal{A}$ outputs a triple (i^*, s^*, b'), then the challenger forwards b' to the sLHAE challenger. Otherwise it outputs a random bit. Since the challenger essentially relays all messages it is easy to see that an adversary $\mathcal{A}$ having advantage ϵ' yields an adversary against the sLHAE security of the encryption scheme with success probability at least $1/2 + \epsilon'$.

Since by assumption any adversary has advantage at most $\epsilon_{\mathsf{sLHAE}}$in breaking the sLHAE security of the symmetric encryption scheme, we have

$$\mathsf{Adv}_3 \leq 1/2 + \epsilon_{\mathsf{sLHAE}}.$$

Collecting probabilities we get that $\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + \eta\phi(\epsilon_{\mathsf{pke}} + 2\epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{sLHAE}})$.

$\square$

5.7 TLS-RSA with Mutual Authentication is ACCE Secure

Theorem 5.5. *Let μ be the output length of thePRF, ν be the length of the premaster secret, and λ be the length of the nonces r_C and r_S. Assume that* *the PRF is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the public key encryption scheme is $(t, \epsilon_{\mathsf{pke}}, q_{\mathsf{pke}})$-secure, the hash function is $(t, \epsilon_{\mathsf{H}})$-secure, the signature scheme is $(t, \epsilon_{\mathsf{sig}}, q_{\mathsf{sig}})$-secure, and the sLHAE scheme is $(t, \epsilon_{\mathsf{sLHAE}})$-secure.*

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the TLS-RSA Handshake protocol with mutual authentication in the sense of Definition 3.8 with $t \approx t'$ and $q_{\mathsf{pke}}, q_{\mathsf{sig}} \geq \eta$, $q_{\mathsf{prf}} \geq 2$, it holds that

$$\epsilon_{\mathsf{tls}} \leq 4\epsilon_{\mathsf{ca}} + \frac{(d\ell)^2}{2^{\lambda-2}} + 2\ell \cdot \epsilon_{\mathsf{sig}} + 16d\ell^2(2^{-\nu} + 2^{-\mu} + \epsilon_{\mathsf{H}}) + (16d\ell^2 + d\ell)(\epsilon_{\mathsf{pke}} + 2\epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{sLHAE}}).$$

We consider the following three types of adversaries.

Type 1. The adversary breaks the authentication property of Definition 3.8 and the first oracle that maliciously accepts is a **client** oracle (i.e. an oracle with $\rho = \mathsf{Client}$).

Type 2. The adversary breaks the authentication property of Definition 3.8 and the first oracle that maliciously accepts is a **server** oracle (i.e. an oracle with $\rho = \mathsf{Server}$).

Type 3. The adversary answers the encryption challenge correctly for sessions in which client and server have mutual authentication.

The proof is similar to the proof of Theorem 5.4 (for TLS-RSA with server-only authentication) given above. We can argue that no adversary of **Type 1** exists (except for some negligible error probability) because we have already shown in the proof of Theorem 5.4 that the client only accepts if the messages sent by the client *and* server have not been modified in transit, even without authentication of the client. The only additional message sent from the client to the server oracle in the mutual authentication setting is the Certificate Verify message m_9 which contains a fresh signature (that is also computed over the random nonces). However, we can argue that any modification to the signature on transit results in the client to not accept. This is because the `ServerFinished` message is computed over all previous messages, which would then be different for the client (which re-computes it for verification over the unmodified signature it sent) and the server (which computes it over the modified signature it received) oracle.

To prove that no adversary of **Type 2** exists (again except for some negligible error probability) we now exploit that the random nonces, contained in messages m_1 and m_2, and the encryption of the premaster secret, message m_8, are signed by the client. If the **Type 2** adversary modifies any of these messages, we can directly use the adversary to output a signature forgery. We can now substitute m_8 by the encryption of a truly random value under the assumption that the public key encryption system is CCA secure. As before, the adversary now cannot gain any information on pms from the ciphertext. Based on this modification and the security of the PRF we can in the next step substitute the master secret ms by a truly random value. With a truly random master secret we can now, again by the security of thePRF, substitute the `Finished` messages $\mathsf{fin}_C^{(S)}$ and $\mathsf{fin}_S^{(S)}$ and the encryption keys $k_C^{(S)}$ and $k_S^{(S)}$ with truly random values. We can now embed the sLHAE key into $k_C^{(S)}$. Since the nonces and m_8 are protected from adversarial modifications, we must have that $k_C^{(S)} = k_C^{(C)}$. Now there are two cases. If the adversary modifies any of the messages m_1 to m_{10}, the `ServerFinished` messages $\mathsf{fin}_S^{(S)}$ and $\mathsf{fin}_S^{(C)}$ can, exploiting the security of the hash function and thePRF, be substituted by uniformly random values which are *distinct* with probability $1 - 2^{-\mu}$. The message m_{11} output by the client oracle thus cannot help the adversary to make the server oracle accept. The adversary must compute an encryption of $\mathsf{fin}_C^{(S)}$ on its own. Otherwise, in case m_{11} (`ClientFinished`) is the only message modified by the adversary, they will be substituted by the same random value, i.e. $\mathsf{fin}_C^{(S)} = \mathsf{fin}_C^{(C)}$. In such a case, any adversary that makes the server oracle accept maliciously has to compute a new encryption m'_{11} of $\mathsf{fin}_C^{(S)} = \mathsf{fin}_C^{(C)}$. In both cases we can use the $\mathsf{Encrypt}$ query from the sLHAE security game to generate m_{11}. Therefore, to make the server accept, the adversary must now break the security of the sLHAE scheme.

To prove that no adversary of **Type 3** exists (again except for some negligible error probability) we exploit that no message up to, but not including, m_{13} has been modified on transit. Then the keys for the sLHAE scheme can be substituted by uniformly random values and we can directly plugin the sLHAE challenge in one of the encryption keys into either $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the $\mathsf{Encrypt}$ query of the sLHAE game. Similarly the $\mathsf{Decrypt}$ query is used to decrypt all these queries on behalf of the receiving oracle. An adversary that solves the encryption challenge can so directly be used to break the security of the sLHAE scheme.

5.8 TLS-SDH with Mutual Authentication is ACCE Secure

In this section we show that TLS-SDH with mutual authentication and client static Diffie-Hellman keys is ACCE secure. We only consider the case where the client certificate contains a static DH key, in contrast to the case where the client only provides a certificate containing a public key of a digital signature scheme and chooses fresh DH shares for each connection. We think that the first case is more interesting as it allows to reduce the security of the protocol to the standard DDH assumption, in contrast to the second scenario, which would again require a reduction to the (non-standard) PRF-ODH assumption.

Theorem 5.6. *Let μ be the output length of the PRF and let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the hash function is $(t, \epsilon_{\mathsf{H}})$-secure, the DDH-problem is $(t, \epsilon_{\mathsf{ddh}})$-hard in the group G used to compute the TLS premaster secret, and the sLHAE scheme is $(t, \epsilon_{\mathsf{sLHAE}})$-secure. Assume that no party will accept in a TLS handshake, if the public key of the communication partner is equal to the public key of that party.*[29]

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the static Diffie-Hellman TLS Handshake protocol with mutual authentication in the sense of Definition 3.8 with $t \approx t'$ and $q_{\mathsf{prf}} \geq 2$, it holds that

$$\begin{aligned}\epsilon_{\mathsf{tls}} \quad &\leq \quad 4\frac{(\eta\phi)^2}{2^\lambda} + 4\phi^2 \cdot \epsilon_{\mathsf{ddh}} + 12\eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu}) \\ &+ \quad \phi^2(\epsilon_{\mathsf{ddh}} + d(2\epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{sLHAE}})) \\ &\leq \quad \frac{(\eta\phi)^2}{2^{\lambda-2}} + 5\phi^2 \cdot \epsilon_{\mathsf{ddh}} + 26\eta\phi^2 \cdot \epsilon_{\mathsf{prf}} + 14\eta\phi^2 \cdot \epsilon_{\mathsf{sLHAE}} + 12\eta\phi^2 \cdot \epsilon_{\mathsf{H}}.\end{aligned}$$

We prove Theorem 5.6 by proving two lemmas. Lemma 5.8 bounds the probability ϵ_{auth} that an adversary succeeds in making any oracle accept maliciously. Lemma 5.11 bounds the probability ϵ_{enc} that an adversary does not succeed in making an oracle accept maliciously, but which answers the encryption challenge correctly. Then we have

$$\epsilon_{\mathsf{tls}} \leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}}.$$

Lemma 5.8. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle π_i^s that accepts maliciously is at most*

$$\epsilon_{\mathsf{auth}} \leq \frac{(\eta\phi)^2}{2^{\lambda-1}} + 2\phi^2 \cdot \epsilon_{\mathsf{ddh}} + 6\eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu}),$$

where all quantities are defined as stated in Theorem 5.6.

Note that $\epsilon_{\mathsf{auth}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}$, where $\epsilon_{\mathsf{client}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Client}$ that accepts maliciously in the sense of Definition 3.8, and $\epsilon_{\mathsf{server}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Server}$ that accepts maliciously. We claim that

$$\begin{aligned}\epsilon_{\mathsf{client}} &\leq \frac{(\eta\phi)^2}{2^\lambda} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + 3\eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu}), \text{ and} \\ \epsilon_{\mathsf{server}} &\leq \frac{(\eta\phi)^2}{2^\lambda} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + 3\eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu}),\end{aligned}$$

and thus

$$\begin{aligned}\epsilon_{\mathsf{auth}} &\leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} \\ &\leq 2 \cdot \left(\frac{(\eta\phi)^2}{2^\lambda} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + 3\eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu})\right).\end{aligned}$$

[29]This can easily be realized technically.

We split up the proof of Lemma 5.8 in two separate lemmas, Lemma 5.9 and Lemma 5.10, which we give (and prove) in the following sections.

5.8.1 Client Authentication

Lemma 5.9. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists any π_i^s with $\rho = \mathsf{Client}$ that accepts maliciously with respect to Definition 3.8 is at most*

$$\epsilon_{\mathsf{client}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + 3\eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu})$$

where all quantities are defined as stated in Theorem 5.6.

Proof. The proof again proceeds in a *sequence of games.* As before we use different subcases to organize the proof. When denoting the different subcases we stay consistent with the general proof outline. We end up in a final game, where no adversary can break the security of the protocol. Let $\mathsf{break}_\delta^{(3)}$ be the event that occurs when a client oracle accepts maliciously in the sense of Definition 3.8 in Game δ.

Game 0. This game equals the ACCE security experiment described in Section 3.2.2. Thus, for some $\epsilon_{\mathsf{client}}$ we have

$$\Pr[\mathsf{break}_0^{(3)}] = \epsilon_{\mathsf{client}}.$$

Game 1. In this game we add an abort rule. The challenger aborts, if there exists any oracle π_i^s that chooses a random nonce r_C or r_S which is not unique. More precisely, the game is aborted if the adversary ever makes a first Send query to an oracle π_i^s, and the oracle replies with random nonce r_C or r_S such that there exists some other oracle $\pi_{i'}^{s'}$ which has previously sampled the same nonce.

In total less than $\eta\phi$ nonces r_C and r_S are sampled, each uniformly random from $\{0,1\}^\lambda$. Thus, the probability that a collision occurs is bounded by $(\eta\phi)^2 2^{-\lambda}$, which implies

$$\Pr[\mathsf{break}_0^{(3)}] \leq \Pr[\mathsf{break}_1^{(3)}] + \frac{(\eta\phi)^2}{2^\lambda}.$$

Note that now each oracle has a unique nonce r_C or r_S, which will be used to compute the master secret.

Game 2. We now try to guess two distinct indices (i^*, j^*) with $i^*, j^* \in [\phi]$, such that the first oracle π_i^s that will accept maliciously with $\rho = \mathsf{Client}$ belongs to party P_{i^*}, i.e. $i = i^*$ and has communication partner $\Pi = j$ with $j = j^*$. If the first oracle to accept maliciously is π_i^s with $i \neq i^*$ or π_i^s has $\Pi = j$ with $j \neq j^*$, then the challenger aborts the game. Since there are at most ϕ parties, the probability to guess correctly, i.e. we have $(i,j) = (i^*, j^*)$, is $\geq 1/\phi^2$, and thus

$$\Pr[\mathsf{break}_1^{(3)}] \leq \phi^2 \cdot \Pr[\mathsf{break}_2^{(3)}].$$

Game 3. In this game we want to substitute the premaster secret *pms* computed by all oracles $\pi_{i^*}^s$ and their partner oracles $\pi_{j^*}^t$ for $s,t \in [\eta]$ by a randomly chosen value. We can exploit the fact that the premaster secret will be the same for all sessions of parties P_{i^*} and P_{j^*} in this setting (static Diffie-Hellman keys and mutual authentication) and that we know the party P_{i^*} holding the oracle that will accept maliciously first and its communication partner P_{j^*} due to the previous game.

The challenger in this game proceeds as before but replaces the Diffie-Hellman public keys in the certificates for P_{i^*} and P_{j^*} with the Diffie Hellman values (g^a, g^b) from a DDH challenge tuple (g, g^a, g^b, g^c). Observe that g^a and g^b are distributed *exactly* as the public keys that have originally been chosen by the challenger. The challenger then proceeds as follows. Whenever two oracles of P_{i^*} and P_{j^*} communicate with each other, the premaster secret will be set to g^c, otherwise the *pms* will be computed honestly and then set by the challenger. The challenger can do so, since it knows the secret keys for all public keys. It knows the corresponding secret Diffie-Hellman key of the communication partner of P_{i^*} or P_{j^*} and can so compute the premaster secret.

If $g^c = g^{ab}$ then we are in the previous game, whereas if $g^c \neq g^{ab}$ we are in the current game. It follows that we can use an adversary that can distinguish between this game and the previous game to break DDH.

We have

$$\Pr[\mathsf{break}_2^{(3)}] \leq \Pr[\mathsf{break}_3^{(3)}] + \epsilon_{\mathsf{ddh}}.$$

Game 4. We now try to guess index s^* with $s^* \in [\eta]$, such that the first oracle of P_{i^*} that will accept maliciously is $\pi_{i^*}^{s^*}$. If the first oracle to accept maliciously is $\pi_{i^*}^s$ with $s \neq s^*$, then the challenger aborts the game. Since there are at most η oracles for each party, the probability to guess correctly, i.e. we have $s = s^*$, is $\geq 1/\eta$, and thus

$$\Pr[\mathsf{break}_3^{(3)}] \leq \eta \cdot \Pr[\mathsf{break}_4^{(3)}].$$

Note that in this game the adversary can only break the security of the protocol, if oracle $\pi_{i^*}^{s^*}$ is the first client oracle that ‘accepts’ maliciously with $\Pi = j^*$, as otherwise the game is aborted.

We now consider two cases. Either the adversary does not modify any of the messages r_C or r_S in transit (**Case 1.1**) or it does so (**Case 1.2**).

Case 1.1:

Game 5. In this game, we substitute the master secret generated by the client oracle and the server oracle by a truly random value. Any adversary that can distinguish between this and the previous game can be used to break the security of the PRF.

To show this, observe that since *pms* is drawn uniformly at random due to the previous game, we can directly embed the security challenge of the PRF challenger: we use a PRF query to compute the master secret. If the answer has been computed with the PRF we are in the previous game. If the answer is a truly random value, we are in the current game. So any adversary that distinguishes this game from the previous can directly be used to break the security of the PRF.

We have,

$$\Pr[\mathsf{break}_4^{(3)}] \leq \Pr[\mathsf{break}_5^{(3)}] + \epsilon_{\mathsf{prf}}.$$

We are now in a game where the master secret of client and server oracle are equal and truly random, and where neither r_S nor r_C have been modified by the adversary. At this point we define two subcases. Either the adversary does not modify any of the (remaining) messages m_1 to m_{11}, **Case 1.1.1**, or it does, **Case 1.1.2**.

Case 1.1.1:

Game 6. In this game, the adversary must by definition modify m_{13} to make the client oracle accept maliciously. However, we first substitute fin_S and the encryption keys generated by the client oracle and the server oracle by truly random values. Any adversary that can distinguish between this and the previous game can be used to break the security of the PRF. To show this, observe that since *ms* is drawn uniformly at random we can directly embed the security challenge of the PRF challenger: we use a PRF query to compute fin_S. If the answer has been computed with the PRF we are in the previous game. If the answer is a truly random value, we are in the current game. So any adversary that distinguishes this game from the previous can directly be used to break the security of the PRF. We have,

$$\Pr[\mathsf{break}_5^{(3)}] \leq \Pr[\mathsf{break}_6^{(3)}] + \epsilon_{\mathsf{prf}}.$$

Game 7. In this game we show that any adversary that wins can be used to break the security of the encryption system. We know that the adversary outputs a message $m'_{12} \neq m_{13}$ which makes the client adversary accept. Since fin_S is a truly random value this can only happen if m'_{12} is also a valid encryption of m_{13}. However, this encryption can be used to break the security of the sLHAE encryption scheme as follows: we embed the sLHAE challenge key into $k_S^{(C)} = k_S^{(S)}$ and use the $\mathsf{Encrypt}$ query to compute m_{13}. Now the adversary outputs a new ciphertext m'_{12} that has not been generated by $\mathsf{Encrypt}$ which breaks the security of the sLHAE game.

$$\Pr[\mathsf{break}_6^{(3)}] \leq \epsilon_{\mathsf{sLHAE}}.$$

Collecting probabilities we get that

$$\epsilon_{\mathsf{client}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + \eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}).$$

In the following, we will only explain the differences to the proof of **Case 1.1.1**.

<u>**Case 1.1.2:**</u>

In this case, the adversary has modified any of the remaining messages m_1 to m_{11} on transit. We first substitute the output values of the PRF when keyed with ms by truly random values. Since the encryption keys do only depend on r_C and r_S we substitute the values that have been computed by the client oracle and the server oracle by the same random keys. However, the computation of fin_S does depend on the messages m_1 to m_{11}. Since m_1 to m_{11} are distinct, so must be their hash values, as otherwise we can use the adversary to break the security of the hash function. The distinct hash values are input to the PRF and we can substitute $\mathsf{fin}_S^{(S)}$, $\mathsf{fin}_S^{(C)}$ and $k_S^{(C)} = k_S^{(S)}$ by independently drawn random values. If $\mathsf{fin}_S^{(C)} = \mathsf{fin}_S^{(S)}$, which happens with probability $2^{-\mu}$, we abort. Otherwise ($\mathsf{fin}_S^{(C)} \neq \mathsf{fin}_S^{(S)}$), the server oracle will not compute a message m_{13} that will make the client oracle accept. To make the client oracle accept the adversary must compute m'_{12} to be an encryption of $\mathsf{fin}_S^{(C)}$.

Now we again embed the sLHAE challenge in $k_S^{(C)} = k_S^{(S)}$. Then the adversary outputs a new ciphertext m'_{12} that has not been generated by $\mathsf{Encrypt}$ which breaks the security of the sLHAE game.

Collecting probabilities we get that

$$\epsilon_{\mathsf{client}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + \eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu}).$$

<u>**Case 1.2:**</u>

In this case the adversary has modified either r_C or r_S in transit. Since these values are input to the PRF when computing the master secret we can by the security of the PRF substitute $ms^{(S)}$ and $ms^{(C)}$ by independently drawn random values. With probability $2^{-\mu}$ these values are equal and we abort. Otherwise we proceed as follows. Similar to before, we can by the security of the PRF show that $k_S^{(C)}$ can be substituted with a uniformly random key that is drawn after the client oracle receives the last message m_{13}. Thus $k_S^{(C)}$ is independent of the server oracle and the adversary. Since $k_S^{(C)}$ is a uniformly random value independent of any value computed by the server oracle and unknown to the adversary we can as before embed the PRF challenge in $k_S^{(C)}$.

Thus, if the adversary outputs a message that makes the client accept, it directly breaks the security of the sLHAE game.

Collecting probabilities we get that

$$\epsilon_{\mathsf{client}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + \eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu}).$$

□

5.8.2 Server Authentication

Lemma 5.10. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists a π_i^s with $\rho = \mathsf{Server}$ that accepts maliciously with respect to Definition 3.8 is at most*

$$\epsilon_{\mathsf{server}} \leq \frac{(\eta\phi)^2}{2^\lambda} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + 3\eta\phi^2 \cdot (2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + 2^{-\mu})$$

where all quantities are defined as stated in Theorem 5.6.

Proof. Due to the symmetry of the handshake the proof is essentially that of Lemma 5.9. The only difference is that now the message m_{13} can be modified in 'transit'. However by the definition of matching conversation this does not result in a successfull adversary. □

5.8.3 Indistinguishability of Ciphertexts

Lemma 5.11. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the encryption-challenge correctly is at most $1/2 + \epsilon_{\mathsf{enc}}$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + 2\eta\phi^2 \cdot (\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}).$$

where ϵ_{auth} is an upper bound on the probability that there exists a client oracle that accepts maliciously in the sense of Definition 3.8 (Lemmas 5.9) and all other quantities are defined as stated in Theorem 5.6.

PROOF. Assume without loss of generality that the $\mathcal{A}$ always outputs a triple such that all conditions in Property 2 of Definition 3.8 are satisfied. Let $\mathsf{break}_\delta^{(5)}$ denote the event that $b' = b_i^s$ in Game δ, where b_i^s is the random bit sampled by the oracle π_i^s, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_\delta := \Pr[\mathsf{break}_\delta^{(5)}] - 1/2$ denote the *advantage* of $\mathcal{A}$ in Game δ.

Consider the following sequence of games.

Game 0. This game equals the ACCE security experiment described in Section 3.2.2. For some ϵ_{enc} we have

$$\Pr[\mathsf{break}_0^{(4)}] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = \frac{1}{2} + \mathsf{Adv}_0.$$

Game 1. The challenger in this game proceeds as before, but it aborts and chooses b' uniformly random, if there exists any oracle that accepts maliciously in the sense of the ACCE definition.

Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}},$$

where ϵ_{auth} an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 3.8 (cf. Lemma 5.9).

Recall that we now assume that $\mathcal{A}$ outputs a triple (i, s, b') such that the oracle π_i^s 'accepts' with a unique partner oracle π_j^t, such that π_i^s has a matching conversation to π_j^t, as the game is aborted otherwise.

Game 2. We now try to guess indices (i^*, j^*) with $i^*, j^* \in [\phi]$, such that when the Adversary terminates and outputs (i, s, b') it holds that $i = i^*$ and oracle $\pi_{i^*}^s$ has communication partner $\Pi = j$ with $j = j^*$.

If the Adversary terminates and outputs (i, s, b'), and $i \neq i^*$ or π_i^s has communication partner $\Pi = j$ with $j \neq j^*$, the challenger aborts the game. Since there are at most ϕ parties, the probability to guess correctly, i.e. we have $(i, j) = (i^*, j^*)$, is $\geq 1/\phi^2$, and thus

$$\Pr[\mathsf{break}_1^{(3)}] \leq \phi^2 \cdot \Pr[\mathsf{break}_2^{(3)}].$$

Game 3. With the same arguments as in Game 3 of the proof of Lemma 5.9 we exchange the premaster secret *pms* computed in all sessions between parties P_{i^*} and P_{j^*} with a randomly chosen value.

Thus we have

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{ddh}}.$$

Game 4. We now try to guess index s^* with $s^* \in [\eta]$, such that when the Adversary terminates and outputs (i, s, b') it holds that $s = s^*$. Since there are at most η oracles for each party, the probability to guess correctly, i.e. we have $s = s^*$, is $\geq 1/d$, and thus

$$\mathsf{Adv}_3 \leq d \cdot \mathsf{Adv}_4.$$

Note that in this game the adversary can only break the security of the protocol, if oracle $\pi_{i^*}^{s^*}$ is the first client oracle that 'accepts' maliciously with $\Pi = j^*$, as otherwise the game is aborted.

Game 5. In this game we exchange the encryption keys computed by $\pi_{i^*}^{s^*}$ (and the partner oracle having a matching conversation with π_i^s) with uniformly random keys. To show this, we, for reasons of simplicity, will make several changes at the same time and argue for each, that the adversary cannot detect this change.

Master secret. With the same arguments as in the previous proof we exchange the master secret with a uniformly random value.

Encryption Keys. With the same arguments as in the previous proof we exchange the encryption keys with a uniformly random value. Observe that since ms is drawn uniformly at random we can again directly embed the security challenge of the PRF challenger: we use a PRF query to compute the encryption keys as

$$K_{\mathsf{enc}}^{C\to S}||K_{\mathsf{enc}}^{S\to C}||K_{\mathsf{mac}}^{C\to S}||K_{\mathsf{mac}}^{S\to C} := \mathsf{PRF}(ms, label_2||r_C||r_S)$$

If the adversary can detect whether the keys have been computed with the PRF or if the answer is a truly random value, we can directly use this adversary to break the security of the PRF.

Summarizing these changes we get that

$$\mathsf{Adv}_4 \leq \mathsf{Adv}_5 + 2 \cdot \epsilon_{\mathsf{prf}}.$$

Game 6. In this game we can directly plug-in the sLHAE challenge into either $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. With probability $\geq 1/2$ our random choice is correct (i.e. the adversary attacks the sLHAE ciphertexts generated under this key). We have

$$\mathsf{Adv}_5 \leq 2\phi \cdot \mathsf{Adv}_6.$$

This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the Encrypt query of the sLHAE game. Similarly the Decrypt query is used to decrypt all these queries on behalf of the receiving oracle. If $\mathcal{A}$ outputs a triple (i^*, s^*, b'), then the challenger forwards

b' to the sLHAE challenger. Otherwise it outputs a random bit. Since the challenger essentially relays all messages it is easy to see that an adversary $\mathcal{A}$ having advantage ϵ' yields an adversary against the sLHAE security of the encryption scheme with success probability at least $1/2 + \epsilon'$.

Since by assumption any adversary has advantage at most $\epsilon_{\mathsf{sLHAE}}$ in breaking the sLHAE security of the symmetric encryption scheme, we have

$$\mathsf{Adv}_6 \leq 1/2 + \epsilon_{\mathsf{sLHAE}}.$$

Collecting probabilities we get that

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + \phi^2 \cdot \epsilon_{\mathsf{ddh}} + 2\eta\phi^2 \cdot (\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}).$$

□

5.9 TLS-SDH with Server-Only Authentication is ACCE Secure

Theorem 5.7. *Let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the sLHAE scheme is $(t, \epsilon_{\mathsf{sLHAE}})$-secure, and the Strong PRF-ODH-problem is $(t, \epsilon_{\mathsf{prfodh}}, q_{\mathsf{prfodh}})$-hard with respect to G and* PRF*.*

Then for any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the TLS-SDH Handshake protocol with server-only authentication in the sense of Definition 3.10 with $t \approx t'$ and $q_{\mathsf{prf}} \geq 1$, and $q_{\mathsf{prfodh}} \geq \eta$, it holds that

$$\epsilon_{\mathsf{tls}} \leq 2 \cdot \frac{(\eta\phi)^2}{2^{\lambda-1}} + 8(\eta\phi)^2 \cdot (\epsilon_{\mathsf{H}} + 2^{-\mu}) + 9(\eta\phi)^2 \cdot (\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + 2\epsilon_{\mathsf{sLHAE}}) .$$

To prove the security of TLS-SDH with server-only authentication we again consider the two types of adversaries against server-only authenticated TLS as described in Section 5.5.

To prove that no adversary of **Type 1** exists (again except for some negligible error probability) we first proceed as in the proof of TLS-DHE with server-only authentication. We first exclude modifications of the server share which is contained as a long-term key in the server certificate. An adversary able to output a valid signature (verifiable under the key used by the simulator to generate certificates) for a new DH public key could directly be used to forge a certificate. We then guess the server oracle π_j^t and its partner (client) oracle π_i^s with probability $1/(\eta\phi)^2$.

Now we again follow the outline of the proof of TLS-DHE with server-only authentication and consider two cases. We have the same cases and subcases as before. Either the adversary does modify the client share (**Case 2**) or not (**Case 1**). Either the adversary does not modify any of the nonces in transit (**Case 1.1**), or it does (**Case 1.2**) so. Either the adversary does not modify any of the remaining messages m_1 to m_{11} in transit (**Case 1.1.1**), or it does (**Case 1.1.2**) so.

In **Case 1.2**, **Case 1.1.1**, and **Case 1.1.2** we exploit the Strong PRF-ODH assumption to show that the master secret is indistinguishable from random. As before, in **Case 2** the master secret $ms^{(C)}$ computed by the client oracle is indistinguishable from random and with overwhelming probability distinct from $ms^{(S)}$.

To show that the master secret is indistinguishable from random in the subcases of **Case 1**, we need to exploit the Strong PRF-ODH problem. We need the enhanced (as compared to the original PRF-ODH assumption) capabilities of the adversary in the Strong PRF-ODH game because any **Type 1** adversary can try to engage in *several* (q_{prfodh}) sessions with the *same server-oracle* and the *same client share* (only using distinct nonces). Thus, the attack capabilities granted in the original PRF-ODH game do not suffice to correctly simulate the behaviour of the oracles. In **Case 1** we substitute the master secret that is computed between oracles π_i^s and π_j^t by a truly random value. Any adversary that can recognize our modification can be used to break the Strong PRF-ODH assumption. Again consider that we send the concatenation of the first label and the random nonces to the PRF-ODH challenger, who responds with z_b, g^u and g^v. At P_j we first embed the g^v share from the PRF-ODH challenge in the certificate (without changing the distribution, since g^v is random). Also we now use the PRF-ODH challenger to simulate that oracles of P_j have access to the secret key v. In the communication between oracles π_i^s and π_j^t, we set g^u to be the Diffie-Hellman share of π_i^s and use z_b as the master secret for the client and server oracle. Again, g^u is distributed exactly as before. We can now simulate all adversarial queries to P_j including those where the adversary re-uses g^u as the client-share. This is because the nonce generated by the server is always fresh and thus the message input to the Strong PRF-ODH oracle is distinct from the values used in the first step of the Strong PRF-ODH security game. Now, if z_b is the real output of PRF this corresponds to the situation where ms is computed honestly, if it is random, ms is random too. So under the PRF-ODH assumption no adversary can notice our substitution of the master secret with a random value in **Case 1.1.1**, **Case 1.1.2**, and **Case 1.2**. The remaining part of the proof is again similar to that of TLS-DHE with server-only authentication.

The **Cases 2, 1.2** and **1.1.2** can all be reduced to the security of the sLHAE scheme. We embed the sLHAE challenge in the key $k_S^{(C)}$ of the client oracle that is used to decrypt server messages. Since $\mathsf{fin}_S^{(S)}$ is independent from $\mathsf{fin}_S^{(C)}$, the adversary cannot use the output of the server oracle to compute m_{13} such that the client accepts. Thus it has to compute this message on its own. However, this breaks the sLHAE assumption. In **Case 1.1.1** the adversary has to compute a new ciphertext on the same message $\mathsf{fin}_S^{(S)} = \mathsf{fin}_S^{(C)}$ to make the client accept. However this also breaks the security of the sLHAE scheme.

To prove that no adversary of **Type 2** exists (again except for some negligible error probability), we proceed similar to the previous proof of TLS-DHE with server-only authentication. We first exploit that the client will not accept if any of the messages is modified. Next, we can directly plug-in the sLHAE challenge into either $k_C^{(C)} = k_C^{(S)}$ or $k_S^{(C)} = k_S^{(S)}$. This means that for the corresponding key every ciphertext generated by the client or server oracle has been produced using the Encrypt query of the sLHAE game. Similarly the Decrypt query is used to decrypt all these queries on behalf of the receiving oracle. So any successful adversary breaks the security of the sLHAE game.

6 On the Security of TLS Renegotiation

This chapter includes results from joint work with Florian Giesen and Douglas Stebila, published at CCS'13 [GKS13, GKS12]. In this chapter we will analyze the TLS renegotiation protocol as described in the Introduction in Section 6.1 under the security definitions given in Section 3.2.3.

We will first discuss how unmodified TLS renegotiation, without the SCSV/RIE countermeasures proposed in RFC 5746 [RRDO10] and described in short in Section 6.1, fits into our model, and show how the attack of Ray and Dispensa is captured in the model in Section 6.2. Then we analyze the security of TLS with the SCSV/RIE countermeasures. We first see, in Sections 6.3 and 6.4, that the SCSV/RIE countermeasures are not enough to prove that TLS satisfies our strongest notion, a secure renegotiable ACCE (Definition 3.15). Our goal, then, will be to show that TLS with the SCSV/RIE countermeasures is both a secure multi-phase ACCE in Section 6.6 and a weakly secure renegotiable ACCE in Section 6.7. Ideally, we would do so generically, with a result saying something like 'If a TLS ciphersuite is a secure ACCE, then that TLS ciphersuite with SCSV/RIE is a weakly secure renegotiable ACCE'.

However, we cannot do so generically since the protocol is *modified* to include the countermeasure values in the `ClientHello` and `ServerHello` messages, and thus we cannot make use of the ACCE security of the particular TLS ciphersuite in a black-box way. Moreover, we must ensure that revealing the `Finished` values from the previous handshake does not impact its security. Although these barriers prevent a generic black-box result, we stress that a white-box result can be achieved that only requires small changes. In Section 6.5 we describe an intermediate result necessary to achieve above mentioned goals.

Summarized Course of Action We will provide a sequence of definitions and results that justifies the security of the SCSV/RIE countermeasure. Figure 6.1 summarizes our approach.

1. Define an extended ACCE security model, called *tagged-ACCE-fin* specific to TLS, in which the adversary can reveal `Finished` messages after the handshake completes and supply tags to be used in extensions.

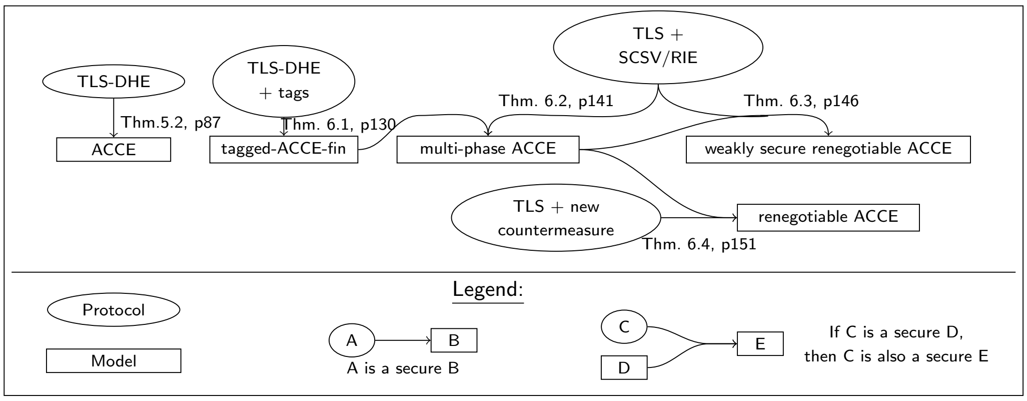

Figure 6.1: Summary of the results on TLS and renegotiation

2. Define *tagged* TLS as a modification of a standard ciphersuite in which arbitrary opaque data can be placed in an extension field in the `ClientHello` and `ServerHello` messages.

3. Explain how the existing proof of that TLS-DHE is ACCE secure can be modified in a very minor way to show that tagged TLS-DHE is tagged-ACCE-fin-secure.

4. Show that, if a tagged TLS ciphersuite is tagged-ACCE-fin secure, then that TLS ciphersuite with SCSV/RIE is a secure multi-phase ACCE.

5. Show that, if a TLS ciphersuite with SCSV/RIE is a secure multi-phase ACCE, then it is also a weakly secure renegotiable ACCE.

Combined, these results yield (a) a general result justifying the security of the SCSV/RIE countermeasure, and (b) that TLS-DHE with SCSV/RIE countermeasures is a weakly secure renegotiable ACCE.

Remark 19. Note that a *tagged TLS* protocol even allows us to prove the security of TLS-DHE under the DDH assumption, instead of requiring the much stronger PRF-ODH assumption.

We finally also provide a new variant of TLS renegotiation in Section 6.8, that meets even our strongest security notions.

6.1 Renegotiation Protocols and TLS Renegotiation

A *renegotiation* protocol is a protocol, which allows two parties to either (a) obtain a fresh session key, (b) change cryptographic parameters, or (c) change authentication credentials. For example, if a client needs to authenticate using a client certificate but wishes to not reveal his identity over a public channel, he could first authenticate anonymously (or with pseudonymous credentials), then renegotiate using his real certificate; since the renegotiation messages are transmitted within the existing record layer, the transmission of his certificate is encrypted, and thus he obtains privacy for his identity. We will examine TLS renegotiation in detail, especially in light of previously identified practical attacks related to TLS renegotiation.

Despite the utility of renegotiation in real-world protocols - beyond TLS, renegotiation, rekeying, or reauthentication is also used in the Secure Shell (SSH) protocol, Internet Key Exchange version 2, the Tor anonymity protocol, and others - there has been almost no research in the literature on the security of protocols involving renegotiation, with the exception of a brief note on the TLS renegotiation attack by Farrell [Far10] and the recent thesis of Gelashvili [Gel12], which uses the Scyther tool to automatically identify the TLS renegotiation attack. Rekeying has been studied in the context of group key agreement for applications such as mobile ad hoc networks, but without reference to AKE security models.

An Attack against TLS Renegotiation

All versions of TLS [DA99, DR06, DR08], and SSL v3 [FKK11] before it, support optional renegotiation. After the initial handshake is completed and secure communication begins in the record layer, either party can request renegotiation. The client can request renegotiation by sending a new `ClientHello` message in the current record layer (i.e. encrypted under the current session key); the server can request renegotiation by sending a `HelloRequest` message in the record layer, which triggers the client to send a new `ClientHello` message.

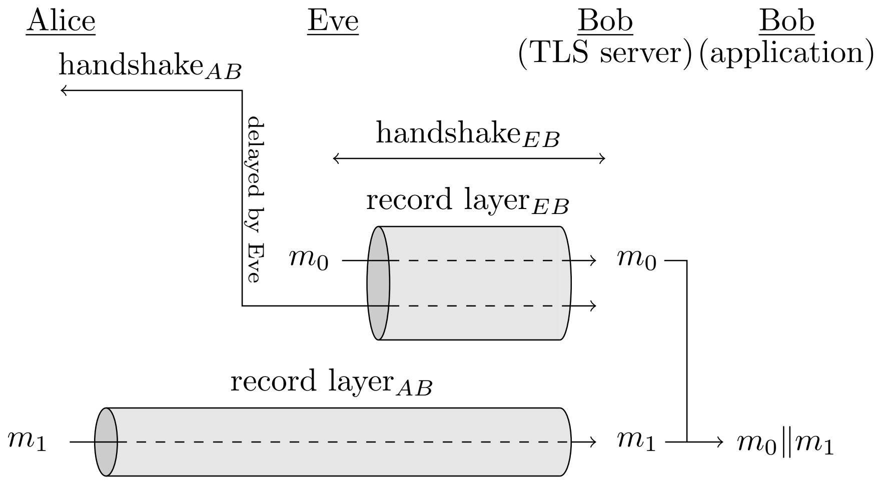

Figure 6.2: Ray and Dispensa's man-in-the-middle renegotiation attack on TLS-reliant applications

In November 2009, Ray and Dispensa [RD09] described a Man-In-The-Middle attack that exploits how certain TLS-reliant applications - such as Hypertext Transfer Protocol (HTTP) over TLS [Res00] - process data across renegotiations. The attack is shown in Figure 6.2. The attacker Eve observes Alice attempting to establish a TLS session with Bob. Eve delays Alice's initial `ClientHello` and instead establishes her own TLS session with Bob and transmits a message m_0 over that record layer. Then Eve passes Alice's initial `ClientHello` to Bob over the Eve-Bob record layer. Bob views this as a valid renegotiation and responds accordingly; Eve relays the handshake messages between Alice and Bob, who eventually establish a new record layer to which Eve has no access. Alice then transmits a message m_1 over the Alice-Bob record layer.

Strictly speaking this is not an attack on TLS, but on how some applications process TLS-protected data. It results from some applications, including Hypertext Transfer Protocol Secure (HTTPS) [RD09] and SMTP over TLS (SMTPS) [Zol09], concatenating m_0 and m_1 and treating them as coming from the same party in the same context. For example, if Eve sends the HTTP request m_0 and Alice sends the HTTP request m_1, where

$$m_0 = \text{‘}\texttt{GET /orderPizza?deliverTo=123-Fake-St} \hookleftarrow \texttt{X-Ignore-This: }\text{’}$$
$$m_1 = \text{‘}\texttt{GET /orderPizza?deliverTo=456-Real-St} \hookleftarrow \texttt{Cookie: Account=111A2B}\text{’}$$

(where $\hookleftarrow$ denotes new-line character), then the concatenated request (across multiple lines for readability) is

$$\begin{aligned} m_0 \| m_1 = \text{‘}&\texttt{GET /orderPizza?deliverTo=123-Fake-St} \hookleftarrow \\ &\texttt{X-Ignore-This: GET /orderPizza?deliverTo=456-Real-St} \hookleftarrow \\ &\texttt{Cookie: Account=111A2B}\text{’} \end{aligned}$$

The '`X-Ignore-This:`' prefix is an invalid HTTP header, and since this header, without a new line character, is concatenated with the first line of Alice's request, so this line is ignored. However, the following line, Alice's account cookie, is still processed. Eve is able to have the pizza delivered to herself but paid for by Alice.

It should be noted that Ray and Dispensa's attack works for both server-only authentication and mutual authentication modes of TLS: the use of client certificates in general does not prevent the attack [RD09, Zol09].

Countermeasures Added to TLS Renegotiation

The immediate recommendation due to this attack was to disable renegotiation except in cases where it was essential. Subsequently, the Internet Engineering Task Force (IETF) TLS working group developed RFC 5746 [RRDO10] to provide countermeasures to this attack, with the goal of applicability to all versions of TLS and SSL 3.0. Two countermeasures were standardized: the *Signalling Ciphersuite Value (SCSV)* and the *Renegotiation Information Extension (RIE)*. These were adopted by major TLS implementation providers and web browsers and servers, including Apache, Apple, Google, Microsoft, Mozilla, and OpenSSL. In RIE, the parties include the key confirmation value (i.e. the `Finished` message) from the previous handshake in a `ClientHello`/`ServerHello` extension [BWNH+03], demonstrating they have the same view of the previous handshake, or a distinguished null value if not renegotiation. The verification data extracted from the `ClientFinished` message is denoted with `client_verify_data`, the data extracted from `ServerFinished` with `server_verify_data`. SCSV is a slight modification that is more compatible with buggy implementations.

Renegotiation Information Extension (RIE). This countermeasure essentially provides *handshake recognition*, confirming that when renegotiating both parties have the same view of the previous handshake. With this countermeasure, each client or server always includes a renegotiation information extension in its respective `ClientHello` or `ServerHello` message. This extension contains one of three values. If the party is not renegotiating, then it includes a fixed 'empty' string which denotes that the party supports and understands the renegotiation extension, and the party is in fact not renegotiating. If the party is renegotiating, then it includes the handshake/key confirmation value from the previous handshake: the client sends the previous `client_verify_data` value while the server sends the concatenation of the previous `client_verify_data` and `server_verify_data` values. Intuitively, by including the verify data from the previous handshake, the parties can be assured that they have the same view of the previous handshake, and thus the attack in Figure 6.2 is avoided.

Signalling Ciphersuite Value (SCSV). SCSV was designed to avoid interoperability problems with TLS 1.0 and SSL 3.0 implementations that did not gracefully ignore extension data at the end of `ClientHello` and `ServerHello` messages. With SCSV, the client uses an alternative method in its initial handshake - an extra, fixed, distinguished ciphersuite value (byte codes `0x00,0xFF`) including in its ciphersuite list - to indicate that it knows how to securely renegotiate. Old servers will ignore this extra value; new servers will recognize that the client supports secure renegotiation, and the server will use the RIE in the remainder of the session. In other words, the only difference between SCSV and RIE is in the `ClientHello` message of the initial handshake: with RIE, the client sends an empty extension, whereas with SCSV the client sends a distinguished value in the list of supported ciphersuites.

6.2 Choosing the Right Model for TLS Renegotiation

In this section we will shortly reason our choice of the security model for analysing TLS renegotiation. We also briefly discuss a composability-based approach.

TLS Renegotiation and existing Countermeasures Based on the TLS renegotiation attack of Ray and Dispensa, TLS without countermeasures is not secure in our model for renegotiation (see also Section 6.3 for details). We subsequently show that, generically, TLS with the SCSV/RIE countermeasures is a *weakly secure renegotiable ACCE protocol*. Recall that in this slightly weaker — but still quite reasonable — model, the adversary is slightly restricted in the previous secrets she is allowed to reveal.

Our approach for proving the renegotiable security of TLS with SCSV/RIE countermeasures is modular. We *cannot* generically prove that if a particular TLS ciphersuite is ACCE-secure, then that ciphersuite with SCSV/RIE is a weakly secure renegotiable ACCE, because the protocol itself is modified by including SCSV/RIE and hence a black-box approach does not work. Instead, we consider a modified version of TLS, where an arbitrary tag can be provided as an extension, called *tagged TLS* (see also Section 3.2.3). Via a chain of results and models, we show that if a tagged TLS ciphersuite is secure in an ACCE variant where `Finished` messages are revealed, then that TLS ciphersuite with SCSV/RIE is a weakly secure renegotiable ACCE protocol. This provides a generic justification for the security of SCSV/RIE. Proving that a TLS ciphersuite is secure in this tagged variant model seems to be almost no harder than a proof that that ciphersuite is ACCE-secure; we only needed to change a few lines from the ACCE security proof given in Section 5.3.

We give an exemplary proof of the ephemeral Diffie–Hellman TLS ciphersuites, although other ciphersuites could be proven as well with small effort.

Remark 20. Although these ciphersuites are not currently as widely used as RSA key transport-based ciphersuites, they are growing in use, for example with Google's 2011 announcement that their default ciphersuite is ephemeral elliptic curve Diffie–Hellman [Lan11].

New Countermeasure for TLS. TLS with SCSV/RIE cannot meet our strongest notion of renegotiable security, only the weaker notion of weakly secure renegotiable ACCE. In the strong definition, even if the adversary learns the session key of one phase, parties who later renegotiate still should detect any earlier message injections by the adversary. Though the ability to learn session keys of phases while the protocol is still running makes the adversary quite powerful, this may be realistic in scenarios with long-lived session keys, for example with session resumption. We present in Section 6.8 a simple adjustment to the renegotiation information extension — adding a fingerprint of the transcript of the previous phase's record layer — so TLS can achieve this stronger security notion. This countermeasure can be seen as providing *record layer recognition*, confirming that both parties have the same view of all communicated messages, rather than just *handshake recognition* as in the SCSV/RIE countermeasure.

On Composability and the Similarities to ACCE. It would be desirable to prove the security of the TLS renegotiation countermeasures via some kind of composability framework, such as UC or the game-based composability framework of Brzuska *et al.* [BFS$^+$12]. Unfortunately, this is not possible. The TLS renegotiation countermeasures are not achieved by *composing* in a black-box manner one protocol or primitive with another. Instead, the SCSV/RIE countermeasure looks inside the protocol and changes it in a white-box way: it *modifies* the messages sent by the protocol, and *re-uses* an internal value. Thus we cannot make use of existing security results in a black-box compositional way. Our approach is the 'next best thing': we modify an existing security definition (ACCE) in what seems to be a minimal way, adding just enough 'hooks' to get at the internal values needed to modify and re-use the required values for the SCSV/RIE countermeasure. We are then able to prove in a fully generic way that any TLS protocol that satisfies this slightly modified ACCE notion with hooks is, when using the SCSV/RIE countermeasure, secure against renegotiation attacks. Since the hooks added are quite small, it is not much work to change a proof that a TLS ciphersuite is ACCE secure to show that it satisfies this slightly modified ACCE notion as well.

We chose the ACCE-based approach (see Section 3.2.3) over the game-based composability approach of Brzuska *et al.* [BFS$^+$12] because renegotiation in TLS makes extensive use of the interplay between the handshake and record layer.

Modeling TLS Renegotiation Now we briefly describe how to map TLS renegotiation into our security model and highlight a few components of that mapping, and the alterations necessary when modeling TLS renegotiation instead of plain TLS.

Oracles generally respond to Send, Encrypt, and Decrypt queries as specified by the TLS handshake and record layer protocols. The Send control message $m =$

($\texttt{newphase}, pk$) when sent to a client causes the client to send a new `ClientHello` message, and when sent to a server causes the server to send a new `HelloRequest` message. For the Encrypt and Decrypt queries, we use a content type field *ctype* that corresponds to the `ContentType` field of the `TLSPlaintext` datatype in the TLS record layer specification [DR08, §6.2.1]:

Packets with `ContentType=change_cipher_spec (20)` or `handshake (22)` are considered in our model to have $ctype = \texttt{control}$ and packets with `ContentType =application_data (23)` are considered in our model to have $ctype = \texttt{data}$. We do not explicitly handle messages with `ContentType=alert (21)`. The Reveal query reveals the encryption and MAC keys derived from the master secret, *not* the master secret itself.

6.3 Renegotiation Security of TLS without SCSV/RIE Countermeasures

In this section we analyze the security of TLS renegotiation (without countermeasures) with special focus on the attack by Ray and Dispensa [RD09].

Ray-Dispensa Attack against TLS Renegotiation without Countermeasures Recall the TLS renegotiation attack by Ray and Dispensa, as described previously in Section 6.1 (see also the figure below). The adversary Eve observes Alice attempting to establish a TLS session with Bob. Eve delays Alice's initial `ClientHello` and instead establishes her own TLS session with Bob and transmits a message m_0 over that record layer. Then Eve passes Alice's initial `ClientHello` to Bob over the Eve–Bob record layer.

Bob views this as a valid renegotiation and responds accordingly; Eve relays the handshake messages between Alice and Bob, who will eventually establish a new record layer to which Eve has no access. Alice then transmits a message m_1 over the Alice–Bob record layer. Intuitively, this is a valid attack: Alice believes this is the initial handshake, but Bob believes this is a renegotiated handshake.

Formally, this attack is captured in our weakly secure renegotiable ACCE model of Definition 3.16 as follows. Assume Alice and Bob each have a single oracle instance, and Eve has carried out the attack described in Section 6.1. Then for Bob's oracle π^1_{Bob}, the value of ℓ^* is 2: the last entry in `phases` where Bob has a matching handshake transcript to some handshake transcript in Alice's oracle π^1_{Alice} is the second (and last) `phases` entry. The adversary has broken renegotiation authentication at both Alice and Bob's instances. At Alice by satisfying condition $\mathbf{M'(a)}$ (Alice's first handshake transcript does not match Bob's first handshake transcript), and at Bob by satisfying both $\mathbf{M'(a)}$ (Bob's second handshake transcript does not match Alice's second handshake transcript) and $\mathbf{M'(b)}$ (for every $\ell < 2$, Bob's ℓth handshake and record layer transcript does not match Alice's ℓth transcripts). Thus TLS without countermeasures is not a weakly secure or secure renegotiable ACCE.

6.4 TLS with SCSV/RIE is not a Secure Renegotiable ACCE

Definition 3.15 requires that, even when the adversary can reveal previous phases' session keys, the parties will not successfully renegotiate if the adversary has manipulated the record layer. The SCSV/RIE countermeasures do not protect against this type of adversary. They only provide assurance that handshake transcripts from previous phases match exactly. TLS itself of course provides integrity protection for record layer transcripts via the MACs, but Definition 3.15 allows the adversary to reveal the encryption and MAC keys of previous phases. Thus, an adversary who reveals the current encryption and MAC keys can modify record layer messages but Alice and Bob will still successfully renegotiate a new phase (although the adversary must not alter the number of messages sent, as the *number* of record layer messages sent in the previous phase happens to be protected by SCSV and RIE countermeasures).

We emphasize that while this demonstrates a theoretical weakness in TLS renegotiation countermeasures compared to our very strong security model, it does not translate into an attack on TLS renegotiation countermeasures when intermediate phases' encryption and MAC keys are not revealed.

6.5 TLS-DHE is a Secure Tagged-ACCE-fin

Figure 6.3 in Section 6.1 shows a generic TLS ciphersuite, along with the SCSV/RIE extensions denoted with a dagger. By *tagged TLS*, we mean the generic TLS ciphersuite from Figure 6.3, without any of the SCSV/RIE extensions shown in green, but where an arbitrary string can be placed in the ext_C and ext_S fields. In other words, it is a normal TLS ciphersuite, but with an arbitrary extension field that just carries strings that are not being interpreted as having any particular meaning.

As noted in the beginning of this section, we cannot generically prove that, if a TLS ciphersuite is ACCE-secure, then the tagged version of that ciphersuite is tagged-ACCE- or tagged-ACCE-fin-secure, as we have made white-box modifications to the TLS protocol in introducing the SCSV/RIE countermeasure. Thus we cannot use its security results in a black-box manner. However, in most cases, a white-box approach, where the actual security proof is modified/extended, should be possible, and even very easy. This was indeed the case when we examined tagged TLS-DHE.

For completeness, we will show that TLS-DHE is a secure tagged-ACCE-fin protocol. The proof follows almost exactly the proof that TLS-DHE is a secure ACCE protocol given in Section 5.3. The intuition that leaking the `Finished` messages does not affect security is as follows. The ACCE proof of TLS-DHE exploits the fact that the pseudo-random function is keyed with a value chosen uniformly at random; the proof then replaces the application keys and `Finished` messages with uniformly random values, which are then completely independent of any information exchanged during the handshake. We can use the same technique to show that no adversary having access to the plaintext `Finished` messages can break the security of the sLHAE scheme used in the record layer. Including arbitrary extra data in the handshake messages does not impact security.

Theorem 6.1 (Tagged TLS-DHE is a secure tagged-ACCE-fin)**.** *Let μ be the output length of* PRF *and let λ be the length of the nonces r_C and r_S. Assume that the pseudo-random function* PRF *is $(t, \epsilon_{\mathsf{prf}}, q_{\mathsf{prf}})$-secure, the signature scheme is $(t, \epsilon_{\mathsf{sig}}, q_{\mathsf{sig}})$-secure, the DDH-problem is $(t, \epsilon_{\mathsf{ddh}})$-hard in the group G used to compute the TLS premaster secret, the hash function is $(t, \epsilon_{\mathsf{H}})$-collision resistant, and the PRF-ODH-problem is $(t, \epsilon_{\mathsf{prfodh}})$-hard with respect to G and* PRF*. Suppose that the stateful symmetric encryption scheme is $(t, \epsilon_{\mathsf{sLHAE}})$-secure.*

For any adversary that $(t', \epsilon_{\mathsf{tls}})$-breaks the tagged TLS-DHE in the sense of Definition 3.14 in the tACCE execution environment with $t \approx t'$ and $q_{\mathsf{prf}} \geq 1$, and $q_{\mathsf{sig}} \geq \eta$ it holds that

$$\epsilon_{\mathsf{tls}} \leq 4\eta\phi \left(\frac{\eta\phi}{2^{\lambda}} + \phi\epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + \eta\phi\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} + (\eta\phi + 1) \cdot \left(\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^{\mu}} \right) \right)$$

Recall that ϕ and η are the maximum number of parties and sessions per party; in tagged-ACCE-fin, the number of phases ψ and number of keypairs γ are both at most 1*.*

To prove Theorem 6.1, we again divide the set of all adversaries into two categories:

1. Adversaries that succeed in making an oracle accept maliciously. We call such an adversary an *authentication-adversary.*

2. Adversaries that do not succeed in making any oracle accept maliciously, but which answer the encryption/integrity challenge. We call such an adversary an *encryption-adversary.*

We again prove Theorem 6.1 by two lemmas. Lemma 6.1 bounds the probability ϵ_{auth} that an authentication-adversary succeeds, Lemma 6.2 bounds the probability ϵ_{enc} that an encryption-adversary succeeds. Then we have

$$\epsilon_{\mathsf{tls}} \leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}} \quad .$$

6.5.1 Authentication

Lemma 6.1. *For any adversary running in time $t' \approx t$, the probability that there exists an oracle π_i^s that accepts maliciously is at most*

$$\epsilon_{\mathsf{auth}} \leq 2 \cdot \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^\mu} \right) \right)$$

where all quantities are defined as stated in Theorem 6.1.

Note that $\epsilon_{\mathsf{auth}} \leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}$, where $\epsilon_{\mathsf{client}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Client}$ that accepts maliciously in the sense of Definition 3.14, and $\epsilon_{\mathsf{server}}$ is an upper bound on the probability that there exists an oracle with $\rho = \mathsf{Server}$ that accepts maliciously. Also note that as $\epsilon_{\mathsf{Server}}$ is an upper bound, this implicitly covers the case of performing server-only authentication in all phases (and by definition no server oracle can then accept maliciously, resulting in $\epsilon_{\mathsf{Server}} = 0$).

We claim that

$$\epsilon_{\mathsf{client}} \leq \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^\mu} \right) \right)$$
$$\epsilon_{\mathsf{server}} \leq \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^\mu} \right)$$

and thus

$$\begin{aligned} \epsilon_{\mathsf{auth}} &\leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} \\ &\leq 2 \cdot \eta\phi \left(\frac{\eta\phi}{2^\lambda} + \phi \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^\mu} \right) \right) . \end{aligned}$$

Proof of Lemma 6.1: $\epsilon_{\mathsf{client}}$

PROOF. We first show, that the probability that there exists an oracle with $\rho = \mathsf{Client}$ that accepts maliciously in the sense of Definition 3.14 is negligible. The proof proceeds in a *sequence of games*, following [BR06, Sho04]. The first game is the real security experiment. We then describe several intermediate games that modify the original game step-by-step, and argue that our complexity assumptions imply that each game is computationally indistinguishable from the previous one. We end up in the final game, where no adversary can break the security of the protocol.

Let $\mathsf{break}_\delta^{(1)}$ be the event that occurs when the first oracle that accepts maliciously in the sense of Definition 3.14 with $\rho = \mathsf{Client}$ in Game δ.

Game 0. This game equals the multi-phase ACCE security experiment used in Section 3.2.3. Thus, for some $\epsilon_{\mathsf{client}}$ we have

$$\Pr[\mathsf{break}_0^{(1)}] = \epsilon_{\mathsf{client}} \ .$$

Game 1. In this game we add an abort rule. The challenger aborts if there exists any oracle π_i^s that chooses a random nonce r_C or r_S which is not unique. More precisely, the game is aborted if the adversary ever makes a first Send query to an oracle π_i^s, and the oracle replies with random nonce r_C or r_S such that there exists some other oracle $\pi_{i'}^{s'}$ which has previously sampled the same nonce.

In total less than $\phi\eta$ nonces r_C and r_S are sampled, each uniformly random from $\{0,1\}^\lambda$. Thus, the probability that a collision occurs is bounded by $(\phi\eta)^2 \cdot 2^{-\lambda}$, which implies

$$\Pr[\mathsf{break}_0^{(2)}] \leq \Pr[\mathsf{break}_1^{(2)}] + \frac{(\phi\eta)^2}{2^\lambda} \ .$$

Note that now each oracle has a unique nonce r_C or r_S, which is included in the signatures. We will use this to ensure that each oracle that accepts with non-corrupted partner has a *unique* partner oracle.

Game 2. We try to guess which client oracle will be the first oracle to accept maliciously and the phase in which this happens. If our guess is wrong, i.e. if there is another (Client or Server) oracle that accepts before or if they accept in a different phase, then we abort the game.

Technically, this game is identical, except for the following. The challenger guesses two random indices $(i^*, s^*) \stackrel{\$}{\leftarrow} [\phi] \times [\eta]$. If there exists an oracle π_i^s that accepts maliciously, and $(i,s) \neq (i^*, s^*)$ and π_i^s has $\rho \neq \mathsf{Client}$, then the challenger aborts the game. Note that if the first oracle π_i^s that accepts maliciously has $\rho = \mathsf{Client}$, then with probability $1/(\phi \cdot \eta)$ we have $(i,s) = (i^*, s^*)$, and thus

$$\Pr[\mathsf{break}_1^{(2)}] = \phi\eta \cdot \Pr[\mathsf{break}_2^{(2)}] \ .$$

Note that in this game the adversary can only break the security of the protocol if oracle $\pi_{i^*}^{s^*}$ is the first oracle that accepts maliciously and has $\rho = \mathsf{Client}$; otherwise the game is aborted.

Game 3. Again the challenger proceeds as before, but we add an abort rule. We want to make sure that $\pi_{i^*}^{s^*}$ receives as input exactly the Diffie–Hellman value T_S that was selected by some other uncorrupted oracle that received the nonce r_C chosen by $\pi_{i^*}^{s^*}$ as first input (note that there may be several such oracles, since the adversary may send copies of r_C to many oracles).

Technically, we abort and raise event $\mathsf{abort_{sig}}$, if oracle $\pi_{i^*}^{s^*}$ ever receives as input a message $m_3 = cert_S$ indicating intended partner $\Pi = j$ and message $m_4 = (p, g, T_S, \sigma_S)$ such that σ_S is a valid signature over $r_C \| r_S \| p \| g \| T_S$, but there exists no oracle π_j^t which has previously output σ_S. Clearly we have

$$Pr[\mathsf{break}_2^{(1)}] \leq \Pr[\mathsf{break}_3^{(1)}] + \Pr[\mathsf{abort_{sig}}] \ .$$

Note that the experiment is aborted, if $\pi_{i^*}^{s^*}$ does not accept *maliciously*, due to Game 2. This means that party P_j must not be corrupted when $\pi_{i^*}^{s^*}$ accepts (as otherwise $\pi_{i^*}^{s^*}$ does not accept *maliciously*). To show that $\Pr[\mathsf{abort_{sig}}] \leq \phi \cdot \epsilon_{\mathsf{sig}}$, we construct a signature forger as follows. The forger receives as input a public key pk^* and simulates the challenger for $\mathcal{A}$. It guesses index $\phi \stackrel{\$}{\leftarrow} [\phi]$, sets $pk_\phi = pk^*$, and generates all long-term public/secret keys as before. Then it proceeds as the challenger in Game 3, except that it uses its chosen-message oracle to generate a signature under pk_ϕ when necessary.

If $\phi = j$ and the corresponding public key is pk_j, which happens with probability $1/(\phi)$, then the forger can use the signature received by $\pi_{i^*}^{s^*}$ to break the EUF-CMA security of the signature scheme with success probability ϵ_{sig}. Therefore we gain that $\Pr[\mathsf{abort_{sig}}]/(\phi) \leq \epsilon_{\mathsf{sig}}$; if $\Pr[\mathsf{abort_{sig}}]$ is not negligible, then ϵ_{sig} is not negligible as well and we have

$$Pr[\mathsf{break}_2^{(1)}] \leq \Pr[\mathsf{break}_3^{(1)}] + \phi\epsilon_{\mathsf{sig}} \ .$$

Note that in Game 3 oracle $\pi_{i^*}^{s^*}$ receives as input a Diffie–Hellman value T_S such that T_S was chosen by another oracle, but not by the adversary. Note also that there may be multiple oracles that issued a signature σ_S containing r_C, since the adversary may have sent several copies of r_C to several oracles.

Game 4. In this game we want to make sure that we know the oracle π_j^t which will issue the signature σ_S that $\pi_{i^*}^{s^*}$ receives. Note that this signature includes the random nonce r_S, which is unique due to Game 1. Therefore the challanger in this game proceeds as before, but additionally guesses two indices $(j^*, t^*) \xleftarrow{\$} [\phi] \times [\eta]$. It aborts, if the adversary does *not* make a Send query containing r_C to $\pi_{j^*}^{t^*}$ and $\pi_{j^*}^{t^*}$ responds in this phase with messages containing σ_S such that σ_S is forwarded to $\pi_{i^*}^{s^*}$.

We know that there must exist at least one oracle that outputs σ_S in some phase such that σ_S is forwarded to $\pi_{i^*}^{s^*}$, due to Game 3. Thus we have

$$\Pr[\mathsf{break}_3^{(1)}] \leq \phi\eta \Pr[\mathsf{break}_4^{(1)}] \ .$$

Note that in this game we know exactly that oracle $\pi_{j^*}^{t^*}$ chooses the Diffie–Hellman share T_S that $\pi_{i^*}^{s^*}$ uses to compute its premaster secret.

Game 5. Recall that $\pi_{i^*}^{s^*}$ computes the master secret as $ms = \mathsf{PRF}(T_S^{t_c}, label_1 \| r_C \| r_S)$, where T_S denotes the Diffie–Hellman share received from $\pi_{j^*}^{t^*}$, and t_c denotes the Diffie–Hellman exponent chosen by $\pi_{i^*}^{s^*}$. In this game we replace the master secret ms computed by $\pi_{i^*}^{s^*}$ with an independent random value $\widetilde{ms}$. Moreover, if $\pi_{j^*}^{t^*}$ receives as input the same Diffie–Hellman share T_C that was sent from $\pi_{i^*}^{s^*}$, then we set the master secret of $\pi_{j^*}^{t^*}$ equal to $\widetilde{ms}$. Otherwise we compute the master secret as specified in the protocol. We claim that

$$Pr[\mathsf{break}_4^{(1)}] \leq \Pr[\mathsf{break}_5^{(1)}] + \epsilon_{\mathsf{prfodh}} \ .$$

Suppose there exists an adversary $\mathcal{A}$ that distinguishes Game 5 from Game 4. We show that this implies an adversary $\mathcal{B}$ that solves the PRF-ODH problem.

Adversary $\mathcal{B}$ outputs $(label_1 \| r_C \| r_S)$ to its oracle and receives in response (g, g^u, g^v, R), where either $R = \mathsf{PRF}(g^{uv}, label_1 \| r_C \| r_S)$ or $R \xleftarrow{\$} \{0,1\}^\mu$. It runs $\mathcal{A}$ by implementing the challenger for $\mathcal{A}$, and embeds (g^u, g^v) as follows. Instead of letting $\pi_{i^*}^{s^*}$ choose $T_C = g^{t_C}$ for random $t_C \xleftarrow{\$} \mathbb{Z}_q$, $\mathcal{B}$ defines $T_C := g^u$. Similarly, the Diffie–Hellman share T_S of $\pi_{j^*}^{t^*}$ is defined as $T_S := g^v$. Finally, the master secret of $\pi_{i^*}^{s^*}$ is set equal to R.

Note that $\pi_{i^*}^{s^*}$ computes the master secret after receiving T_S from $\pi_{j^*}^{t^*}$, and then it sends $m_8 = T_C$. If the adversary decides to forward m_8 to $\pi_{j^*}^{t^*}$, then the master secret of $\pi_{j^*}^{t^*}$ is set equal to R. If $\pi_{j^*}^{t^*}$ receives $T_{C'} \neq T_C$, then $\mathcal{B}$ queries its oracle to compute $ms' = \mathsf{PRF}(T_{C'}^v, label_1 \| r_C \| r_S)$, and sets the master secret of $\pi_{j^*}^{t^*}$ equal to ms'.

Note that in any case algorithm $\mathcal{B}$ knows the master secret of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, and thus is able to compute all further protocol messages (in particular the finished messages fin_C and fin_S) and answer a potential Reveal query to $\pi_{j^*}^{t^*}$ as required (note that there is no Reveal query to $\pi_{i^*}^{s^*}$, as otherwise the experiment is aborted, due to Game 2). If $R = \mathsf{PRF}(g^{uv}, label_1 \| r_C \| r_S)$, then the view of $\mathcal{A}$ is identical to Game 4, while if $R \xleftarrow{\$} \{0,1\}^\mu$ then it is identical to Game 5, which yields the above claim.

Game 6. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ with a random function F. If $\pi_{j^*}^{t^*}$ uses the same master secret $\widetilde{ms}$ as $\pi_{i^*}^{s^*}$ (cf. Game 5), then the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{j^*}^{t^*}$ is replaced as well. Of course the same random function is used for both oracles sharing the same $\widetilde{ms}$. In particular, this function is used to compute the `Finished` messages by both partner oracles.

Distinguishing Game 6 from Game 5 implies an algorithm breaking the security of the pseudorandom function PRF, thus

$$\Pr[\mathsf{break}_5^{(1)}] \leq \Pr[\mathsf{break}_6^{(1)}] + \epsilon_{\mathsf{prf}} \ .$$

Game 7. In Game 6 we have replaced the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ with a random function. We now want to make sure, that the `ServerFinished` message still cannot be predicted by an adversary. Remember that the `ServerFinished` is computed as

$$\mathsf{fin}_C^* = F(label_4 || \mathsf{H}(m_1 || \cdots || m_{12})),$$

where $m_1 || \cdots || m_{12}$ denotes the transcript of all messages sent and received by $\pi_{i^*}^{s^*}$.

Before we can do so, we need to make sure that the only other oracle potentially having access to F, which is $\pi_{j^*}^{t^*}$, never evaluates the function F on any input $label_4 || \mathsf{H}(m')$ with

$$m' \neq m_1 || \cdots || m_{12} \quad \text{and} \quad \mathsf{H}(m') = \mathsf{H}(m_1 || \cdots || m_{12}).$$

We now abort the game, if oracle $\pi_{j^*}^{t^*}$ ever evaluates the conditions hold. Since that directly implies a collision for the hash function H, we have

$$\Pr[\mathsf{break}_6^{(1)}] \leq \Pr[\mathsf{break}_7^{(1)}] + \epsilon_{\mathsf{H}}$$

Game 8. Now we use that the full transcript of all messages sent and received (*including the tags*) is used to compute the `Finished` messages, and that `Finished` messages are computed by evaluating a truly random function that is only accessible to $\pi_{i^*}^{s^*}$ and (possibly) $\pi_{j^*}^{t^*}$ due to Game 7.

The `Finished` messages are computed by evaluating a truly random function $F_{\widetilde{ms}}$, so they are completely independent of the master secret of the current phase. This allows us to show that any adversary has probability at most $2^{-\mu}$ of learning the `Finished` messages. We have

$$\Pr[\mathsf{break}_7^{(1)}] \leq \Pr[\mathsf{break}_8^{(1)}] + \frac{1}{2^\mu} \ .$$

Also note, that leaking the `Finished` messages now does not reveal any information about this phase to the adversary.

Game 9. Finally we use that the key material $K_{\mathsf{enc}}^{C \to S} || K_{\mathsf{enc}}^{S \to C} || K_{\mathsf{mac}}^{C \to S} || K_{\mathsf{mac}}^{S \to C}$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ in the stateful symmetric encryption scheme is drawn uniformly at random and independent of all TLS handshake messages. This game proceeds exactly like the previous game, except that the challenger now aborts if oracle $\pi_{i^*}^{s^*}$ accepts without having a matching conversation to $\pi_{j^*}^{t^*}$. Thus we have $\Pr[\mathsf{break}_9^{(1)}] = 0$.

The only remaining way for an adversary to make the client oracle $\pi_{i^*}^{s^*}$ maliciously accept and win is to output a fresh, valid encryption of the `Finished` message fin_S, which must be distinct from the ciphertext output by $\pi_{j^*}^{t^*}$. If the adversary now outputs such a ciphertext, we can directly use it to break the security of the sLHAE scheme, thus

$$\Pr[\mathsf{break}_8^{(1)}] \leq \Pr[\mathsf{break}_9^{(1)}] + \epsilon_{\mathsf{sLHAE}} = \epsilon_{\mathsf{sLHAE}} \ .$$

□

Proof of Lemma 6.1: $\epsilon_{\mathsf{server}}$

Proof. We now show that the probability that there exists an oracle with $\rho = \mathsf{Server}$ that accepts maliciously in the sense of Definition 3.14 is negligible. Let $\mathsf{break}_\delta^{(2)}$ be the event that occurs when the first oracle that accepts maliciously in the sense of Definition 3.14 with $\rho = \mathsf{Server}$ in Game δ.

Game 0. This game equals the ACCE security experiment described in Definition 3.14. Thus, for some $\epsilon_{\mathsf{server}}$ we have

$$\Pr[\mathsf{break}_0^{(2)}] = \epsilon_{\mathsf{server}} \ .$$

Game 1. In this game we add an abort rule. The challenger aborts, if there exists any oracle π_i^s that chooses a random nonce r_C or r_S which is not unique. With the same arguments as in Game 1 of the first proof we have

$$\Pr[\mathsf{break}_0^{(2)}] \leq \Pr[\mathsf{break}_1^{(2)}] + \frac{(\phi\eta)^2}{2^\lambda} \ .$$

Game 2. This game is identical, except for the following. The challenger guesses three random indices $(i^*, s^*) \xleftarrow{\$} [\phi] \times [\eta]$. If there exists an oracle π_i^s that accepts maliciously, and $(i, s) \neq (i^*, s^*)$ and π_i^s has $\rho \neq \mathsf{Server}$, then the challenger aborts the game. Note that if the first oracle π_i^s that accepts maliciously has $\rho = \mathsf{Server}$, then with probability $1/(\phi\eta)$ we have $(i, s) = (i^*, s^*)$, and thus

$$\Pr[\mathsf{break}_1^{(2)}] = (\phi\eta) \cdot \Pr[\mathsf{break}_2^{(2)}] \ .$$

Note that in this game the adversary can only break the security of the protocol if oracle $\pi_{i^*}^{s^*}$ is the first oracle that accepts maliciously and has $\rho = \mathsf{Server}$; otherwise the game is aborted.

Game 3. The challenger proceeds as before, but we add an abort rule. We want to make sure that $\pi_{i^*}^{s^*}$ receives as input exactly the Diffie–Hellman value $m_8 = T_C$ that was selected by some other uncorrupted oracle.

Technically, we abort and raise event $\mathsf{abort_{sig}}$, if oracle $\pi_{i^*}^{s^*}$ ever receives as input a message $m_7 = cert_C$ indicating intended partner $\Pi = j$ and message $m_9 = \sigma_C$ such that $\sigma_C = \mathsf{SIG.Sign}(sk_C, m_1\| \ldots, \|m_8)$ is a valid signature but there exists no oracle π_j^t which has previously output σ_C. Clearly we have

$$Pr[\mathsf{break}_2^{(2)}] \leq \Pr[\mathsf{break}_3^{(2)}] + \Pr[\mathsf{abort_{sig}}] \ .$$

Note that the experiment is aborted if $\pi_{i^*}^{s^*}$ does not accept *maliciously*, due to Game 2. This means that party P_j must not be corrupted when $\pi_{i^*}^{s^*}$ accepts. To show that $\Pr[\mathsf{abort_{sig}}] \leq$

$(\phi) \cdot \epsilon_{\mathsf{sig}}$, we construct a signature forger as follows. The forger receives as input a public key pk^* and simulates the challenger for $\mathcal{A}$. It guesses index $\phi \xleftarrow{\$} [\phi]$, sets $pk_\phi = pk^*$, and generates all long-term public/secret keys as before. Then it proceeds as the challenger in Game 3, except that it uses its chosen-message oracle to generate a signature under pk_ϕ when necessary.

If $\phi = j$ and the corresponding public key is pk_j, which happens with probability $1/(\phi)$, then the forger can use the signature received by $\pi_{i^*}^{s^*}$ to break the EUF-CMA security of the signature scheme with success probability ϵ_{sig}, so $\Pr[\mathsf{abort}_{\mathsf{sig}}]/(\phi) \leq \epsilon_{\mathsf{sig}}$. Therefore if $\Pr[\mathsf{abort}_{\mathsf{sig}}]$ is not negligible, then ϵ_{sig} is not negligible as well and we have

$$Pr[\mathsf{break}_2^{(2)}] \leq \Pr[\mathsf{break}_3^{(2)}] + \phi\epsilon_{\mathsf{sig}} \ .$$

Note that in Game 3 oracle $\pi_{i^*}^{s^*}$ receives as input a Diffie–Hellman value T_C such that T_C was chosen in some phase by another oracle, but not by the adversary. Note also that this phase of this oracle is unique, since the signature includes the client nonce r_C, which is unique due to Game 1. From now on we denote this unique oracle and phase with $\pi_{j^*}^{t^*}$.

Note also that $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ share a premaster secret $pms = T_C^{t_S} = T_S^{t_C}$, where $T_C = g^{t_C}$ and $T_S = g^{t_S}$ for random exponents t_S and t_C chosen by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively.

Game 4. In this game, we replace the premaster secret $pms = g^{t_C t_S}$ shared by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random value g^r, $r \xleftarrow{\$} \mathbb{Z}_q$. The fact that the challenger has full control over the Diffie–Hellman shares T_C and T_S exchanged between $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, due to the modifications introduced in the previous games, provides us with the ability to prove indistinguishability under the Decisional Diffie–Hellman assumption.

Technically, the challenger in Game 4 proceeds as before, but when $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ compute the premaster secret as $pms = g^{t_C t_S}$, the challenger replaces this value with a uniformly random value $\widetilde{pms} = g^r, r \xleftarrow{\$} \mathbb{Z}_p^*$, which is in the following used by both partner oracles.

Suppose there exists an algorithm distinguishing Game 4 from Game 3. Then we can construct an algorithm $\mathcal{B}$ solving the DDH problem as follows. Algorithm $\mathcal{B}$ receives as input a DDH challenge (g, g^u, g^v, g^w). The challenger defines $T_C := g^u$ and $T_S := g^v$ for the Diffie–Hellman shares chosen by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively. Instead of computing the Diffie–Hellman key as in Game 3, it sets $pms = g^w$ both for the client and the server oracle. Now if $w = uv$, then this game proceeds exactly like Game 3, while if w is random then this game proceeds exactly like Game 4. Thus,

$$\Pr[\mathsf{break}_3^{(2)}] \leq \Pr[\mathsf{break}_4^{(2)}] + \epsilon_{\mathsf{ddh}} \ .$$

Note that in Game 4 the premaster secret of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is uniformly random and independent of T_C and T_S. This will provide us with the ability to replace the function $\mathsf{PRF}(\widetilde{pms}, \cdot)$ with a truly random function in the next game.

Game 5. In Game 5 we make use of the fact that the premaster secret $\widetilde{pms}$ of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is chosen uniformly at random, independently of T_C and T_S. We thus replace the value $ms = \mathsf{PRF}(\widetilde{pms}, label_1 \| r_C \| r_S)$ with a random value $\widetilde{ms}$.

Distinguishing Game 5 from Game 4 implies an algorithm breaking the security of the pseudorandom function PRF, thus

$$\Pr[\mathsf{break}_4^{(2)}] \leq \Pr[\mathsf{break}_5^{(2)}] + \epsilon_{\mathsf{prf}} \ .$$

Game 6. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random function F. Of course the same random function is used for both oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$. In particular, this function is used to compute the `Finished` messages by both partner oracles.

Distinguishing Game 6 from Game 5 again implies an algorithm breaking the security of the pseudorandom function PRF, thus

$$\Pr[\mathsf{break}_5^{(2)}] \leq \Pr[\mathsf{break}_6^{(2)}] + \epsilon_{\mathsf{prf}} \ .$$

Game 7. In Game 6 we have replaced the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ with a random function F. We now want to make sure, that the `ClientFinished` message still cannot be predicted by an adversary. Remember that the `ClientFinished` is computed as

$$\mathsf{fin}_S^* = F(label_3 || \mathsf{H}(m_1 || \cdots || m_{10})),$$

where $m_1 || \cdots || m_{10}$ denotes the transcript of all messages sent and received by $\pi_{i^*}^{s^*}$.

Before we can do so, we need to make sure that the only other oracle potentially having access to F, which is $\pi_{j^*}^{t^*}$, never evaluates the function F on any input $label_3 || \mathsf{H}(m')$ with

$$m' \neq m_1 || \cdots || m_{10} \quad \text{and} \quad \mathsf{H}(m') = \mathsf{H}(m_1 || \cdots || m_{10}).$$

We now abort the game, if oracle $\pi_{j^*}^{t^*}$ ever evaluates the conditions hold. Since that directly implies a collision for the hash function H, we have

$$\Pr[\mathsf{break}_6^{(2)}] \leq \Pr[\mathsf{break}_7^{(2)}] + \epsilon_{\mathsf{H}}$$

Game 8. Finally we use that the full transcript of all messages sent and received (**including the tags**) is used to compute the `Finished` messages, and that `Finished` messages are computed by evaluating a truly random function that is only accessible to $\pi_{i^*}^{s^*}$ and (possibly) $\pi_{j^*}^{t^*}$ due to Game 7.

The `Finished` messages are computed by evaluating a truly random function $F_{\widetilde{ms}}$, so they are completely independent of the master secret of the current phase. This allows us to show that any adversary has probability at most $2^{-\mu}$ of learning the `Finished` messages.

Thus we have

$$\Pr[\mathsf{break}_7^{(2)}] \leq \Pr[\mathsf{break}_8^{(2)}] + \frac{1}{2^{\mu}}.$$

Also note, that leaking the `Finished` messages now does not reveal any information about this phase to the adversary.

Game 9. Finally we use that the key material $K_{\mathsf{enc}}^{C \to S} \| K_{\mathsf{enc}}^{S \to C} \| K_{\mathsf{mac}}^{C \to S} \| K_{\mathsf{mac}}^{S \to C}$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ in the stateful symmetric encryption scheme is drawn uniformly at random and independent of all TLS handshake messages. Thus, this game proceeds exactly like the previous game, except that the challenger now aborts if oracle $\pi_{i^*}^{s^*}$ accepts without having a matching conversation to $\pi_{j^*}^{t^*}$. Thus we have $\Pr[\mathsf{break}_9^{(2)}] = 0$.

The only remaining way for an adversary to make the server oracle $\pi_{i^*}^{s^*}$ maliciously accept and win is to output a fresh, valid encryption of the `ClientFinished` message fin_C, which must be distinct from the ciphertext output by $\pi_{j^*}^{t^*}$. If the adversary now outputs such a ciphertext, we can directly use it to break the security of the sLHAE scheme, thus

$$\Pr[\mathsf{break}_8^{(2)}] \leq \Pr[\mathsf{break}_9^{(2)}] + \epsilon_{\mathsf{sLHAE}} = \epsilon_{\mathsf{sLHAE}} \ .$$

□

Collecting probabilities of both previous sections yields Lemma 6.1. We obtain that

$$\begin{aligned}\epsilon_{\mathsf{auth}} &\leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}}\\ &\leq 2 \cdot \eta\phi \left(\frac{\eta\phi}{2^{\lambda}} + \phi \cdot \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2 \cdot \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} + \eta\phi \left(\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^{\mu}} \right) \right) .\end{aligned}$$

6.5.2 Indistinguishability of Ciphertexts

Proof of Confidentiality

Lemma 6.2. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that $\mathcal{A}$ answers the encryption-challenge correctly is at most $1/2 + \epsilon_{\mathsf{enc}}$ with*

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + \phi\eta \left(\epsilon_{\mathsf{ddh}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} \right) \ ,$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 3.14 (cf. Lemma 6.1) and all other quantities are defined as stated in Theorem 6.1.

Proof. Assume without loss of generality that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 3.14 are satisfied. Let $\mathsf{break}_{\delta}^{(3)}$ denote the event that $b' = b$ in Game δ, where b is the random bit sampled by the Test query, and b' is either the bit output by $\mathcal{A}$ or (if $\mathcal{A}$ does not output a bit) chosen by the challenger. Let $\mathsf{Adv}_{\delta} := \Pr[\mathsf{break}_{\delta}^{(3)}] - 1/2$ denote the *advantage* of $\mathcal{A}$ in Game δ. Consider the following sequence of games.

Game 0. This game equals the ACCE security experiment used in Section 3.2.3. For some ϵ_{enc} we have

$$\Pr[\mathsf{break}_{0}^{(3)}] = \frac{1}{2} + \epsilon_{\mathsf{enc}} = \frac{1}{2} + \mathsf{Adv}_0 \ .$$

Game 1. The challenger in this game proceeds as before, but it aborts and chooses b' uniformly random if there exists any oracle that accepts maliciously in any phase in the sense of Definition 3.16. Thus we have

$$\mathsf{Adv}_0 \leq \mathsf{Adv}_1 + \epsilon_{\mathsf{auth}} \ ,$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 3.14 (cf. Lemma 6.1).

Recall that we assume that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 3.14 are satisfied. In particular it outputs (i, s, b') such that π_i^s accepts with intended partner $\Pi = j$, and P_j is not corrupted. Note that in Game 1 for any such phase π_i^s there exists a unique partner phase π_j^t such that $\pi_i^s.\mathtt{phases}[1].T$ has a matching conversation to $\pi_j^t.\mathtt{phases}[1].T$, as the game is aborted otherwise.

Game 2. The challenger in this game proceeds as before, but in addition guesses indices $(i^*, s^*) \xleftarrow{\$} [\phi] \times [\eta]$. It aborts and chooses b' at random if the adversary outputs (i, s, b') with $(i, s) \neq (i^*, s^*)$. With probability $1/(\phi\eta)$ we have $(i, s) = (i^*, s^*)$, and thus

$$\mathsf{Adv}_1 \leq \phi\eta\mathsf{Adv}_2 \ .$$

Note that in Game 2 we know that $\mathcal{A}$ will output (i^*, s^*, b'). Note also that $\pi_{i^*}^{s^*}$ has a unique partner due to Game 1. In the sequel we denote with $\pi_{j^*}^{t^*}$ the unique oracle and phase such that $\pi_{i^*}^{s^*}$ has a matching conversation to $\pi_{j^*}^{t^*}$, and say that $\pi_{j^*}^{t^*}$ is the *partner* of $\pi_{i^*}^{s^*}$.

Game 3. The challenger in this game proceeds as before, but replaces the premaster secret pms of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random group element $\widetilde{pms} = g^w$, $w \xleftarrow{\$} \mathbb{Z}_q$. Note that both g^u and g^v are chosen by oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$, respectively, as otherwise $\pi_{i^*}^{s^*}$ would not have a matching conversation to $\pi_{j^*}^{t^*}$ and the game would be aborted. Thus, both oracles compute the premaster secret as $pms = g^{uv}$. Let $T_{i^*,s^*} = g^u$ denote the Diffie–Hellman share chosen by $\pi_{i^*}^{s^*}$, and let $T_{j^*,t^*} = g^v$ denote the share chosen by its partner $\pi_{j^*}^{t^*}$.

Suppose that there exists an algorithm $\mathcal{A}$ distinguishing Game 3 from Game 2. Then we can construct an algorithm $\mathcal{B}$ solving the DDH problem as follows. $\mathcal{B}$ receives as input (g, g^u, g^v, g^w). It implements the challenger for $\mathcal{A}$ as in Game 2, except that it sets $T_{i^*,s^*} := g^u$ and $T_{j^*,t^*} := g^v$, and the premaster secret of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ equal to $pms := g^w$. Note that $\mathcal{B}$ can simulate all messages exchanged between $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ properly, in particular the finished messages using knowledge of $pms = g^w$. Since all other oracles are not modified, $\mathcal{B}$ can simulate these oracles properly as well.

If $w = uv$, then the view of $\mathcal{A}$ when interacting with $\mathcal{B}$ is identical to Game 2, while if $w \xleftarrow{\$} \mathbb{Z}_q$ then it is identical to Game 3. Thus,

$$\mathsf{Adv}_2 \leq \mathsf{Adv}_3 + \epsilon_{\mathsf{ddh}} \ .$$

Game 4. In Game 4 we make use of the fact that the premaster secret $\widetilde{pms}$ of $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is chosen uniformly random. We thus replace the value $ms = \mathsf{PRF}(\widetilde{pms}, label_1 \| r_C \| r_S)$ with a random value $\widetilde{ms}$.

Distinguishing Game 4 from Game 3 implies an algorithm breaking the security of the pseudorandom function PRF, thus

$$\mathsf{Adv}_3 \leq \mathsf{Adv}_4 + \epsilon_{\mathsf{prf}} \ .$$

Game 5. In this game we replace the function $\mathsf{PRF}(\widetilde{ms}, \cdot)$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ with a random function $F_{\widetilde{ms}}$. Of course the same random function is used for both oracles (in their respective phases) $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$. In particular, this function is used to compute the key material as

$$K_{\mathsf{enc}}^{C\to S} \| K_{\mathsf{enc}}^{S\to C} \| K_{\mathsf{mac}}^{C\to S} \| K_{\mathsf{mac}}^{S\to C} := F_{\widetilde{ms}}(label_2 \| r_C \| r_S) \ .$$

Distinguishing Game 5 from Game 4 again implies an algorithm breaking the security of the pseudorandom function PRF. Moreover, in Game 5 the adversary always receives a random key in response to a Test query, and thus receives no information about b', which implies $\mathsf{Adv}_5 = 0$ and

$$\mathsf{Adv}_4 \leq \mathsf{Adv}_5 + \epsilon_{\mathsf{prf}} = \epsilon_{\mathsf{prf}} \ .$$

Note that in Game 5 the key material $K_{\mathsf{enc}}^{C\to S} \| K_{\mathsf{enc}}^{S\to C} \| K_{\mathsf{mac}}^{C\to S} \| K_{\mathsf{mac}}^{S\to C}$ of oracles $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ is uniformly random and independent of all TLS handshake messages exchanged in the pre-accept phase.

Game 6. Now we use that the key material $K_{\mathsf{enc}}^{C\to S}\|K_{\mathsf{enc}}^{S\to C}\|K_{\mathsf{mac}}^{C\to S}\|K_{\mathsf{mac}}^{S\to C}$ used by $\pi_{i^*}^{s^*}$ and $\pi_{j^*}^{t^*}$ in the stateful symmetric encryption scheme is drawn uniformly at random and independent of all TLS handshake messages.

In this game we construct a simulator $\mathcal{B}$ that uses a successful ACCE adversary $\mathcal{A}$ to break the security of the underlying sLHAE secure symmetric encryption scheme. By assumption, the simulator $\mathcal{B}$ is given access to an encryption oracle Encrypt and a decryption oracle Decrypt. $\mathcal{B}$ embeds the sLHAE experiment by simply forwarding all $\mathsf{Encrypt}(\pi_{i^*}^{s^*},\cdot)$ queries to Encrypt, and all $\mathsf{Decrypt}(\pi_{j^*}^{t^*},\cdot)$ queries to Decrypt. Otherwise it proceeds as the challenger in Game 5.

Observe that the values generated in this game are exactly distributed as in the previous game. We thus have

$$\mathsf{Adv}_5 = \mathsf{Adv}_6 \enspace .$$

If $\mathcal{A}$ outputs a triple (i^*, s^*, b'), then $\mathcal{B}$ forwards b' to the sLHAE challenger. Otherwise it outputs a random bit. Since the simulator essentially relays all messages it is easy to see that an adversary $\mathcal{A}$ having advantage ϵ' yields an adversary $\mathcal{B}$ against the sLHAE security of the encryption scheme with success probability at least $1/2 + \epsilon'$.

Since by assumption any adversary has advantage at most $\epsilon_{\mathsf{sLHAE}}$ in breaking the sLHAE security of the symmetric encryption scheme, we have

$$\mathsf{Adv}_6 \leq 1/2 + \epsilon_{\mathsf{sLHAE}} \enspace .$$

□

Adding up probabilities from Lemmas 6.1 and 6.2, we obtain that

$$\begin{aligned}
\epsilon_{\mathsf{tls}} &\leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}} \\
&\leq d\ell\left(\frac{d\ell}{2^{\lambda-2}} + 4\ell\epsilon_{\mathsf{sig}} + 3\epsilon_{\mathsf{ddh}} + 2d\ell\epsilon_{\mathsf{prfodh}} + 4\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} + 2\left(d\ell+1\right)\left(\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^{\mu}}\right)\right) \\
&\leq 4d\ell\left(\frac{d\ell}{2^{\lambda}} + \ell\epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + d\ell\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} + (d\ell+1)\cdot\left(\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^{\mu}}\right)\right) \\
&\leq 4\eta\phi\left(\frac{\eta\phi}{2^{\lambda}} + \phi\epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + \eta\phi\epsilon_{\mathsf{prfodh}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}} + (\eta\phi+1)\cdot\left(\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{sLHAE}} + \frac{1}{2^{\mu}}\right)\right)
\end{aligned}$$

which yields Theorem 6.1.

Note, that we do lose some tightness compared to the original ACCE proof of TLS-DHE. For the authentication game, we additionally have to guess the phase in which the adversary makes an oracle maliciously accept, and for the encryption game we also have to guess the phase to which we input the challenge keys.

6.6 TLS with SCSV/RIE is a Secure Multi-Phase ACCE

We begin by showing that including the SCSV/RIE countermeasure does not *weaken* security. In other words, putting the `Finished` messages in the `ClientHello` and `ServerHello` does not introduce any vulnerabilities. Having done so, in the next subsection we will show how including the SCSV/RIE countermeasure yields a weakly secure renegotiable ACCE.

Theorem 6.2 (TLS with SCSV/RIE is a secure multi-phase ACCE)**.** *Let* Π *be a generic tagged TLS ciphersuite as described in Section 3.2.3. Assume that* Π *is* $(t, \epsilon_{\mathsf{tagged}})$*-tagged-ACCE-fin-secure. Let* Π' *denote* Π *with the SCSV/RIE countermeasure as described in Figure 6.3. For any adversary that* $(t', \epsilon_{\mathsf{mp}})$*-breaks the multi-phase ACCE security of* Π' *with* $\tau \approx \tau'$*, it holds* *that $\epsilon_{\mathsf{mp}} \leq 2\epsilon'$, where ϵ' is obtained from ϵ by replacing all instances of ϕ in ϵ with $\phi \cdot \gamma$ and replacing all instances of η in ϵ with $\eta \cdot \psi$. (Recall that ϕ, η, ψ, and γ are the maximum number of parties, sessions per party, phases per session, and keypairs per party, respectively.)*

Proof. The basic idea of the proof is as follows. We will construct a multi-phase ACCE simulator $\mathcal{S}$ for Π' that makes use of an tagged-ACCE-fin challenger $\mathcal{C}$ for Π. $\mathcal{S}$ will simulate every (party, public key) pair and every (session, phase) pair with distinct parties and sessions in $\mathcal{C}$. For the most part, $\mathcal{S}$ will relay queries down to $\mathcal{C}$ and return the result. However, for queries that relate to renegotiation (Send, Decrypt), $\mathcal{S}$ needs to carefully manage the handshake messages and transition one session in $\mathcal{C}$ to another.

First, consider when the adversary is causing two honest parties to negotiate their first phase in a session. The simulator will pass these queries down to the tagged-ACCE-fin challenger and pass the responses back up to the adversary. Eventually these sessions may switch to using the encrypted channel, in which case the simulator will also pass the encrypted channel queries down to the tagged-ACCE-fin challenger.

Now the adversary may eventually ask the two honest parties to renegotiate. The simulator will construct the RIE extension by obtaining the `Finished` messages issuing a RevealFin query to the tagged-ACCE-fin challenger. Then the simulator will ask the parties to start a new session with those RIE extension values as the arbitrary data. Finally, it will encrypt those handshake messages using the Encrypt oracle of the existing session and give those ciphertexts to the adversary. If the adversary delivers the exact ciphertexts, then even though the simulator cannot decrypt the ciphertexts it can still carry out the handshake because it knows they are the right ciphertexts. If the adversary delivers modified ciphertexts, then the simulator rejects. This is the correct behaviour unless the adversary managed to forge ciphertexts in the underlying tagged-ACCE-fin.

Second, consider when the adversary is playing the role of a corrupted party with an honest party. In other words, the adversary has issued a Corrupt query for a long-term key of some party (which the simulator answered by issuing a Corrupt query to the corresponding party in the tagged-ACCE-fin challenger). For the initial handshake, the simulator simply relays the handshake messages down to the tagged-ACCE-fin challenger and returns the responses. However, once the handshake completes, the multi-phase ACCE simulator has no clue whether a ciphertext it receives contains a data message or a control (handshake) message, yet it needs to start a new handshake if it receives a control message. Fortunately, as soon as the initial handshake completes, the multi-phase ACCE simulator can issue a Reveal query to the underlying session in the ACCE challenger. And because the adversary is the peer in this phase, it will never be a valid session for the multi-phase ACCE authentication game or the multi-phase ACCE confidentiality/integrity game, and thus the simulator does not violate any freshness condition in the underlying tagged-ACCE-fin game by issuing a Reveal query.

The simulator $\mathcal{S}$. The details of the simulation follow. For each (party, public key) pair (i, pk) in the multi-phase ACCE experiment, $\mathcal{S}$ will allocate a distinct party, abstractly denoted $i|pk$, in the tagged-ACCE-fin experiment run by $\mathcal{C}$. Similarly, each phase ℓ in a session s in the multi-phase ACCE experiment will correspond to a session, abstractly denoted $s|\ell$, in the tagged-ACCE-fin experiment.

$\mathcal{S}$ answers the adversary's multi-phase ACCE queries as follows. In all of the following, let ℓ

be the current phase of π_i^s, let pk denote $\pi_i^s.\mathtt{phases}[\ell].pk$, pk^* denote $\pi_i^s.\mathtt{phases}[\ell+1].pk$, and suppose $\pi_i^s.\Pi = \pi_j^t$.

- $\mathsf{Send}(\pi_i^s, m)$: The behaviour of $\mathcal{S}$'s simulator of the Send oracle depends on whether the initial handshake or a renegotiation is occurring. First we consider the initial handshake:
  - If $m = (\mathtt{newphase}, pk, \alpha)$ and $\pi_i^s.\mathtt{phases}$ is empty: $\mathcal{S}$ at first issues a $\mathsf{Send}(\pi_{i|pk}^{s|1}, (\mathtt{newphase}, \alpha, \mathtt{empty}))$ query to $\mathcal{C}$ and returns the result.
  - If $m = (\mathtt{ready}, pk, \alpha)$ and $\pi_i^s.\mathtt{phases}$ is empty: $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk}^{s|1}, (\mathtt{ready}, \alpha, \mathtt{empty})$ query to $\mathcal{C}$; no result is received or returned.
  - If $m = m_1 = (r_C, \mathtt{cs\text{-}list}, ext_C)$ in Figure 6.3, $\mathcal{S}$ aborts if $ext_C \neq \mathtt{empty}$. Otherwise, $\mathcal{S}$ sends issues a $\mathsf{Send}(\pi_{i|pk}^{s|1}, (r_C, \mathtt{cs\text{-}list}))$ query to $\mathcal{C}$ and returns the result.
  - If $m = m_2 \| \dots \| m_6$ and $m_2 = (r_C, \mathtt{cs\text{-}list}, ext_S)$ in Figure 6.3, $\mathcal{S}$ aborts if $ext_S \neq \mathtt{empty}$. Otherwise, $\mathcal{S}$ sets $m_2' = (r_C, \mathtt{cs\text{-}list})$ and issues a $\mathsf{Send}(\pi_{i|pk}^{s|1}, m_2' \| \dots \| m_6)$ query to $\mathcal{C}$ and returns the result.
  - If $m = m_7 \| \dots \| m_{11}$ or $m = m_{12} \| m_{13}$, $\mathcal{S}$ relays the query to $\mathcal{C}$ and returns the result.

  Now consider renegotiation handshakes. For renegotiation handshakes, the only Send queries issued will involve `newphase` or `ready` messages.
  - If $m = (\mathtt{newphase}, pk^*, \alpha)$, $\pi_i^s.\mathtt{phases}$ is not empty, and $\pi_i^s.\rho = \mathsf{Client}$: $\mathcal{S}$ issues a $\mathsf{RevealFin}(\pi_{i|pk}^{s|\ell})$ query to $\mathcal{C}$ to obtain $fin_C \| fin_S$. $\mathcal{S}$ starts in `phases` a new phase $\ell+1$ of π_i^s with authentication mode α and public key pk^*. $\mathcal{S}$ obtains handshake message m_1^* by issuing a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, (\mathtt{newphase}, \alpha, fin_C))$ query to $\mathcal{C}$. $\mathcal{S}$ then issues an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, \mathtt{control}, m_1^*, m_1^*, \mathsf{len}, H)$ query to $\mathcal{C}$ and returns the result.
  - If $m = (\mathtt{newphase}, pk^*, \alpha)$, $\pi_i^s.\mathtt{phases}$ is not empty, and $\pi_i^s.\rho = \mathsf{Server}$: $\mathcal{S}$ starts in `phases` a new phase $\ell+1$ of π_i^s with authentication mode α and public key pk^*, sets $m_0^* = \mathtt{ServerHelloRequest}$, issues an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, \mathtt{control}, m_0^*, m_0^*, \mathsf{len}, H)$ query to $\mathcal{C}$, and returns the result.
  - If $m = (\mathtt{ready}, pk^*, \alpha)$ and $\pi_i^s.\mathtt{phases}$ is not empty: $\mathcal{S}$ starts in `phases` a new phase $\ell+1$ of π_i^s with authentication mode α and public key pk^*. $\mathcal{S}$ issues a $\mathsf{RevealFin}(\pi_{i|pk}^{s|\ell})$ query to $\mathcal{C}$ to obtain $fin_C \| fin_S$. $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, (\mathtt{ready}, \alpha, fin_S))$ query to $\mathcal{C}$; no result is received or returned.

  The actual handshake messages in renegotiation handshakes are delivered using Decrypt queries.

- $\mathsf{Corrupt}(P_i, pk)$: $\mathcal{S}$ issues a $\mathsf{Corrupt}(P_{i|pk})$ query to $\mathcal{C}$ and returns the result.

- $\mathsf{Reveal}(\pi_i^s, \ell)$: $\mathcal{S}$ issues a $\mathsf{Reveal}(\pi_{i|pk}^{s|\ell})$ query to $\mathcal{C}$ and returns the result.

- $\mathsf{Encrypt}(\pi_i^s, ctype, m_0, m_1, \mathsf{len}, H)$: $\mathcal{S}$ aborts if $ctype = \mathtt{control}$. Otherwise, $\mathcal{S}$ issues an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, m_0, m_1, \mathsf{len}, H)$ query to $\mathcal{C}$ and returns the result.

- $\mathsf{Decrypt}(\pi_i^s, C, H)$: First suppose that all of conditions **C2**–**C5** of Definition 3.13; namely that neither the phase's owner public key pk nor the peer's public key pk' has been corrupted, nor have the session keys been revealed. Now we explain each part of the renegotiation handshake:

– Suppose $\pi_i^s.\rho = \mathsf{Server}$ and the last query that π_i^s received was $\mathsf{Send}(\pi_i^s, (\mathtt{ready}, \ldots))$ or $\mathsf{Send}(\pi_i^s, (\mathtt{newphase}, \ldots))$. If C does not equal the last ciphertext that was sent by π_j^t, then abort. Otherwise, $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C, H)$ query to $\mathcal{C}$. (Note this returns $\perp$.)

  Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, m_1^*)$ query to $\mathcal{C}$, where m_1^* is the handshake message obtained by $\mathcal{S}$ from $\mathcal{C}$ in the $\mathsf{Send}(\pi_j^t, (\mathtt{newphase}, \ldots))$ query above. $\mathcal{S}$ receives $m_2^* \| \ldots \| m_6^*$ from $\mathcal{C}$, issuing an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, \mathtt{control}, m_2^* \| \ldots \| m_6^*, \ldots)$ query to encrypt them, and returns the resulting ciphertext C'. [30]

– Suppose $\pi_i^s.\rho = \mathsf{Client}$ and the last query that π_i^s received was $\mathsf{Send}(\pi_i^s, (\mathtt{ready}, \ldots))$. If C does not equal the last ciphertext that was sent by π_j^t, then abort. Otherwise, $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C, H)$ query to $\mathcal{C}$. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, m_2^* \| \ldots \| m_6^*)$ query to $\mathcal{C}$, where $m_2^* \| \ldots \| m_6^*$ are the handshake messages obtained by $\mathcal{S}$ from $\mathcal{C}$ in the $\mathsf{Decrypt}(\pi_j^t, \ldots)$ query in the bullet point immediately preceding this one. $\mathcal{S}$ receives $m_7^* \| \ldots \| m_{11}^*$ from $\mathcal{C}$, encrypts $m_7^* \| \ldots \| m_{10}^*$ by issuing an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, \mathtt{control}, m_7^* \| \ldots \| m_{10}^*, \ldots)$ query, and returns the resulting ciphertext C' along with m_{11}^*.

– Suppose $\pi_i^s.\rho = \mathsf{Server}$ and the last query that π_i^s received was the $\mathsf{Decrypt}$ query in the first bullet point in this list. If C does not equal the last ciphertext that was sent by π_j^t, then abort. Otherwise, $\mathcal{S}$ splits the ciphertext as $C^* \| m_{11}^*$ and issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C^*, H)$ query. (Note that this query returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, m_7^* \| \ldots \| m_{11}^*)$ query to $\mathcal{C}$, where $m_7^* \| \ldots \| m_{10}^*$ are the handshake messages obtained by $\mathcal{S}$ from $\mathcal{C}$ in the $\mathsf{Decrypt}(\pi_j^t, \ldots)$ query in the bullet point immediately preceding this one. $\mathcal{S}$ receives $m_{12}^* \| m_{13}^*$ from $\mathcal{C}$, encrypts m_{12}^* by issuing an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, \mathtt{control}, m_{12}^*, \ldots)$ query, and returns the resulting ciphertext C' along with m_{13}^*.

– Suppose $\pi_i^s.\rho = \mathsf{Client}$ and the last query that π_i^s received was the $\mathsf{Decrypt}$ query in the second bullet point in this list. If C does not equal the last ciphertext that was sent by π_j^t, then abort. Otherwise, $\mathcal{S}$ splits the ciphertext as $C^* \| m_{13}^*$ and issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C^*, H)$ query. (Note this returns $\perp$.)

  Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, m_{12}^* \| \ldots \| m_{13}^*)$ query to $\mathcal{C}$, where m_{12}^* is the handshake message obtained by $\mathcal{S}$ from $\mathcal{C}$ in the $\mathsf{Decrypt}(\pi_j^t, \ldots)$ query in the bullet point immediately preceding this one.

– All other $\mathsf{Decrypt}(\pi_i^s, \ldots)$ queries: $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C, H)$ query to $\mathcal{C}$ and returns the result.

Now suppose otherwise, namely that one or more of conditions **C2–C5** of Defintion 3.13 is violated, so either the phase owner's public key pk or the peer's public key pk' has been corrupted, or either $\mathsf{Reveal}(\pi_j^t, \ell')$ or $\mathsf{Reveal}(\pi_j^t, \ell')$ has been called.

If it has not already done so, $\mathcal{S}$ issues a $\mathsf{Reveal}(\pi_{i|pk}^{s|\ell})$ query to $\mathcal{C}$ and obtains session key k containing encryption and MAC keys $K_{\mathsf{enc}}^{C \to S} || K_{\mathsf{enc}}^{S \to C} || K_{\mathsf{mac}}^{C \to S} || K_{\mathsf{mac}}^{S \to C}$. Using the appro-

[30]Note that here, and throughout the $\mathsf{Decrypt}$ query, our simulator $\mathcal{S}$ is not disadvantaged by its call to $\mathcal{C}.\mathsf{Decrypt}$ not returning plaintext because, in this first part, it 'knows' what the plaintext handshake message is from having simulated the other side; and in the second part, it can reveal the session key and become able to decrypt the ciphertext itself.

priate encryption key, $\mathcal{S}$ steps through all previous calls to $\mathsf{Decrypt}(\pi_i^s, \dots)$ to bring the decryption state st_d up-to-date. Now we explain each part of the renegotiation handshake:

– Suppose $\pi_i^s.\rho = \mathsf{Server}$ and the last query that π_i^s received was $\mathsf{Send}(\pi_i^s, (\mathtt{ready}, \dots))$ or $\mathsf{Send}(\pi_i^s, (\mathtt{newphase}, \dots))$. $\mathcal{S}$ uses $K_{\mathsf{enc}}^{C\to S}$ and st_d to decrypt C and obtain m_1^* or $\perp$, in which case $\mathcal{S}$ aborts. $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C, H)$ query to $\mathcal{C}$. (Note this returns $\perp$.)

  Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, m_1^*)$ query to $\mathcal{C}$. $\mathcal{S}$ receives $m_2^*\| \dots \|m_6^*$ from $\mathcal{C}$, encrypts them by issuing an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, \mathtt{control}, m_2^*\| \dots \|m_6^*, \dots)$ query, and returns the resulting ciphertext C'.

– Suppose $\pi_i^s.\rho = \mathsf{Client}$ and the last query that π_i^s received was $\mathsf{Send}(\pi_i^s, (\mathtt{ready}, \dots))$. $\mathcal{S}$ then uses $K_{\mathsf{enc}}^{S\to C}$ and st_d to decrypt C and obtain $m_2^*\| \dots \|m_6^*$ or $\perp$, in which cases $\mathcal{S}$ aborts. $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C, H)$ query to $\mathcal{C}$. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, m_2^*\| \dots \|m_6^*)$ query to $\mathcal{C}$.

  $\mathcal{S}$ receives $m_7^*\| \dots \|m_{11}^*$ from $\mathcal{C}$, encrypts $m_7^*\| \dots \|m_{10}^*$ by issuing an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, \mathtt{control}, m_7^*\| \dots \|m_{10}^*, \dots)$ query, and returns the resulting ciphertext C' along with m_{11}^*.

– Suppose $\pi_i^s.\rho = \mathsf{Server}$ and the last query that π_i^s received was the $\mathsf{Decrypt}$ query in the first bullet point in this list. $\mathcal{S}$ splits the ciphertext as $C^*\|m_{11}^*$. $\mathcal{S}$ uses $K_{\mathsf{enc}}^{C\to S}$ and st_d to decrypt C^* and obtain $m_7^*\| \dots \|m_{10}^*$ or $\perp$, in which case $\mathcal{S}$ aborts. $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C^*, H)$ query to $\mathcal{C}$. (Note this returns $\perp$.)

  Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, m_7^*\| \dots \|m_{11}^*)$ query to $\mathcal{C}$. $\mathcal{S}$ receives $m_{12}^*\|m_{13}^*$ from $\mathcal{C}$, encrypts m_{12}^* by issuing an $\mathsf{Encrypt}(\pi_{i|pk}^{s|\ell}, \mathtt{control}, m_{12}^*, \dots)$ query, and returns the resulting ciphertext C' along with m_{13}^*.

– Suppose $\pi_i^s.\rho = \mathsf{Client}$ and the last query that π_i^s received was the $\mathsf{Decrypt}$ query in the second bullet point in this list. $\mathcal{S}$ splits the ciphertext as $C^*\|m_{13}^*$. $\mathcal{S}$ uses $K_{\mathsf{enc}}^{S\to C}$ and st_d to decrypt C^* and obtain m_{12}^* or $\perp$, in which case $\mathcal{S}$ aborts.

  $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C^*, H)$ query. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{i|pk^*}^{s|\ell+1}, m_{12}^*\| \dots \|m_{13}^*)$ query to $\mathcal{C}$.

– All other $\mathsf{Decrypt}(\pi_i^s, \dots)$ queries: $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{i|pk}^{s|\ell}, C, H)$ query to $\mathcal{C}$ and returns the result.

Correctness of the simulator. The simulation presented by $\mathcal{S}$ is perfect except in its handling of $\mathsf{Decrypt}$ queries when the peer's public key $\pi_i^s.pk$ has not been corrupted. During renegotiation, the simulator $\mathcal{S}$ rejects any ciphertext C that is not exactly equal to the ciphertext $C^{(-1)}$ sent by the peer in the previous query. It is possible that C is in fact a valid ciphertext. However, at the time the improper simulator occurred, all of conditions **C1**–**C5** of Definition 3.13 were satisfied. And under the assumption that C is a valid ciphertext, if $\mathcal{S}$ was to make a query $\mathsf{Decrypt}(\pi_i^s, C, H)$ to $\mathcal{C}$, then $\mathcal{S}$ would receive either $\perp$ if $\mathcal{C}$'s secret challenge bit b in $\pi_{i|pk}^{s|\ell}$ is 0, or $m \neq \perp$ if the secret challenge bit $b = 1$. In other words, $\mathcal{S}$ can (t', ϵ')-break the confidentiality of Π in the tagged-ACCE-fin experiment, with $\tau \approx \tau'$ and $\epsilon' \geq \epsilon_{\mathsf{tagged}}$.

Attack on a correct simulator. Finally, suppose no failure event happens as described above, and suppose the adversary breaks the multi-phase ACCE security of Π'. We show how to translate this attack into an attack on the ACCE-tag-fin security of Π in $\mathcal{C}$.

Confidentiality/integrity. First suppose that the multi-phase ACCE adversary for Π' succeeds in breaking confidentiality/integrity (Definition 3.13), namely by outputting a tuple (i, s, ℓ, b') for which $b' = \pi_i^s.\texttt{phases}[\ell].b$, and that adversary never violated conditions **C1**–**C6** of Definition 3.13. Let pk denote $\pi_i^s.\texttt{phases}[\ell].pk$. Since $\mathcal{S}$ only violates conditions **C1**–**C6** for $\pi_{i|pk}^{s|\ell}.\texttt{phases}[1]$ when the adversary violates them for $\pi_i^s.\texttt{phases}[\ell]$, $\mathcal{S}$ has not violated these conditions either. The simulator $\mathcal{S}$ outputs the tuple $((i|pk), (s|\ell), 1, b')$; it holds that $b' = \pi_{i|pk}^{s|\ell}.\texttt{phases}[1].b$ in Π. Thus, we have that $\mathcal{S}$ $(t', \epsilon_{\mathsf{c}})$-breaks the (tagged-ACCE-fin) confidentiality/integrity of Π, where $\tau \approx \tau'$ and ϵ_{c} is obtained from ϵ by replacing all instances of ϕ in ϵ with $\phi \cdot \gamma$ and replacing all instances of η in ϵ with $\eta \cdot \psi$.

Authentication. Now suppose that the multi-phase ACCE adversary for Π' succeeds in breaking authentication (as in Definition 3.14, namely by causing to exist a phase $\pi_i^s.\texttt{phases}[\ell]$ which has accepted (condition **A1**), have not been trivially compromised (conditions **A2**–**A5**), and have no phase with a matching handshake transcript at the peer (condition **M**). Since $\mathcal{S}$ only violates conditions **A2**, **A3**, or **A5** for $\pi_{i|pk}^{s|\ell}.\texttt{phases}[1]$ when the adversary violates them for $\pi_i^s.\texttt{phases}[\ell]$, $\mathcal{S}$ has not violated these conditions either. Moreover, condition **A4** for $\pi_{i|pk}^{s|\ell}.\texttt{phases}[1]$ is satisfied precisely when $\pi_i^s.\texttt{phases}[\ell]$ is. Thus, we have that $\mathcal{S}$ $(t', \epsilon_{\mathsf{a}})$-breaks the (tagged-ACCE-fin) authentication of Π, where $\tau \approx \tau'$ and ϵ_{a} is obtained from ϵ by replacing ϕ and η in ϵ as in the previous paragraph.

The result follows. □

Remark 21. A simulation similar to the one in the proof allows us to prove that TLS with SCSV/RIE countermeasures is a multi-phase ACCE protocol, even when different ciphersuites are used in different phases. The simulator interacts with a different tagged-ACCE-fin challenger for each ciphersuite; when a renegotiation inside one ciphersuite will result in a new ciphersuite, the simulator uses the Encrypt/Decrypt queries in the old ciphersuite to encrypt the Send messages from the handshake of the new ciphersuite. Unfortunately, for this multi-ciphersuite simulation to work, it is essential that public keys not be shared across ciphersuites: this technique could show that switching between an RSA-based ciphersuite and an ECDSA-based ciphersuite is safe. However, to analyze using the same RSA public key in two different ciphersuites, one would have to take an alternative approach, as it may not be possible to generically prove that re-using the same public key in two ACCE protocols is safe.

6.7 TLS with SCSV/RIE is a Weakly Secure Renegotiable ACCE

We are now in a position to show that the use of the SCSV/RIE countermeasure in TLS results in a weakly secure renegotiable ACCE. We will do so generically, starting from the consequence of the previous theorem: that TLS with SCSV/RIE is a secure multi-phase ACCE.

Theorem 6.3 (TLS with SCSV/RIE is a weakly secure renegotiable ACCE)**.** *Let* Π *be a TLS ciphersuite with SCSV/RIE countermeasures, as described in Figure 6.3. If* Π *is a* $(t, \epsilon_{\mathsf{mp}})$-

secure multi-phase ACCE protocol, and PRF *is a* $(t, \epsilon_{\mathsf{prf}})$*-secure pseudorandom function, then* Π *is a* (t, ϵ)*-weakly secure renegotiable ACCE, with* $\epsilon = \epsilon_{\mathsf{mp}} + \epsilon_{\mathsf{prf}}$.

Intuitively, the use of the RIE countermeasure guarantees that each party who renegotiates has the same view of (a) whether they are renegotiating, and (b) which handshake is the 'previous' handshake. We can chain these together to obtain the property of a secure renegotiable ACCE: parties who renegotiate have the same view of all previous handshakes. If this is violated, either the non-renegotiable aspects of TLS have been broken, or a collision has been found in the computation of the renegotiation indication extension.

Proof. Suppose $\mathcal{A}$ breaks the weak renegotiable ACCE security of the protocol Π. We will show that either $\mathcal{A}$ directly breaks the multi-phase ACCE security of Π or $\mathcal{A}$ can be used to construct another algorithm that breaks either the security of the PRF or the multi-phase ACCE security of Π.

We approach the proof in three cases: either $\mathcal{A}$ has broken the confidentiality/integrity of the weakly secure renegotiable ACCE, or $\mathcal{A}$ has broken the weak renegotiation authentication of the weakly secure renegotiable ACCE, and the latter can happen by meeting either condition $\mathbf{M'(a)}$ or $\mathbf{M'(b)}$.

Confidentiality/integrity. Since the winning conditions for the confidentiality/ integrity part of the security game are the same for both definitions, every adversary who breaks confidentiality/integrity in the weakly secure renegotiable ACCE security game for Π directly breaks confidentiality/integrity in the multi-phase ACCE security game for Π.

Authentication — $\mathbf{M'(a)}$. Suppose $\mathcal{A}$ wins the weak renegotiable ACCE security experiment for Π using condition $\mathbf{M'(a)}$. Either there is no ℓ at all such that $\pi_j^t.\mathtt{phases}[\ell].T$ matches $\pi_i^s.\mathtt{phases}[\ell^*].T$, or there is such an ℓ but $\ell \neq \ell^*$.

First consider the case where there exists no phase ℓ at all such that $\pi_j^t.\mathtt{phases}[\ell].T$ matches $\pi_i^s.\mathtt{phases}[\ell^*].T$. That meets condition $\mathbf{M}$ of Definition 3.14 for Π.

Now consider the case where there is an ℓ such that $\pi_j^t.\mathtt{phases}[\ell].T$ matches $\pi_i^s.\mathtt{phases}[\ell^*].T$ but $\ell \neq \ell^*$. Assume without loss of generality $\ell < \ell^*$ (otherwise we could swap the oracles).

There must exist some value $n \in [\ell - 1]$ such that $\pi_i^s.\mathtt{phases}[\ell^* - n].T \neq \pi_j^t.\mathtt{phases}[\ell - n].T$. In particular, $j \leq \ell - 1$, since in π_j^t's first phase its outgoing message m_1 contains $ext_C = \mathtt{empty}$ but π_i^s received a message m_1 with $ext_c \neq \mathtt{empty}$. Let j be minimal. Then $\pi_j^t.\mathtt{phases}[\ell - n + 1].T$ matches $\pi_i^s.\mathtt{phases}[\ell^* - n + 1].T$. In particular, messages m_1 of those two transcripts are equal, and so are messages m_2 of those two transcripts. Since RIE is being used, m_1 and m_2 contain $\mathsf{fin}_C^{(-1)}$ and $\mathsf{fin}_S^{(-1)}$, and since π_i^{s,ℓ^*-n+1} accepted, both π_i^{s,ℓ^*-n+1} and $\pi_j^{t,\ell-n+1}$ used the same $\mathsf{fin}_C^{(-1)}$ and $\mathsf{fin}_S^{(-1)}$ values. But at each party, $\mathsf{fin}_C^{(-1)}$ and $\mathsf{fin}_S^{(-1)}$ are the hash (using a PRF) of the handshake transcripts from phases π_i^{s,ℓ^*-n} and $\pi_j^{t,\ell-n}$, and we know that these handshake transcripts are not equal. This means a collision has occurred in PRF, which happens with negligible probability.

Thus, assuming PRF is secure and Π is a secure multi-phase ACCE, no $\mathcal{A}$ can achieve conditions $\mathbf{M'(a)}$ and $\mathbf{A1}$–$\mathbf{A7}$.

Authentication — $\mathbf{M'(b)}$. Now suppose $\mathcal{A}$ wins the weak renegotiable ACCE security experiment for Π using condition $\mathbf{M'(b)}$ but not $\mathbf{M'(a)}$. In particular, for every $\ell' < \ell^*$,

$\pi_i^s.\mathtt{phases}[\ell'].T = \pi_j^t.\mathtt{phases}[\ell'].T$ *but* there is some $\ell < \ell^*$ such that $\pi_i^s.\mathtt{phases}[\ell].\mathsf{RT}_s \| \mathsf{RT}_r \neq \pi_j^t.\mathtt{phases}[\ell].\mathsf{RT}_r \| \mathsf{RT}_s$. Choose ℓ minimal. Let v be the smallest index such that the vth ciphertext C_v of $\pi_i^s.\mathtt{phases}[\ell].\mathsf{RT}_s \| \mathsf{RT}_r$ is not equal to the vth ciphertext of $\pi_j^t.\mathtt{phases}[\ell].\mathsf{RT}_r \| \mathsf{RT}_s$.

Assume without loss of generality that C_v was received by π_i^s as the vth ciphertext but was not sent by π_j^t as the vth ciphertext. (The alternative is that C_v was *sent* by π_i^s as the vth ciphertext but was not received by π_j^t as the vth ciphertext. However, we could then focus on everything from π_j^t's perspective and apply the same argument.)

This means that when $\mathcal{A}$ called $\mathsf{Decrypt}(\pi_i^s, C_v, H)$, if $b = 0$ then $\mathsf{Decrypt}$ returned $(\perp, \cdot)$, whereas if $b = 1$ then $\mathsf{Decrypt}$ returned $(m', \cdot)$ where $m' \neq \perp$. Our simulator can thus output (i, s, ℓ, b') for its guess of b' as above, and this will equal b with probability at least ϵ, making condition **C6** hold in Definition 3.14. We need to show that conditions **C1**–**C5** also hold for (i, s, ℓ).

Since $\mathcal{A}$ wins the weak renegotiable ACCE experiment using condition $\mathbf{M'(b)}$, we have that **A1**–**A7** all hold. We want to show that, at the time that π_i^s accepted in phase $\ell + 1$, conditions **C1**–**C5** also hold for (i, s, ℓ).

- **C1**: **A1** directly implies **C1**, since if π_i^s has rejected in any phase prior to ℓ^* then it would not have a phase ℓ^*.
- **C2** and **C3**: Conditions **A2** and **A3** of Definition 3.16 do not imply that $\mathcal{A}$ did not ask $\mathsf{Corrupt}$ queries prohibited by **C2** and **C3**. However, we do have that $\pi_i^s.\mathtt{phases}[\ell].T = \pi_j^t.\mathtt{phases}[\ell].T$; in other words, $\mathcal{A}$ was not *active* in the handshake for phase ℓ. Thus, $\mathcal{A}$ is *equivalent* to an adversary who did not ask any $\mathsf{Corrupt}$ queries for public keys used in phase ℓ until after π_i^s accepts in phase ℓ.
- **C4**: **A6** directly implies **C4**, at the time that π_i^s accepted.
- **C5**: Since π_i^s chooses nonce r_C (if a client) or r_S (if a server) randomly, except with negligible probability there is no $\ell' < \ell$ such that $\pi_i^s.\mathtt{phases}[\ell'].T = \pi_i^s.\mathtt{phases}[\ell].T$. By **A7**, $\mathcal{A}$ did not issue $\mathsf{Reveal}(\pi_j^t, \ell)$ before π_i^s accepted in phase $\ell + 1$. Thus at the time that π_i^s accepted, $\mathcal{A}$ did not issue $\mathsf{Reveal}(\pi_j^t, \ell')$ to any phase with $\pi_j^t.\mathtt{phases}[\ell'].T = \pi_i^s.\mathtt{phases}[\ell].T$, satisfying condition **C5**.

Thus, assuming Π is a secure multi-phase ACCE no $\mathcal{A}$ can achieve conditions $\mathbf{M'(b)}$ and **A1**–**A7**. □

We can combine Theorems 6.2 and 6.3 to obtain the central result of the paper, justifying the security of the SCSV/RIE countermeasure:

Corollary 1 (TLS with SCSV/RIE is a weakly secure renegotiable ACCE)**.** *If a tagged TLS ciphersuite* Π *is a secure tagged-ACCE-fin protocol as described in Section 3.2.3 and* PRF *is a secure pseudorandom function, then that TLS ciphersuite* Π *with SCSV/RIE countermeasures as described in Figure 6.3 is a weakly secure renegotiable ACCE.* □

For concreteness, we can combine this with Theorem 6.1 to obtain:

Corollary 2 (TLS-DHE with SCSV/RIE is a weakly secure renegotiable ACCE)**.** *Under the same assumptions on the building blocks as in Theorem 6.1, TLS-DHE with SCSV/RIE countermeasures is a weakly secure renegotiable ACCE protocol.* □

6.7.1 On Renegotiation Security of TLS-RSA with SCSV/RIE

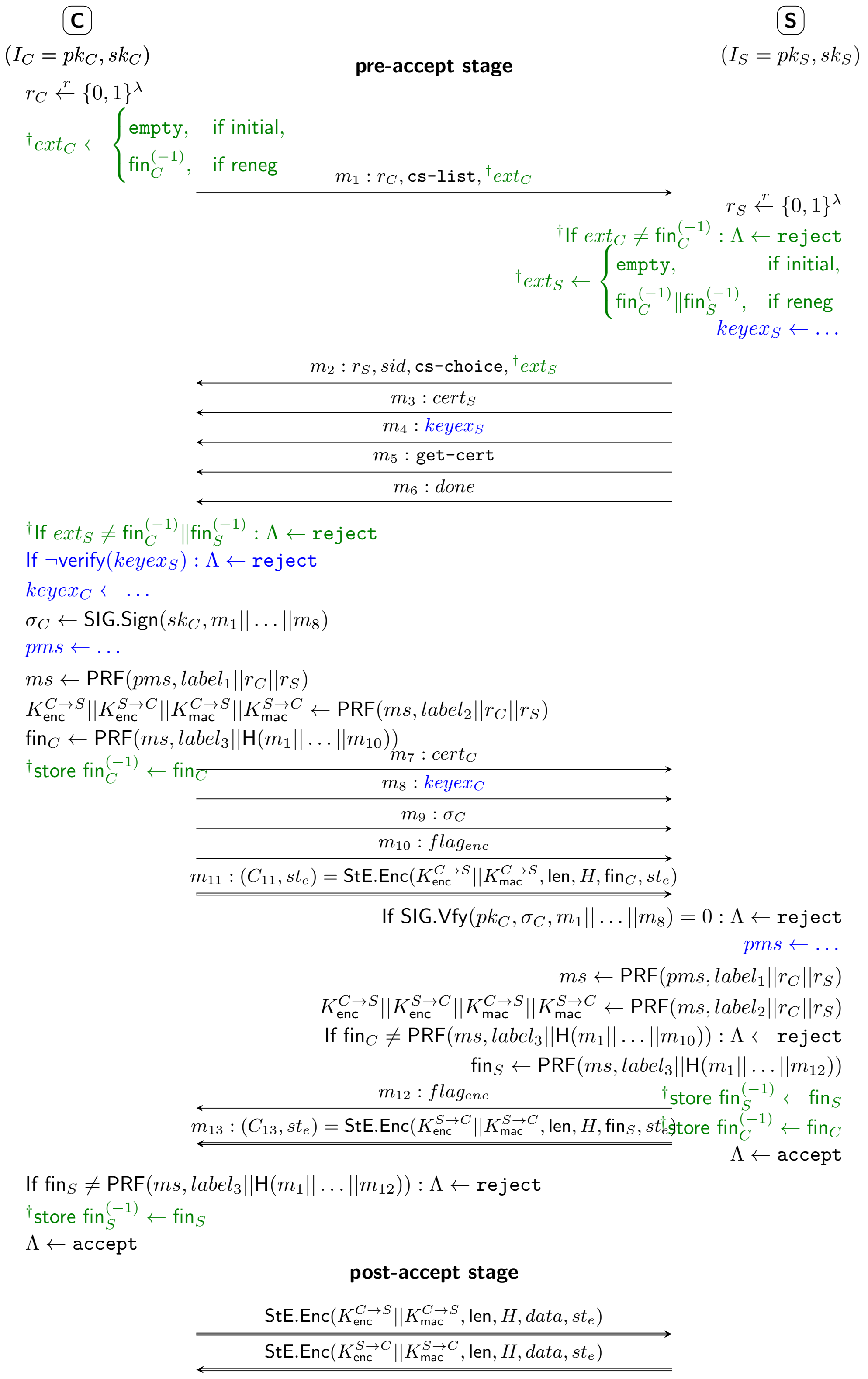
C
S
$(I_C = pk_C, sk_C)$
pre-accept stage
$(I_S = pk_S, sk_S)$
$r_C \xleftarrow{r} \{0,1\}^\lambda$
$^\dagger ext_C \leftarrow \begin{cases} \texttt{empty}, & \text{if initial,} \\ \mathsf{fin}_C^{(-1)}, & \text{if reneg} \end{cases}$
$m_1 : r_C, \texttt{cs-list}, {}^\dagger ext_C$
$r_S \xleftarrow{r} \{0,1\}^\lambda$
†If $ext_C \neq \mathsf{fin}_C^{(-1)} : \Lambda \leftarrow \texttt{reject}$
$^\dagger ext_S \leftarrow \begin{cases} \texttt{empty}, & \text{if initial,} \\ \mathsf{fin}_C^{(-1)} \| \mathsf{fin}_S^{(-1)}, & \text{if reneg} \end{cases}$
$keyex_S \leftarrow \dots$
$m_2 : r_S, sid, \texttt{cs-choice}, {}^\dagger ext_S$
$m_3 : cert_S$
$m_4 : keyex_S$
$m_5 : \texttt{get-cert}$
$m_6 : done$
†If $ext_S \neq \mathsf{fin}_C^{(-1)} \| \mathsf{fin}_S^{(-1)} : \Lambda \leftarrow \texttt{reject}$
If $\neg\mathsf{verify}(keyex_S) : \Lambda \leftarrow \texttt{reject}$
$keyex_C \leftarrow \dots$
$\sigma_C \leftarrow \mathsf{SIG.Sign}(sk_C, m_1 \| \dots \| m_8)$
$pms \leftarrow \dots$
$ms \leftarrow \mathsf{PRF}(pms, label_1 \| r_C \| r_S)$
$K_{\mathsf{enc}}^{C\to S} \| K_{\mathsf{enc}}^{S\to C} \| K_{\mathsf{mac}}^{C\to S} \| K_{\mathsf{mac}}^{S\to C} \leftarrow \mathsf{PRF}(ms, label_2 \| r_C \| r_S)$
$\mathsf{fin}_C \leftarrow \mathsf{PRF}(ms, label_3 \| \mathsf{H}(m_1 \| \dots \| m_{10}))$
†store $\mathsf{fin}_C^{(-1)} \leftarrow \mathsf{fin}_C$
$m_7 : cert_C$
$m_8 : keyex_C$
$m_9 : \sigma_C$
$m_{10} : flag_{enc}$
$m_{11} : (C_{11}, st_e) = \mathsf{StE.Enc}(K_{\mathsf{enc}}^{C\to S} \| K_{\mathsf{mac}}^{C\to S}, \mathsf{len}, H, \mathsf{fin}_C, st_e)$
If $\mathsf{SIG.Vfy}(pk_C, \sigma_C, m_1 \| \dots \| m_8) = 0 : \Lambda \leftarrow \texttt{reject}$
$pms \leftarrow \dots$
$ms \leftarrow \mathsf{PRF}(pms, label_1 \| r_C \| r_S)$
$K_{\mathsf{enc}}^{C\to S} \| K_{\mathsf{enc}}^{S\to C} \| K_{\mathsf{mac}}^{C\to S} \| K_{\mathsf{mac}}^{S\to C} \leftarrow \mathsf{PRF}(ms, label_2 \| r_C \| r_S)$
If $\mathsf{fin}_C \neq \mathsf{PRF}(ms, label_3 \| \mathsf{H}(m_1 \| \dots \| m_{10})) : \Lambda \leftarrow \texttt{reject}$
$\mathsf{fin}_S \leftarrow \mathsf{PRF}(ms, label_3 \| \mathsf{H}(m_1 \| \dots \| m_{12}))$
$m_{12} : flag_{enc}$
†store $\mathsf{fin}_S^{(-1)} \leftarrow \mathsf{fin}_S$
$m_{13} : (C_{13}, st_e) = \mathsf{StE.Enc}(K_{\mathsf{enc}}^{S\to C} \| K_{\mathsf{mac}}^{S\to C}, \mathsf{len}, H, \mathsf{fin}_S, st_e)$
†store $\mathsf{fin}_C^{(-1)} \leftarrow \mathsf{fin}_C$
$\Lambda \leftarrow \texttt{accept}$
If $\mathsf{fin}_S \neq \mathsf{PRF}(ms, label_3 \| \mathsf{H}(m_1 \| \dots \| m_{12})) : \Lambda \leftarrow \texttt{reject}$
†store $\mathsf{fin}_S^{(-1)} \leftarrow \mathsf{fin}_S$
$\Lambda \leftarrow \texttt{accept}$
post-accept stage
$\mathsf{StE.Enc}(K_{\mathsf{enc}}^{C\to S} \| K_{\mathsf{mac}}^{C\to S}, \mathsf{len}, H, data, st_e)$
$\mathsf{StE.Enc}(K_{\mathsf{enc}}^{S\to C} \| K_{\mathsf{mac}}^{S\to C}, \mathsf{len}, H, data, st_e)$

Figure 6.3: Generic TLS handshake protocol with †SCSV / RIE renegotiation countermeasures

As the security definitions for multi-phase and renegotiation protocols in Section 3.2.3 only cover protocols providing forward secrecy, TLS ciphersuites without forward secrecy, e.g. RSA key transport-based ciphersuites, cannot be shown to be secure or weakly secure renegotiable ACCE protocols. However, it is plausible that the definition can be modified to consider protocols without forward secrecy, in which case RSA key transport may be able to proven secure. In such a scenario, it should be possible to show that such ciphersuites, when using SCSV/RIE, also satisfy a non-forward-secure notion of weak renegotiation security; or when using the new countermeasure in Section 6.8, also satisfy a non-forward-secure notion of renegotiation security. A diagram showing the message flow for a generic TLS ciphersuite with SCSV/RIE countermeasures appears in Figure 6.3.

6.8 New Countermeasure for TLS Renegotiation

We now present a new TLS renegotiation countermeasure that provides integrity protection for the record layer transcript upon renegotiation (even when previous phases' session keys are leaked while the phase is still active), thereby achieving the full security of Definition 3.15. This countermeasure is quite straightforward: by including a hash of all record layer messages in the renegotiation information extension, parties can confirm that they share the same view of their previous record layers.

The renegotiation information extension already contains a fingerprint of the previous phrase's handshake transcript via the `client_verify_data` ($\mathsf{fin}_C^{(-1)}$)[31] and `server_verify_data` ($\mathsf{fin}_S^{(-1)}$) values. We modify the renegotiation information extension to include an additional value, the fingerprint of the encrypted messages sent over the previous phase's record layer. In particular, if negotiating:

$$ext_C \leftarrow \mathsf{fin}_C^{(-1)} \parallel \mathsf{PRF}(ms^{(-1)}, label_5 \| H(\mathsf{RT}_s^{(-1)} \| \mathsf{RT}_r^{(-1)})) \ , \tag{6.1}$$

where $ms^{(-1)}$ is the previous phase's master secret, H is a collision-resistant hash function, and $\mathsf{RT}_s^{(-1)} \| \mathsf{RT}_r^{(-1)}$ is the client's view of the previous phase's record layer transcript; the server uses $\mathsf{RT}_r^{(-1)} \| \mathsf{RT}_s^{(-1)}$ instead. Appropriate checks are performed by the server. With this additional information, the two parties will now not complete renegotiation unless they have matching views of the record layer transcripts from the previous phase.

Remark 22. Note that the proof does not require the server to also send its view of the record layer transcript; the server simply checks what it receives from the client and stops if it is not what it expects. The same is actually true as well of the RIE countermeasure, and the proof of Theorem 6.3 would go through if only ext_C contained $\mathsf{fin}_S^{(-1)}$. However, if the security model is altered to allow $\mathsf{Corrupt}$s of the current phase's public keys but not Reveals of the previous phase's session keys, then having both ext_C and ext_S include each party's view of the the transcript is required to achieve security.

In practice, it is not difficult to, on an incremental basis, compute hashes of the ciphertexts sent and received over the record layer in that phase. In particular, it is not necessary to store all record layer messages to input to the hash function all at once, as common programming APIs for hash functions allow the hash value to be provided incrementally. However, the cost of the MAC computation can dominate the cryptographic cost of record layer computations [GSF+04].

[31] $\mathsf{fin}_X^{(-1)}$ indicates a `Finished` message from the previous phase

Alternatively, if the sLHAE scheme for the record layer is implemented as an encrypt-then-MAC or MAC-then-encrypt, it should be possible to use MAC contained in the last encrypted message of the sLHAE scheme instead of the hash value computed above; this would result in no additional performance impact and would be easier to implement.

Theorem 6.4 (TLS with new countermeasure is a secure renegotiable ACCE)**.** *Let* Π *be a TLS ciphersuite with the original RIE countermeasures as in Figure 6.3 but using* ext_C *as in equation (6.1). If* Π *is a* $(t, \epsilon_{\mathsf{mp}})$*-secure multi-phase ACCE protocol,* H *is a* $(t, \epsilon_{\mathsf{H}})$*-collision-resistant hash function, and* PRF *is a* $(t, \epsilon_{\mathsf{prf}})$*-secure pseudorandom function, then* Π *is a* (t, ϵ)*-secure renegotiable ACCE, where* $\epsilon = \epsilon_{\mathsf{mp}} + \epsilon_{\mathsf{H}} + \epsilon_{\mathsf{prf}}$*.*

The proof proceeds similarly to that of Theorem 6.3. The main difference is that, in one case, the removal of restrictions **A6** and **A7** means we can no longer reduce down to a violation of confidentiality/integrity in the multi-phase security of Π, and instead have to rely on the new countermeasure to detect non-matching record layer transcripts and reduce to the security of the PRF and hash function.

Proof. The proof proceeds similarly to that of Theorem 6.3. Suppose $\mathcal{A}$ breaks the renegotiable ACCE security of the protocol Π. We will show that either $\mathcal{A}$ directly breaks the multi-phase ACCE security of Π or $\mathcal{A}$ can be used to construct another algorithm that breaks either the security of thePRF, the collision resistance of H, or the multi-phase ACCE security of Π.

As before, we approach the proof in three cases: either $\mathcal{A}$ has broken the confidentiality/integrity of the secure renegotiable ACCE, or $\mathcal{A}$ has broken the renegotiation authentication of the secure renegotiable ACCE, and the latter can happen by meeting either condition $\mathbf{M'(a)}$ or $\mathbf{M'(b)}$. The first two cases, confidentiality and authentication for $\mathbf{M'(a)}$, proceed exactly as in the proof of Theorem 6.3.

Authentication — $\mathbf{M'(b)}$. Suppose $\mathcal{A}$ wins the renegotiable ACCE security experiment for Π using condition $\mathbf{M'(b)}$ but not $\mathbf{M'(a)}$.

When conditions **A1**–**A7** hold, then the same argument as in the proof for case $\mathbf{M'(b)}$ of Theorem 6.3 still holds, in which case conditions **C1**–**C6** hold and $\mathcal{A}$ can be used to break the confidentiality/integrity of Π in the multi-phase ACCE experiment.

However, in the secure renegotiable ACCE experiment, the adversary is no longer constrained by conditions **A6** and **A7**, so it can make Reveal queries while the phase is active. This means that conditions **C4** and **C5** are no longer satisfied, so we cannot reduce to the confidentiality/integrity of Π in the multi-phase ACCE experiment when such Reveal queries are issued. Instead, we make use of the new countermeasure, and apply an argument similar to that for case $\mathbf{M'(a)}$ of Theorem 6.3.

If $\mathcal{A}$ wins using condition $\mathbf{M'(b)}$ but not $\mathbf{M'(a)}$, then, for every phase $\ell' < \ell^*$, holds that $\pi_i^s.\mathtt{phases}[\ell'].T = \pi_j^t.\mathtt{phases}[\ell'].T$. But by definition there exists a phase $\ell < \ell^*$ such that $\pi_i^s.\mathtt{phases}[\ell].\mathsf{RT}_s \| \mathsf{RT}_r \neq \pi_j^t.\mathtt{phases}[\ell].\mathsf{RT}_r \| \mathsf{RT}_s$. Choose ℓ *maximal.* Then $\pi_j^t.\mathtt{phases}[\ell+1].T$ matches $\pi_i^s.\mathtt{phases}[\ell+1].T$. In particular, messages m_1 of those two transcripts are equal, and so are messages m_2 of those two transcripts. Since the new countermeasure is being used, m_2 contains ext_C, which itself contains the output of $\mathsf{PRF}(ms^{(-1)}, label_5 \| H(\mathsf{RT}_s^{(-1)} \| \mathsf{RT}_r^{(-1)}))$. Since $\pi_i^{s,\ell+1}$ accepted, both $\pi_i^{s,\ell+1}$ and $\pi_j^{t,\ell+1}$ used the same value in ext_C. But at each party, this value is the value of the PRF applied to the hash of the record layer transcripts from phases

$\pi_i^{s,\ell}$ and $\pi_j^{t,\ell}$, which we know are not equal. This means a collision has occurred either in H or PRF, which happens with negligible probability.

Thus, assuming PRF is secure, H is collision-resistant, and Π is a secure multi-phase ACCE, no $\mathcal{A}$ can achieve conditions $\mathbf{M'(a)}$ and **A1**–**A5**. $\square$

7 Conclusion

In this chapter we conclude the thesis by discussing our main results. We first discuss our generic construction of secure AKE protocols. Then we discuss our results on the TLS protocol, and finally our results on the security of multi-phase and renegotiation protocols.

7.1 Discussion of our Generic Compiler for AKE Protocols

Our construction described in Chapter 4 combines a Key Exchange (KE) protocol with a digital signature scheme (and one-way functions) to achieve an authenticated key exchange protocol. To protect against active Man-In-The-Middle (MITM) adversaries in our generic scenario, it seems sufficient to simply include the transcript of the KE protocol into the authentication protocol, such that any modification of messages in the KE protocol is automatically detected in the authentication protocol (since the transcript is included). Unfortunately, this simple construction cannot be proven secure in the BR model. An adversary has the same access to the transcript of the protocol, and can modify messages in the authentication protocol in a specific way to perform Unknown Key Share Attacks (UKS) as described in Section 1.4. To avoid this attack, a secret value only known to the communication partner of the key exchange protocol (i.e. the session key) must be embedded in the authentication protocol in a generic way.

In our compiler, we send an additional pair of messages after the key exchange and the authentication protocol. These messages contain a cryptographic checksum over the transcripts of both protocols. This checksum is basically a MAC, computed over the transcript of both the KE and the authentication protocol, using a key derived from the negotiated session key (which is returned by the key exchange protocol) and a Pseudo-Random Function.

An alternative would be to modify a value that is present in *all* secure authentication protocols, in such a way that it does not change the security properties of the protocol: In a generic authentication protocol, a random challenge r_A guaranteeing the freshness of the message(s) must be sent from party A to party B, which is answered with a response s_B from B. Instead of using the challenge r_A directly to compute the response, one could use a derived value r'_A from the same distribution:

$$r'_A := \mathsf{H}(k, r_A, r_B, transcript_{KE}),\ s'_B := \mathsf{SIG.Sign}(sk_B, r'_A),$$

where H is a hash function modeled as a Random Oracle and k is the key that is output by the underlying KE protocol. Please note that r'_A is never sent but has to be computed by A and B. Thus the adversary E does not learn r'_A. This construction does not alter the security properties of the authentication protocol. However, a security proof for this compiler would require the Random Oracle model, so this construction is left out-of-scope.

7.2 Discussion of the Results on TLS

The design of the TLS-DHE Handshake seems to support our proof idea of reducing active adversaries to passive adversaries very naturally. This is not due to the key transport mechanism itself, but rather to the fact that Ephemeral Diffie-Hellman (DHE) uses signatures computed over all previously exchanged messages to authenticate the protocol parties. Our proof essentially exploits that this authentication mechanism at the same time also protects the first phase of the TLS-DHE Handshake from adversarial modifications. We cannot identify a similarly straight-forward approach for encrypted key transport, as server authentication is done rather implicitly when verifying the `Finished` messages. Likewise, the static Diffie-Hellman key exchange does not use signatures over the first Handshake messages and our proof technique does not apply.

The whole TLS protocol suite is much more complex than the cryptographic protocol underlying TLS-DHE. It is very flexible, as it allows to negotiate ciphersuites at the beginning of the TLS Handshake, or to resume sessions using an abbreviated TLS Handshake. We need to leave an analysis of these features for future work, since the complexity of the protocol and security model grows dramatically.

The goal of this analysis is to make security statements about TLS-DHE, TLS-SDH and TLS-RSA on the protocol layer and give rise to new constructions which are provably secure under standard assumptions solely. As common in cryptographic protocol analyzes, we therefore have ignored implementational issues like error messages, which of course might also be used to break the security of the protocol. We leave it as an interesting open question to find an adequate approach for modeling such side-channels in complex scenarios like AKE involving many parties and parallel, sequential, and concurrent executions. Another important open problem is to consider cross-protocol attacks that exploit for instance possible subtle connections between different ciphersuites. While the example of [WS96] is impractical (and in this case rather an implementational issue) there may be other even more sophisticated attacks.

We consider our results as a strong indicator for the soundness of the TLS-DHE protocol. We also believe that future revisions of the TLS standard should be guided by provable security results – ideally in the standard model.

Limitations of our Results on TLS

STANDARD MODEL SECURITY OF RSA Our results do have some important limitations which we want to point out here. First, for TLS-RSA *we need to require that the underlying public key encryption system is CCA secure.* This stands in line with previous works like [GMP$^+$08b, BFS$^+$12]. However the TLS-RSA ciphersuites all rely on RSA-PKCS#1 v1.5 which is not CCA secure [Ble98], thus strictly speaking our result does not apply to the current state of TLS-RSA. However, we believe that our results are still meaningful and useful for the understanding of TLS and the future development of the TLS standard. They show that changing the encryption system to a provably secure one pays off in the sense that it allows to obtain a strong security result for the *entire* TLS protocol.

We stress that although standardized in PKCS#1 v2.1, TLS does *not* allow to use RSA-OAEP [BR94b] because of ‘maximal compatibility with earlier versions of TLS’ [DR08]. Note that RSA-OAEP is secure against Chosen-Plaintext Attacks (CPA) under the RSA assumption in the Random Oracle Model only [FOPS01, KP09] — in the standard model it is only known to be CPA secure under the Φ-Hiding assumption [KOS10] and the assumption that the employed hash functions are t-wise independent.

Assuming new Primitives for TLS Our results show that if we replace the encryption system and the signature scheme with primitives provably secure in the standard model[32], we can obtain provably secure results for TLS that avoid Random Oracles at all. This again can guide future revisions of the TLS standards, although it is obvious that, for compatibility reasons, the primitives in TLS should not be simply exchanged with new ones. In general, we support a careful transition to a more modular structure of TLS in line with one of the most important goals of the TLS protocol given in the specification of TLS 1.2 [DR08] – extensibility.

The standard is specifically explicit with respect to the support of new public key encryption mechanisms:

'Extensibility: TLS seeks to provide a framework into which new public key and bulk encryption methods can be incorporated as necessary'.

We emphasize that in the past there have already been modifications of TLS that tend into this direction. For example, while the PRF used in TLS 1.1 is fixed for all ciphersuites, TLS 1.2 allows to use 'cipher-suite-specified PRFs'. Similarly, we might regard the introduction of Elliptic Curve Cryptography (ECC) into the SSL/TLS protocol stack as a positive example of an extension of the TLS standard [BWBG$^+$06]: more and more servers support and use the efficient ECC-based ciphersuites as default, the most prominent example probably being Google that switched to Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) in 2011, to support and push the use of TLS handshakes with forward secrecy [Ada]. To us, our results not only propose (and analyze) an alternative instantiation of TLS with different primitives. They rather help to lead the process of selecting new, adequate building blocks into a wortwhile direction (towards provable security in the standard model under standard assumptions).

7.3 Discussion of the Results on Multi-Phase and Renegotiation Protocols

This thesis only makes use of the multi-phase ACCE and renegotiable ACCE security models to give proofs of the TLS renegotiation protocol. However, our definitions are very general and not specifically constructed or limited to TLS renegotiation. We think that the three different security notions capture the essence of renegotiation protocols in general and can be of independent interest for analyzing other protocols that allow for renegotiation.

Let us shortly sum up the different security notions and discuss their field of application. The first notion of multi-phase ACCE protocols is arguably the weakest, as it does not require a cryptographic link between different protocol phases. It only ensures, that each single phase yields a secure ACCE protocol in that for each phase in which a party 'Alice' accepts (thinking that her communication partner is 'Bob') there exists a 'corresponding' phase for the party 'Bob'. We do not require that it has to be the same phase, such that attacks similar to the renegotiation attack by Ray and Dispensa as described in Section 6.1 are not captured by

[32]Natural candidates secure under the RSA assumption are [HK09b] for public key encryption and [HW09] for signature generation.

this model. We stress that this attack would not have any impact in the real world if the application running on the server would distinguish between messages from different phases instead of just concatenating them and evaluating them completely in the security context of the second phase.[33] Note that an application for renegotiation can be to switch identities, thus we only need to to make sure that the parties 'behind' the identities remain the same.

To capture such attacks formally, we introduced the notion of weakly secure renegotiable ACCE protocols. The term 'weakly' only indicates that this notion is followed by an even stronger notion. For protocols to be secure under this notion we now additionally require a cryptographic link between phases, i.e. a party 'Alice' should only accept in a phase n (with intended communication partner 'Bob'), if this party 'Bob' accepts in a corresponding phase **and** it is the n-th phase for 'Bob' as well, while forbidding the adversary to learn session keys of previous phases. For most applications this security notion should be strong enough, as it (similar to the ACCE notion) implicitly also protects application data sent after a handshake protocol by encrypting the messages.

To completely cover all cases, we now additionally consider very specific adversaries that are somehow able to modify application data in a meaningful way without breaking ACCE security. So the (non-weakly) secure renegotiable ACCE security notion also requires that when parties accept, their communication partners not only accept in a corresponding phase but also have 'experienced' (that is, sent and received) the same set of messages in the previous phase. While this notion may seem too strong or too hard to achieve, for TLS it is easy to realize, as we show with our proposed countermeasure in Section 6.8.

7.4 Extension for Public Key Certification

In practice, the security of public key protocols like TLS heavily relies on trust in the underlying PKI. When analyzing protocols that use certificates to authenticate parties (e.g. by giving some proof of knowledge or possession that corresponds to some public value contained in the certificate, e.g. by signing certain values in a TLS handshake), in cryptography we usually assume that certificates for honest parties are always trusted. That is, either an adversary cannot create certificates at all or at least he cannot create certificates for these trusted parties.

On the internet we are dealing with Certification Authoritys (CAs), which provide institutions and individuals with 'certified' public keys, i.e. with certificates that have been signed by a CA. And as long as the signature scheme used to sign certificates is secure (and an adversary does not obtain a private key used by a CA to sign certificates), we can rely on these certificates and trust the parties in possession of the private keys corresponding the public keys contained in the certificates.

However, although in theory this problem may be considered solved, in practice we often face new and diverse attacks against PKIs. When the security of protocols depends on the security of the PKI, modeling these additional attack capabilities becomes interesting. Using the example of the TLS protocol, attacks against CAs can almost immediately be transformed into attacks against TLS (see for example attacks against Diginotar [Adv] and Comodo [Blo]).

[33]In this case however it was easier to deploy countermeasures for TLS renegotiation instead of patching all server applications (or HTTP).

Modeling Public Key Certification One could easily extend any security model described in Chapter 3 to capture secure public key registration and certification, by letting the execution environment act as CA, which computed all certificates as a signature over the party's public key pk_i, a *unique* identifier (for example a random string or a serial number) id_i, and some additional information aux_i (that can for example be used to identify the party P_i, like a domain name or an email address). The certificates are computed with the private key sk of a certification key pair (pk, sk).[34] All oracles would have access to the public certification key pk. We note, that security proofs might need to rely on different assumptions to enable simulatability.

Following this approach, one could give the adversary the power to register (up to n) arbitrary public keys and make the following change to the partner id Π:

- $\Pi \in [\phi + n]$ holds an index j that now points to a globally unique **certificate identity**.

Formally, registering a new public key would be performed by asking the following query:

- Register$(pk', aux, proof)$: Upon receiving this query, the execution environment first checks whether the adversary 'knows' the private key sk' corresponding to the public key pk' by evaluating the non-interactive proof $proof$ of possession of sk'. On failure it outputs an error symbol $\perp$, on success it outputs a certificate, binding the public key pk' and aux to a new globally unique identity id' generated by the execution environment. We need not make the environment 'generate' *all* information (including those used in the certificates requested by the adversary). This strengthens the model slightly as the adversary can freely specify aux as part of the certificate. This query could be restricted to be called at most n times.

Remark 23. We would only require that the proof is non-interactive to simplify the model (if a common reference string is required we may assume that it is held by the execution environment and made publicly available). In practice, the concrete implementation of these proofs of possession is up to the CA [AFKM05] and may also be interactive. We only require that it is secure under concurrent executions. Examples can be found in RFC 4210 [AFKM05] and PKCS#10 [NK00].

In general this proof only needs to be sound. However when relying on the knowledge of secret key (KOSK) assumption, we would require that the proof consists of sk'. One can think of $id' = id_z$ as being associated with an index $z \in \mathbb{N}$ that is initialized to $\phi + 1$. For each call of Register, z is incremented by one.

[34]Also, one could easily further extend the model to cover multiple certification authorities.

Bibliography

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway, *The oracle Diffie-Hellman assumptions and an analysis of DHIES*, Topics in Cryptology – CT-RSA 2001 (San Francisco, CA, USA) (David Naccache, ed.), Lecture Notes in Computer Science, vol. 2020, Springer, Berlin, Germany, April 8–12, 2001, pp. 143–158.

[Ada] Adam Langley, Google Security Team, *Protecting data for the long term with forward secrecy*, `http://googleonlinesecurity.blogspot.co.uk/2011/11/protecting-data-for-long-term-with.html`.

[Adv] Microsoft Security Advisory, *Fraudulent Digital Certificates Could Allow Spoofing, http://technet.microsoft.com/en-us/security/advisory/2607712.*

[AFKM05] C. Adams, S. Farrell, T. Kause, and T. Mononen, *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*, RFC 4210 (Proposed Standard), September 2005, Updated by RFC 6712.

[Bar04] Gregory V. Bard, *The Vulnerability of SSL to Chosen Plaintext Attack*, Cryptology ePrint Archive, Report 2004/111, 2004, `http://eprint.iacr.org/`.

[Bar06] ———, *A Challenging but Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL*, SECRYPT (Manu Malek, Eduardo Fernández-Medina, and Javier Hernando, eds.), INSTICC Press, 2006, pp. 99–109.

[BBM09] Timo Brecher, Emmanuel Bresson, and Mark Manulis, *Fully robust tree-Diffie-Hellman group key exchange*, CANS 09: 8th International Conference on Cryptology and Network Security(Kanazawa, Japan) (Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, eds.), Lecture Notes in Computer Science, vol. 5888, Springer, Berlin, Germany, December 12–14, 2009, pp. 478–497.

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio, *An uninstantiable random-oracle-model scheme for a hybrid-encryption problem*, Advances in Cryptology – EUROCRYPT 2004 (Interlaken, Switzerland) (Christian Cachin and Jan Camenisch, eds.), Lecture Notes in Computer Science, vol. 3027, Springer, Berlin, Germany, May 2–6, 2004, pp. 171–188.

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk, *A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract)*, 30th Annual ACM Symposium on Theory of Computing(Dallas, Texas, USA), ACM Press, May 23–26, 1998, pp. 419–428.

[BFCZ08] Karthikeyan Bhargavan, Cédric Fournet, Ricardo Corin, and Eugen Zalinescu, *Cryptographically verified implementations for TLS*, ACM CCS 08: 15th Conference on Computer and Communications Security(Alexandria, Virginia, USA)

(Peng Ning, Paul F. Syverson, and Somesh Jha, eds.), ACM Press, October 27–31, 2008, pp. 459–468.

[BFS$^+$12] Christina Brzuska, Mark Fischlin, Nigel P. Smart, Bogdan Warinschi, and Stephen C. Williams, *Less is More: Relaxed yet Composable Security Notions for Key Exchange*, 2012, Cryptology ePrint Archive, Report 2012/242.

[BFWW11] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams, *Composability of Bellare-Rogaway key exchange protocols*, ACM CCS 11: 18th Conference on Computer and Communications Security(Chicago, Illinois, USA) (Yan Chen, George Danezis, and Vitaly Shmatikov, eds.), ACM Press, October 17–21, 2011, pp. 51–62.

[Ble98] Daniel Bleichenbacher, *Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1*, Advances in Cryptology – CRYPTO'98 (Santa Barbara, CA, USA) (Hugo Krawczyk, ed.), Lecture Notes in Computer Science, vol. 1462, Springer, Berlin, Germany, August 23–27, 1998, pp. 1–12.

[Blo] Comodo Blogs, *The Recent RA Compromise,* `http://blogs.comodo.com/it-security/data-security/the-recent-ra-compromise/`.

[BLR04] Boaz Barak, Yehuda Lindell, and Tal Rabin, *Protocol Initialization for the Framework of Universal Composability*, Cryptology ePrint Archive, Report 2004/006, 2004, `http://eprint.iacr.org/`.

[BM08a] Emmanuel Bresson and Mark Manulis, *Contributory group key exchange in the presence of malicious participants*, IET Information Security **2** (2008), no. 3, 85–93.

[BM08b] ——, *Securing group key exchange against strong corruptions and key registration attacks*, IJACT **1** (2008), no. 2, 91–107.

[BN00] Mihir Bellare and Chanathip Namprempre, *Authenticated encryption: Relations among notions and analysis of the generic composition paradigm*, Advances in Cryptology – ASIACRYPT 2000 (Kyoto, Japan) (Tatsuaki Okamoto, ed.), Lecture Notes in Computer Science, vol. 1976, Springer, Berlin, Germany, December 3–7, 2000, pp. 531–545.

[BN08] ——, *Authenticated encryption: Relations among notions and analysis of the generic composition paradigm*, Journal of Cryptology **21** (2008), no. 4, 469–491.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway, *Authenticated key exchange secure against dictionary attacks*, Advances in Cryptology – EUROCRYPT 2000 (Bruges, Belgium) (Bart Preneel, ed.), Lecture Notes in Computer Science, vol. 1807, Springer, Berlin, Germany, May 14–18, 2000, pp. 139–155.

[BR93] Mihir Bellare and Phillip Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, ACM CCS 93: 1st Conference on Computer and Communications Security(Fairfax, Virginia, USA) (V. Ashby, ed.), ACM Press, November 3–5, 1993, pp. 62–73.

[BR94a] ———, *Entity authentication and key distribution*, Advances in Cryptology – CRYPTO'93 (Santa Barbara, CA, USA) (Douglas R. Stinson, ed.), Lecture Notes in Computer Science, vol. 773, Springer, Berlin, Germany, August 22–26, 1994, pp. 232–249.

[BR94b] ———, *Optimal asymmetric encryption*, Advances in Cryptology – EUROCRYPT'94 (Perugia, Italy) (Alfredo De Santis, ed.), Lecture Notes in Computer Science, vol. 950, Springer, Berlin, Germany, May 9–12, 1994, pp. 92–111.

[BR95] ———, *Provably secure session key distribution: The three party case*, 27th Annual ACM Symposium on Theory of Computing(Las Vegas, Nevada, USA), ACM Press, May 29 – June 1, 1995, pp. 57–66.

[BR06] ———, *The security of triple encryption and a framework for code-based game-playing proofs*, Advances in Cryptology – EUROCRYPT 2006 (St. Petersburg, Russia) (Serge Vaudenay, ed.), Lecture Notes in Computer Science, vol. 4004, Springer, Berlin, Germany, May 28 – June 1, 2006, pp. 409–426.

[Bro06] Daniel R. L. Brown, *What Hashes Make RSA-OAEP Secure?*, Cryptology ePrint Archive, Report 2006/223, 2006, `http://eprint.iacr.org/`.

[BSWW13] Christina Brzuska, Nigel P. Smart, Bogdan Warinschi, and Gaven J. Watson, *EMV Key Agreement*, Cryptology ePrint Archive, Report 2013/031, 2013, `http://eprint.iacr.org/`.

[BWBG$^+$06] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*, RFC 4492 (Informational), May 2006, Updated by RFC 5246.

[BWJM97] Simon Blake-Wilson, Don Johnson, and Alfred Menezes, *Key agreement protocols and their security analysis*, 6th IMA International Conference on Cryptography and Coding (Cirencester, UK) (Michael Darnell, ed.), Lecture Notes in Computer Science, vol. 1355, Springer, Berlin, Germany, December 17–19, 1997, pp. 30–45.

[BWM99] Simon Blake-Wilson and Alfred Menezes, *Unknown key-share attacks on the station-to-station (STS) protocol*, PKC'99: 2nd International Workshop on Theory and Practice in Public Key Cryptography(Kamakura, Japan) (Hideki Imai and Yuliang Zheng, eds.), Lecture Notes in Computer Science, vol. 1560, Springer, Berlin, Germany, March 1–3, 1999, pp. 154–170.

[BWNH$^+$03] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, *Transport Layer Security (TLS) Extensions*, RFC 3546 (Proposed Standard), June 2003, Obsoleted by RFC 4366.

[Can00] Ran Canetti, *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, Cryptology ePrint Archive, Report 2000/067, 2000, `http://eprint.iacr.org/`.

[Can01] Ran Canetti, *Universally composable security: A new paradigm for cryptographic protocols*, 42nd Annual Symposium on Foundations of Computer Science(Las Vegas, Nevada, USA), IEEE Computer Society Press, October 14–17, 2001, pp. 136–145.

[CBH05a] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock, *Errors in computational complexity proofs for protocols*, Advances in Cryptology – ASIACRYPT 2005 (Chennai, India) (Bimal K. Roy, ed.), Lecture Notes in Computer Science, vol. 3788, Springer, Berlin, Germany, December 4–8, 2005, pp. 624–643.

[CBH05b] ———, *Examining indistinguishability-based proof models for key establishment protocols*, Advances in Cryptology – ASIACRYPT 2005 (Chennai, India) (Bimal K. Roy, ed.), Lecture Notes in Computer Science, vol. 3788, Springer, Berlin, Germany, December 4–8, 2005, pp. 585–604.

[CBH05c] Kim-Kwang Raymond Choo, Colin Boyd, and Yvonne Hitchcock, *On Session Key Construction in Provably-Secure Key Establishment Protocols*, Mycrypt (Ed Dawson and Serge Vaudenay, eds.), Lecture Notes in Computer Science, vol. 3715, Springer, 2005, pp. 116–131.

[CD09] S. Chaki and A. Datta, *ASPIER: An Automated Framework for Verifying Security Protocol Implementations*, Computer Security Foundations Symposium, 2009. CSF '09. 22nd IEEE, july 2009, pp. 172–185.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi, *The random oracle methodology, revisited (preliminary version)*, 30th Annual ACM Symposium on Theory of Computing(Dallas, Texas, USA), ACM Press, May 23–26, 1998, pp. 209–218.

[CJNP00] Jean-Sébastien Coron, Marc Joye, David Naccache, and Pascal Paillier, *New attacks on PKCS#1 v1.5 encryption*, Advances in Cryptology – EUROCRYPT 2000 (Bruges, Belgium) (Bart Preneel, ed.), Lecture Notes in Computer Science, vol. 1807, Springer, Berlin, Germany, May 14–18, 2000, pp. 369–381.

[CK01] Ran Canetti and Hugo Krawczyk, *Analysis of key-exchange protocols and their use for building secure channels*, Advances in Cryptology – EUROCRYPT 2001 (Innsbruck, Austria) (Birgit Pfitzmann, ed.), Lecture Notes in Computer Science, vol. 2045, Springer, Berlin, Germany, May 6–10, 2001, pp. 453–474.

[CK02] ———, *Security analysis of IKE's signature-based key-exchange protocol*, Advances in Cryptology – CRYPTO 2002 (Santa Barbara, CA, USA) (Moti Yung, ed.), Lecture Notes in Computer Science, vol. 2442, Springer, Berlin, Germany, August 18–22, 2002, `http://eprint.iacr.org/2002/120/`, pp. 143–161.

[Cre09] Cas J. F. Cremers, *Session-state reveal is stronger than ephemeral key reveal: Attacking the NAXOS authenticated key exchange protocol*, ACNS 09: 7th International Conference on Applied Cryptography and Network Security(Paris-Rocquencourt, France) (Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, eds.), Lecture Notes in Computer Science, vol. 5536, Springer, Berlin, Germany, June 2–5, 2009, pp. 20–33.

[DA99] T. Dierks and C. Allen, *The TLS Protocol Version 1.0*, RFC 2246 (Proposed Standard), January 1999, Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176.

[Den02] Alexander W. Dent, *Adapting the weaknesses of the random oracle model to the generic group model*, Advances in Cryptology – ASIACRYPT 2002 (Queenstown, New Zealand) (Yuliang Zheng, ed.), Lecture Notes in Computer Science, vol. 2501, Springer, Berlin, Germany, December 1–5, 2002, pp. 100–109.

[DH76] Whitfield Diffie and Martin E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), no. 6, 644–654.

[DR06] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.1*, RFC 4346 (Proposed Standard), April 2006, Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176.

[DR08] ______, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246 (Proposed Standard), August 2008, Updated by RFCs 5746, 5878, 6176.

[DY83] Danny Dolev and Andrew Chi-Chih Yao, *On the security of public key protocols*, IEEE Transactions on Information Theory **29** (1983), no. 2, 198–207.

[Far10] Stephen Farrell, *Why didn't we spot that?*, IEEE Internet Computing **14** (2010), no. 1, 84–87.

[FKK11] A. Freier, P. Karlton, and P. Kocher, *The Secure Sockets Layer (SSL) Protocol Version 3.0*, RFC 6101 (Historic), August 2011.

[FOPS01] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern, *RSA-OAEP is secure under the RSA assumption*, Advances in Cryptology – CRYPTO 2001 (Santa Barbara, CA, USA) (Joe Kilian, ed.), Lecture Notes in Computer Science, vol. 2139, Springer, Berlin, Germany, August 19–23, 2001, pp. 260–274.

[FPZ08] Pierre-Alain Fouque, David Pointcheval, and Sébastien Zimmer, *HMAC is a randomness extractor and applications to TLS*, ASIACCS 08: 3rd Conference on Computer and Communications Security(Tokyo, Japan) (Masayuki Abe and Virgil Gligor, eds.), ACM Press, March 18–20, 2008, pp. 21–32.

[FSXY12] Atsushi Fujioka, Koutarou Suzuki, Keita Xagawa, and Kazuki Yoneyama, *Strongly secure authenticated key exchange from factoring, codes, and lattices*, PKC 2012: 15th International Workshop on Theory and Practice in Public Key Cryptography(Darmstadt, Germany) (Marc Fischlin, Johannes Buchmann, and Mark Manulis, eds.), Lecture Notes in Computer Science, vol. 7293, Springer, Berlin, Germany, May 21–23, 2012, pp. 467–484.

[Gel12] Rati Gelashvili, *Attacks on re-keying and renegotiation in Key Exchange Protocols*, April 2012, Bachelor's thesis, ETH Zurich.

[GKS12] Florian Giesen, Florian Kohlar, and Douglas Stebila, *On the Security of TLS Renegotiation*, IACR Cryptology ePrint Archive **2012** (2012), 630.

[GKS13] ———, *On the Security of TLS Renegotiation*, 20th ACM Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 2013.

[GMP$^+$08a] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk, *Universally composable security analysis of TLS*, ProvSec 2008: 2nd International Conference on Provable Security(Shanghai, China) (Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, eds.), Lecture Notes in Computer Science, vol. 5324, Springer, Berlin, Germany, October 31 – November 1, 2008, pp. 313–327.

[GMP$^+$08b] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk, *Universally Composable Security Analysis of TLS*, ProvSec (Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, eds.), LNCS, vol. 5324, Springer, 2008, pp. 313–327.

[GSF$^+$04] Vipul Gupta, Douglas Stebila, Stephen Fung, Sheueling Chang Shantz, Nils Gura, and Hans Eberle, *Speeding up secure web transactions using elliptic curve cryptography*, ISOC Network and Distributed System Security Symposium – NDSS 2004 (San Diego, California, USA), The Internet Society, February 4–6, 2004.

[HJK11] Dennis Hofheinz, Tibor Jager, and Eike Kiltz, *Short signatures from weaker assumptions*, Advances in Cryptology – ASIACRYPT 2011 (Seoul, South Korea) (Dong Hoon Lee and Xiaoyun Wang, eds.), Lecture Notes in Computer Science, vol. 7073, Springer, Berlin, Germany, December 4–8, 2011, pp. 647–666.

[HK09a] Dennis Hofheinz and Eike Kiltz, *The group of signed quadratic residues and applications*, Advances in Cryptology – CRYPTO 2009 (Santa Barbara, CA, USA) (Shai Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, Berlin, Germany, August 16–20, 2009, pp. 637–653.

[HK09b] ———, *Practical chosen ciphertext secure encryption from factoring*, Advances in Cryptology – EUROCRYPT 2009 (Cologne, Germany) (Antoine Joux, ed.), Lecture Notes in Computer Science, vol. 5479, Springer, Berlin, Germany, April 26–30, 2009, pp. 313–332.

[HS11] Dennis Hofheinz and Victor Shoup, *GNUC: A New Universal Composability Framework*, Cryptology ePrint Archive, Report 2011/303, 2011, `http://eprint.iacr.org/`.

[HW09] Susan Hohenberger and Brent Waters, *Short and stateless signatures from the RSA assumption*, Advances in Cryptology – CRYPTO 2009 (Santa Barbara, CA, USA) (Shai Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, Berlin, Germany, August 16–20, 2009, pp. 654–670.

[JK02] Jakob Jonsson and Burton S. Kaliski Jr., *On the security of RSA encryption in TLS*, Advances in Cryptology – CRYPTO 2002 (Santa Barbara, CA, USA) (Moti Yung, ed.), Lecture Notes in Computer Science, vol. 2442, Springer, Berlin, Germany, August 18–22, 2002, pp. 127–142.

[JKSS10] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk, *Generic Compilers for Authenticated Key Exchange*, Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings (Masayuki Abe, ed.), Lecture Notes in Computer Science, vol. 6477, Springer, 2010, pp. 232–249.

[JKSS12] ———, *On the Security of TLS-DHE in the Standard Model*, Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings (Reihaneh Safavi-Naini and Ran Canetti, eds.), Lecture Notes in Computer Science, vol. 7417, Springer, 2012, pp. 273–293.

[JSS12] Tibor Jager, Sebastian Schinzel, and Juraj Somorovsky, *Bleichenbacher's attack strikes again: Breaking PKCS#1 v1.5 in XML encryption*, ESORICS 2012: 17th European Symposium on Research in Computer Security(Pisa, Italy) (Sara Foresti, Moti Yung, and Fabio Martinelli, eds.), Lecture Notes in Computer Science, vol. 7459, Springer, Berlin, Germany, September 10–12, 2012, pp. 752–769.

[Kal98] B. Kaliski, *PKCS #1: RSA Encryption Version 1.5*, RFC 2313 (Informational), March 1998, Obsoleted by RFC 2437.

[KOS10] Eike Kiltz, Adam O'Neill, and Adam Smith, *Instantiability of RSA-OAEP under chosen-plaintext attack*, Advances in Cryptology – CRYPTO 2010 (Santa Barbara, CA, USA) (Tal Rabin, ed.), Lecture Notes in Computer Science, vol. 6223, Springer, Berlin, Germany, August 15–19, 2010, pp. 295–313.

[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*, Advances in Cryptology – EUROCRYPT 2009 (Cologne, Germany) (Antoine Joux, ed.), Lecture Notes in Computer Science, vol. 5479, Springer, Berlin, Germany, April 26–30, 2009, pp. 389–406.

[Kra01] Hugo Krawczyk, *The order of encryption and authentication for protecting communications (or: How secure is SSL?)*, Advances in Cryptology – CRYPTO 2001 (Santa Barbara, CA, USA) (Joe Kilian, ed.), Lecture Notes in Computer Science, vol. 2139, Springer, Berlin, Germany, August 19–23, 2001, pp. 310–331.

[Kra05] ———, *HMQV: A high-performance secure diffie-hellman protocol*, Advances in Cryptology – CRYPTO 2005 (Santa Barbara, CA, USA) (Victor Shoup, ed.), Lecture Notes in Computer Science, vol. 3621, Springer, Berlin, Germany, August 14–18, 2005, pp. 546–566.

[KT11] Ralf Küsters and Max Tuengerthal, *Composition theorems without pre-established session identifiers*, ACM CCS 11: 18th Conference on Computer and Communications Security(Chicago, Illinois, USA) (Yan Chen, George Danezis, and Vitaly Shmatikov, eds.), ACM Press, October 17–21, 2011, pp. 41–50.

[KY03] Jonathan Katz and Moti Yung, *Scalable protocols for authenticated group key exchange*, Advances in Cryptology – CRYPTO 2003 (Santa Barbara, CA, USA)

(Dan Boneh, ed.), Lecture Notes in Computer Science, vol. 2729, Springer, Berlin, Germany, August 17–21, 2003, pp. 110–125.

[Lan11] Adam Langley, *Google Online Security Blog: Protecting data for the long term with forward secrecy*, November 2011, `http://googleonlinesecurity.blogspot.com/2011/11/protecting-data-for-long-term-with.html`.

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin, *Stronger security of authenticated key exchange*, ProvSec 2007: 1st International Conference on Provable Security(Wollongong, Australia) (Willy Susilo, Joseph K. Liu, and Yi Mu, eds.), Lecture Notes in Computer Science, vol. 4784, Springer, Berlin, Germany, November 1–2, 2007, pp. 1–16.

[Man09] Mark Manulis, *Group key exchange enabling on-demand derivation of peer-to-peer keys*, ACNS 09: 7th International Conference on Applied Cryptography and Network Security(Paris-Rocquencourt, France) (Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, eds.), Lecture Notes in Computer Science, vol. 5536, Springer, Berlin, Germany, June 2–5, 2009, pp. 1–19.

[Mit98] John C. Mitchell, *Finite-State Analysis of Security Protocols*, CAV (Alan J. Hu and Moshe Y. Vardi, eds.), LNCS, vol. 1427, Springer, 1998, pp. 71–76.

[MPT10] Mark Manulis, Bertram Poettering, and Gene Tsudik, *Affiliation-hiding key exchange with untrusted group authorities*, ACNS 10: 8th International Conference on Applied Cryptography and Network Security(Beijing, China) (Jianying Zhou and Moti Yung, eds.), Lecture Notes in Computer Science, vol. 6123, Springer, Berlin, Germany, June 22–25, 2010, pp. 402–419.

[MSW08] Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi, *A modular security analysis of the TLS handshake protocol*, Advances in Cryptology – ASIACRYPT 2008 (Melbourne, Australia) (Josef Pieprzyk, ed.), Lecture Notes in Computer Science, vol. 5350, Springer, Berlin, Germany, December 7–11, 2008, pp. 55–73.

[MSW10] ———, *The TLS handshake protocol: A modular analysis*, Journal of Cryptology **23** (2010), no. 2, 187–223.

[MT10] Ueli Maurer and Björn Tackmann, *On the soundness of authenticate-then-encrypt: formalizing the malleability of symmetric encryption*, ACM CCS 10: 17th Conference on Computer and Communications Security(Chicago, Illinois, USA) (Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, eds.), ACM Press, October 4–8, 2010, pp. 505–515.

[NK00] M. Nystrom and B. Kaliski, *PKCS #10: Certification Request Syntax Specification Version 1.7*, RFC 2986 (Informational), November 2000, Updated by RFC 5967.

[OF05] Kazuhiro Ogata and Kokichi Futatsugi, *Equational Approach to Formal Analysis of TLS*, ICDCS, IEEE Computer Society, 2005, pp. 795–804.

[Oka07] Tatsuaki Okamoto, *Authenticated key exchange and key encapsulation in the standard model (invited talk)*, Advances in Cryptology – ASIACRYPT 2007 (Kuching, Malaysia) (Kaoru Kurosawa, ed.), Lecture Notes in Computer Science, vol. 4833, Springer, Berlin, Germany, December 2–6, 2007, pp. 474–484.

[Pau99] Lawrence C. Paulson, *Inductive Analysis of the Internet Protocol TLS*, ACM Trans. Inf. Syst. Secur. **2** (1999), no. 3, 332–351.

[PRS11] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton, *Tag size does matter: Attacks and proofs for the TLS record protocol*, Advances in Cryptology – ASIACRYPT 2011 (Seoul, South Korea) (Dong Hoon Lee and Xiaoyun Wang, eds.), Lecture Notes in Computer Science, vol. 7073, Springer, Berlin, Germany, December 4–8, 2011, pp. 372–389.

[RD09] Marsh Ray and Steve Dispensa, *Renegotiating TLS*, November 2009, `http://extendedsubset.com/Renegotiating_TLS.pdf`.

[RD11] Juliano Rizzo and Thai Duong, *BEAST: Surprising crypto attack against HTTPS*, 2011, `http://www.ekoparty.org/2011/juliano-rizzo.php`.

[RD12] ———, *The CRIME Attack*, ekoparty Security Conference 8° edition, 2012, `http://www.ekoparty.org/2012/thai-duong.php`.

[Res00] E. Rescorla, *HTTP Over TLS*, RFC 2818 (Informational), May 2000, Updated by RFC 5785.

[RRDO10] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov, *Transport Layer Security (TLS) Renegotiation Indication Extension*, RFC 5746 (Proposed Standard), February 2010.

[SEVB10] Augustin P. Sarr, Philippe Elbaz-Vincent, and Jean-Claude Bajard, *A new security model for authenticated key agreement*, SCN 10: 7th International Conference on Security in Communication Networks(Amalfi, Italy) (Juan A. Garay and Roberto De Prisco, eds.), Lecture Notes in Computer Science, vol. 6280, Springer, Berlin, Germany, September 13–15, 2010, pp. 219–234.

[Sho04] Victor Shoup, *Sequences of games: a tool for taming complexity in security proofs*, Cryptology ePrint Archive, Report 2004/332, Nov 2004.

[Vau02] Serge Vaudenay, *Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ...*, Advances in Cryptology – EUROCRYPT 2002 (Amsterdam, The Netherlands) (Lars R. Knudsen, ed.), Lecture Notes in Computer Science, vol. 2332, Springer, Berlin, Germany, April 28 – May 2, 2002, pp. 534–546.

[WS96] David Wagner and Bruce Schneier, *Analysis of the SSL 3.0 protocol*, Proceedings of the Second USENIX Workshop on Electronic Commerce, USENIX Association, 1996, pp. 29–40.

[Zol09] Thierry Zoller, *TLS & SSLv3 renegotiation vulnerability*, Tech. report, G-SEC, 2009, `http://www.g-sec.lu/practicaltls.pdf`.

Glossary

ms master secret. 72, 73, 79, 95, 96, 100, 101, 107, 111, 112, 115, 118, 134

pms premaster secret. 71, 73, 86, 90, 94, 95, 99, 100, 103, 107, 110, 111, 114, 140

ACCE Authenticated and Confidential Channel Establishment. 2, 3, 23, 28, 33–45, 48–53, 65, 69, 89, 91, 97, 98, 104, 107, 109, 114, 121, 122, 126–128, 130, 132, 136, 139, 141, 142, 146–148, 150, 151, 155, 156

AKE Authenticated Key Exchange. 2, 4, 23, 26–33, 36–39, 41, 43, 44, 57, 60, 65, 66, 69, 77, 81, 85, 123, 153, 154

CA Certification Authority. 156, 157

CBC Cipher Block Chaining. 65, 68

CCA Chosen-Ciphertext Attacks. 16, 68, 106, 154

CDH Computational Diffie-Hellman. 69

CPA Chosen-Plaintext Attacks. 15, 154, 155

DDH Decisional Diffie-Hellman. 13, 69, 75, 76, 83, 86, 87, 107, 108, 110, 122, 130, 137, 140

DH Diffie-Hellman. 6, 20, 21, 25, 38, 44, 58, 66–69, 71–76, 78, 79, 82, 83, 86, 88, 92, 95, 97, 107, 108, 110, 118, 154

DHE Ephemeral Diffie-Hellman. 154

ECC Elliptic Curve Cryptography. 155

ECDHE Ephemeral Elliptic Curve Diffie-Hellman. 155

EUF-CMA Existential Unforgeability under Chosen-Message Attacks. 14

HTTP Hypertext Transfer Protocol. 124, 156

HTTPS Hypertext Transfer Protocol Secure. 124

IND-CCA Indistinguishable under Chosen-Ciphertext Attacks. 66

KCI Key-Compromise Impersonation Attacks. 38, 39

KE Key Exchange. 18, 153

MAC Message Authentication Code. 4, 17, 34, 44, 58, 59, 63, 66, 68, 73, 75, 128, 129, 144, 150, 151, 153

MEE Mac-then-Encode-then-Encrypt. 68

MITM Man-In-The-Middle. 4, 59, 66, 124, 153

OW-CPA One-Wayness under Chosen-Plaintext Attacks. 68

PKCS Public Key Cryptography Standard. 8, 66, 68, 74, 154, 157

PKE Public-Key Encryption. 15, 99, 103

PKI Public-Key Infrastructure. 71, 156

PPT Probabilistic Polynomial-Time. 8, 24

PRF Pseudo-Random Function. 17, 20, 59, 62, 65, 69, 73, 93, 94, 96, 97, 100–103, 105–108, 111–113, 115, 147, 151, 153, 155

PRF-ODH PRF-Oracle Diffie Hellman. 2, 20, 21, 69, 76, 79, 87, 95, 96, 108, 117, 118, 122, 130, 134

RFC Request For Comments. 121, 124, 157

RIE Renegotiation Information Extension. 42, 51, 53, 121, 122, 126, 127, 129, 130, 141, 146, 148, 150

ROM Random Oracle Model. 9, 68, 155

RSA Rivest-Shamir-Adleman (Cryptosystem). 9, 66, 68, 126, 146, 150, 154, 155

RSA-OAEP RSA with Optimal Asymmetric Encryption Padding. 9, 67, 154

SCSV Signalling Ciphersuite Value. 42, 51, 53, 121, 122, 126, 127, 129, 130, 141, 146, 148, 150

sLHAE Stateful Length-Hiding Authenticated Encryption. 21, 35, 39, 44, 91–97, 101–103, 105–108, 112, 113, 115–119, 130, 136, 138, 141, 151

SMTPS SMTP over TLS. 124

SSL Secure Socket Layer. 4, 66, 67, 74, 123, 125, 155

TLS Transport Layer Security. 1, 2, 4, 5, 9–11, 17, 27, 28, 32–35, 38–42, 44–47, 49–51, 53, 57, 65–70, 73–76, 87, 88, 91, 94, 95, 108, 117, 121–130, 135, 138, 140, 141, 146–148, 150, 151, 153–156

TLS-DHE TLS Ciphersuites with Ephemeral Diffie-Hellman-based Key Exchange. 2, 65, 69, 71, 72, 75, 87, 92, 95, 117, 118, 122, 154

TLS-RSA TLS Ciphersuites with RSA-based Key Transport. 2, 65, 68, 71–73, 92, 96, 97, 106, 154

TLS-SDH TLS Ciphersuites with Static Diffie-Hellman-based Key Exchange. 2, 65, 71–73, 92, 107, 117, 154

UC Universal Composability. 7, 8, 67, 127

UKS Unknown Key Share Attacks. 9, 59, 68, 153

Florian Kohlar

Curriculum Vitae

Personal Information

born	**March 9th, 1982**. in Bochum, Germany
Parents	**Carmen Kohlar**, *Head of a day nursery in Bochum*.
	Heinrich Kohlar, *Chief inspector at the correctional facility in Bochum*.
Brother	**Tobias Kohlar**, *Storeman at IKEA*.
Partner	**Annette Klocke**, *Notary assistant*.
Child	**Levi Alexander Kohlar**. born 2012

Education

School

1988–1992	**Basic school**, *Köllerholzschule*, Bochum.
1992–2001	**Secondary School**, *Theodor-Körner-Schule*, Gymnasium, Bochum. graduated with Abitur

University

2001–2008	**Diploma degree course**, *IT-Security*, Ruhr-University Bochum. graduated with Diploma
2009–2013	**PhD studies**, *IT-Security*, Ruhr-University Bochum. graduated with Diploma

Working Experience

2003–2008	**Student assistant**, *Ruhr-University Bochum*. Programming the student management system
2009-2013	**Research associate**, *Chair for Network- and Datasecurity*. Ruhr-University Bochum

Spoken Languages

German	**Native speaker**
English	**Fluent in speech and writing**
French	**Basic Skills**

Am Krüzweg 49 – D-44879 Bochum
0160-8073245 • ☎ 0234-32-26798 • FAX 0234-32-14347
✉ florian.kohlar@ruhr-uni-bochum.de

Miscellaneous

- Active member of the Federal Agency for Technical Relief in Bochum

Diploma Thesis

Title	*Sicherheit von XML-basierten Nachrichten und Dateien*
Supervisor	Prof. Dr. Jörg Schwenk

PhD Thesis

Title	*On the Cryptographic Security of Browser-Based Protocols*
Supervisor	Prof. Dr. Jörg Schwenk

Publications

Florian Giesen, Florian Kohlar, and Douglas Stebila. On the Security of TLS Renegotiation. *IACR Cryptology ePrint Archive*, 2012:630, 2012.

Nils Gruschka, Meiko Jensen, Florian Kohlar, and Lijun Liao. On Interoperability Failures in WS-Security. In Ejub Kajan, editor, *Electronic Business Interoperability: Concepts, Opportunities and Challenges*, pages 615–635. IGI Global, 2011.

Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic Compilers for Authenticated Key Exchange. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 2010.

Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the Security of TLS-DHE in the Standard Model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, 2012.

Florian Kohlar, Jörg Schwenk, Meiko Jensen, and Sebastian Gajek. Secure Bindings of SAML Assertions to TLS Sessions. In *ARES 2010, Fifth International Conference on Availability, Reliability and Security, 15-18 February 2010, Krakow, Poland*, pages 62–69. IEEE Computer Society, 2010.

Florian Kohlar, Jörg Schwenk, Meiko Jensen, and Sebastian Gajek. On Cryptographically Strong Bindings of SAML Assertions to Transport Layer Security. *IJMCMC*, 3(4):20–35, 2011.

Jörg Schwenk, Florian Kohlar, and Marcus Amon. The Power of Recognition: Secure Single Sign-On using TLS Channel Bindings. In *Proceedings of the 7th ACM workshop on Digital identity management*, DIM '11, pages 63–72, New York, NY, USA, 2011. ACM.

Pavol Sovis, Florian Kohlar, and Jörg Schwenk. Security Analysis of OpenID. In Felix C. Freiling, editor, *Sicherheit 2010: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 5. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 5.-7. Oktober 2010 in Berlin*, volume 170 of *LNI*, pages 329–340. GI, 2010.

Project Experience

AROSI **2009**	Joint project with esrcypt GmbH and the German Federal Office for Information Security (BSI) with the goal the implement an architecture to realise object security in internal networks.
Gematik **2009**	Goal of this project was to write a security expertise of the Telematik infrastructure, used in context with the German eHealth card. My part was to help evaluating the decentralized components deployed in the environments of the health care providers, specifically to check if personal data was stored and protected according to data privacy laws.
Sec^2 **2010**	This project aimed at creating decentralized security solutions for storing collaborative data on the cloud.
SkIDentity **2012**	This project aimed at creating a bridge solution between secure electronic ID cards and currently existing cloud computing infrastructures. The goal was to provide trustworthy identities for the cloud to secure processes and values.

April 2013