On the Security of TLS Renegotiation*

Florian Giesen[1] Florian Kohlar[1] Douglas Stebila[2]

[1] *Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Bochum, Germany*
florian.giesen@rub.de, florian.kohlar@rub.de

[2] *Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia*
stebila@qut.edu.au

March 2, 2013

Abstract

The Transport Layer Security (TLS) protocol is the most widely used security protocol on the Internet. It supports negotiation of a wide variety of cryptographic primitives through different cipher suites, various modes of client authentication, and additional features such as renegotiation. Despite its widespread use, only recently has the full TLS protocol been proven secure, and only then a single ciphersuite family (`TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA`) with no additional features. These additional features have been the cause of practical attacks on TLS. In 2009, Ray and Dispensa demonstrated how TLS renegotiation allows an attacker to splice together its own session with that of a victim, resulting in a man-in-the-middle attack on TLS-reliant applications such as HTTP. TLS was subsequently patched with two defence mechanisms for protection against this attack.

We present the first formal treatment of renegotiation in secure channel establishment protocols. We add optional renegotiation to the authenticated and confidential channel establishment model of Jager et al., an adaptation of the Bellare–Rogaway authenticated key exchange model. We describe the attack of Ray and Dispensa on TLS within our model. We show generically that the proposed fixes for TLS offer good protection against renegotiation attacks, and give a simple new countermeasure that provides renegotiation security for TLS even in the face of stronger adversaries.

Keywords: Transport Layer Security (TLS), renegotiation, security models, key exchange

*The research leading to these results has received funding from the European Community (FP7/2007-2013) under grant agreement number ICT-2007-216646 - European Network of Excellence in Cryptology II (ECRYPT II), the Australian Technology Network–German Academic Exchange Service (ATN-DAAD) Joint Research Co-operation Scheme, and the Australian Research Council (ARC) Discovery Project scheme.

Contents

1 Introduction **3**
- 1.1 The TLS Renegotiation Issue . . . 3
- 1.2 Countermeasures Added to TLS . . . 4
- 1.3 Contributions . . . 5

2 Security Definitions for Multi-Phase and Renegotiable ACCE **6**
- 2.1 Overview . . . 7
- 2.2 Execution Environment . . . 7
- 2.3 Security Definitions . . . 10
  - 2.3.1 Confidentiality. . . . 10
  - 2.3.2 Secure multi-phase ACCE. . . . 11
  - 2.3.3 Secure renegotiable ACCE. . . . 11
  - 2.3.4 Weakly secure renegotiable ACCE. . . . 12

3 Renegotiation (In)security of TLS **12**
- 3.1 TLS without countermeasures is not a (weakly) secure renegotiable ACCE protocol . . . 12

4 Renegotiation Security of TLS with SCSV/RIE Countermeasures **13**
- 4.1 TLS_∗ with SCSV/RIE is not a secure renegotiable ACCE . . . 14
- 4.2 Tagged-ACCE security model and tagged TLS . . . 15
  - 4.2.1 Tagged-ACCE security model . . . 15
  - 4.2.2 Tagged-ACCE-fin security model . . . 15
  - 4.2.3 Tagged TLS . . . 15
  - 4.2.4 Proof of Lemma 1: $\epsilon_{\mathsf{client}}$ . . . 17
  - 4.2.5 Proof of Lemma 1: $\epsilon_{\mathsf{server}}$ . . . 20
  - 4.2.6 Proof of Confidentiality . . . 22
- 4.3 TLS with SCSV/RIE is a secure multi-phase ACCE . . . 24
- 4.4 TLS with SCSV/RIE is a weakly secure renegotiable ACCE . . . 28

5 Renegotiation Security of TLS with a New Countermeasure **30**

6 Conclusion **31**

References **32**

A Additional Definitions **33**
- A.1 Matching Conversations . . . 33
- A.2 Stateful Length-Hiding Authenticated Encryption (sLHAE) . . . 33
- A.3 The PRF-Oracle-Diffie-Hellman Assumption . . . 34

B Protocols without Forward Security **35**
- B.1 Model . . . 35
- B.2 On Renegotiation Security of TLS_RSA_ with SCSV/RIE . . . 35

C TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA Protocol with Renegotiation Extensions **36**

D Generic TLS Protocol with Renegotiation Extensions **37**

1 Introduction

The Transport Layer Security (TLS) protocol, the successor of the Secure Sockets Layer (SSL) protocol, provides secure channel establishment on the Internet. It is commonly used to protect information sent via the Hypertext Transfer Protocol (HTTP) on the web, and many other application layer protocols such as email and file transfer. TLS consists of a *handshake* protocol, which is used to agree on security parameters, establish a secret key, and authenticate the parties; and a *record layer* protocol, which is used to send encrypted data.

Despite the importance of TLS, progress on formally modelling the security of TLS has been slow. A technicality of TLS prevents it from being proven secure in standard authenticated key exchange (AKE) models: in AKE, the session key must be indistinguishable from a random key of the same length. However, the final handshake message of the TLS protocol is encrypted under the session key, so an adversary can distinguish the session key from a random key by trying to verify the final handshake message. Some analyses [18, 22] have shown that a truncated form of the TLS handshake is AKE-secure. Others [13] deal with a substantially weaker security requirement, namely unauthenticated key agreement. Krawczyk [19] analyzed a variant of the TLS record layer.

Only very recently have analyses of unmodified TLS functionality appeared. Paterson *et al.* at ASIACRYPT 2011 [23] showed that TLS's MAC-then-encode-then-encrypt record layer when used with CBC encryption (with certain length restrictions) satisfies a notion called *length-hiding authenticated encryption (LHAE)*. Subsequently, at CRYPTO 2012 Jager *et al.* [16] gave the first full proof of the security of (one ciphersuite of) unmodified TLS in a strong security model. Jager *et al.* introduced a variant of the Bellare–Rogaway authenticated key exchange model, called *authenticated and confidential channel establishment (ACCE)*. They proved that the TLS 1.2 protocol using the `TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA` ciphersuite (which we shorten to `TLS_DHE_DSS_`) is a secure ACCE protocol, under standard assumptions on the cryptographic components. (We note an alternative modular approach to proving the full security of TLS was recently given by Brzuska *et al.* [6].)

But TLS is not just a basic secure channel: it consists of hundreds of variants with many optional complex functionalities. Alert messages report various error conditions. Previous sessions can be resumed with a shortened handshake. As of February 2013, over 300 ciphersuites — combinations of cryptographic primitives — have been standardized. Client authentication is optional, and can be certificate-based or password-based. Various additional options can be specified via extensions and optional fields. Record layer communication can be compressed. And most importantly for this paper, after a TLS handshake has been completed and transmission on the record layer has started, parties can renegotiate the handshake. There have been many attacks on TLS over the years, such Bleichenbacher's attack [5] and others involving padding, and Ray and Dispensa's renegotiation attack [24], all of which exploit flaws in the non-core functionality of TLS.

In this paper, we focus on renegotiation, 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 her identity over a public channel, she could first authenticate anonymously (or with pseudonymous credentials), then renegotiate using her real certificate; since the renegotiation messages are transmitted within the existing record layer, the transmission of her certificate is encrypted, and thus she obtains privacy for her 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 [11] and the recent thesis of Gelashvili [14], 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.

1.1 The TLS Renegotiation Issue

All versions of TLS [8, 9, 10], and SSL v3 [12] 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.

In November 2009, Ray and Dispensa [24] described a man-in-the-middle attack that exploits how certain TLS-reliant applications — such as HTTP over TLS [25] — process data across renegotiations. The attack is shown in Figure 1. 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.

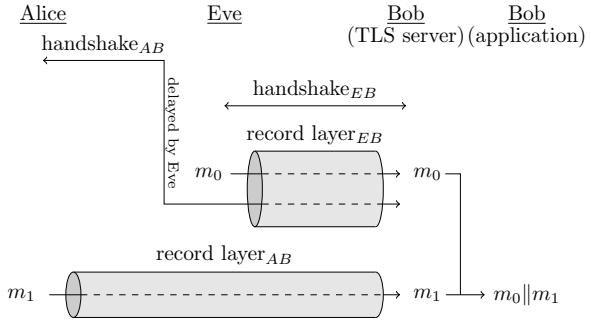

Figure 1: Ray and Dispensa's man-in-the-middle renegotiation attack on TLS-reliant applications

This is not strictly speaking an attack on TLS but on how some applications process TLS-protected data. It results from some applications, including HTTPS [24] and SMTPS [28], 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

$$m_0 \| m_1 = \text{“}\texttt{GET /orderPizza?deliverTo=123-Fake-St} \hookleftarrow$$
$$\texttt{X-Ignore-This: GET /orderPizza?deliverTo=456-Real-St} \hookleftarrow$$
$$\texttt{Cookie: Account=111A2B}\text{”}$$

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 [24, 28].

1.2 Countermeasures Added to TLS

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 [26] 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 from the previous handshake in a `ClientHello`/`ServerHello` extension [4], demonstrating they have the same view of the previous handshake, or a distinguished null value if not renegotiation. SCSV is a slight modification that is more compatible with buggy implementations. A diagram showing the message flow for a generic TLS ciphersuite with SCSV/RIE countermeasures appears in Figure 6 in Appendix D on page 37.

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 1 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.

1.3 Contributions

Security model for renegotiable channel establishment protocols. In Section 2, we present a new security model for renegotiable protocols. Since our goal is to analyze the security of TLS, we start from the ACCE model, rather than AKE security models. 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.[1] This is qualitatively different than 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. Our extension to the ACCE model also models server-only authentication in addition to mutual authentication.

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

Analysis of TLS without and with SCSV/RIE countermeasures. Based on the TLS renegotiation attack of Ray and Dispensa, we see in Section 3 that TLS without countermeasures is not secure in our model for renegotiation. We subsequently show in Section 4 that, generically, TLS with the SCSV/RIE countermeasures of RFC 5746 [26] is a *weakly secure renegotiable ACCE protocol*. 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 *tagged TLS* where an arbitrary tag can be provided as an extension. 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

[1] 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*.

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 of `TLS_DHE_DSS_` [16].

Current formulations of ACCE focus on protocols with forward secrecy. Although ephemeral Diffie–Hellman TLS 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 [21]. We also describe how our approach to renegotiation could be extended if and when ciphersuites without forward secrecy, such as RSA key transport, are shown ACCE-like-secure.

New countermeasure for TLS. TLS with SCSV/RIE cannot meet our strongest notion of renegotiable security, only the weaker notion described above. 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 5 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 choice of ACCE. It would be desirable to prove the security of the TLS renegotiation countermeasures via some kind of composability framework, such as universal composability or the game-based composability framework of Brzuska *et al.* [6]. 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.

Of the two existing definitional approaches for proving the full security of the TLS protocol [16, 6], we chose the ACCE approach over the game-based composability approach because renegotiation in TLS makes extensive use of the interplay between the handshake and record layer.

2 Security Definitions for Multi-Phase and Renegotiable ACCE

In this section we describe what a multi-phase authenticated and confidential channel establishment (ACCE) protocol is and our various renegotiation security notions. Essentially, a multi-phase protocol can have many key exchanges—each called a *phase*—linked to a single session. Our definition builds on the ACCE definition of Jager *et al.* [16], which combined the Bellare–Rogaway model for authenticated key exchange [2] with a stateful variant of Paterson *et al.*'s length-hiding authenticated encryption [23], described in detail in Appendix A.2.

Notation. If S is a set, then $x \xleftarrow{\$} S$ denotes sampling a value x uniformly at random from S. $x \xleftarrow{\$} \mathcal{A}(y)$ denotes the output x of the probabilistic algorithm $\mathcal{A}$ when run on input y and randomly chosen coins. $\mathcal{A}^{\mathcal{O}(\cdot)}$ means $\mathcal{A}$ is run with access to oracle $\mathcal{O}(\cdot)$. The notation $[1, n]$ denotes the set $\{1, 2, \ldots, n\}$; $\texttt{phases}[\ell]$ denotes the ℓth entry in the array `phases` and $|\texttt{phases}|$ denotes the number of entries in the array.

2.1 Overview

The first security notion, a *secure multi-phase ACCE protocol*, 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.

Our central security definition is that of a *secure renegotiable ACCE protocol*, which strengthens the authentication notion: parties should successfully renegotiate only when they have 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.*

However, it is not possible to prove that TLS with the SCSV/RIE countermeasures is a secure renegotiable ACCE protocol: as we will see in Section 3, the 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. TLS's SCSV/RIE countermeasures do not fully protect against the case when these secret values are revealed.

As a result, we introduce the weaker, though still quite reasonable, notion of a *weakly secure renegotiable ACCE protocol*, and prove in Section 3 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.

2.2 Execution Environment

Parties. The environment consists of n_{pa} parties, $\{P_1, \ldots, P_{n_{\text{pa}}}\}$. Each party P_A is a potential protocol participant, and has a list of n_{ke} long-term key pairs $(pk_{A,1}, sk_{A,1}), \ldots, (pk_{A,n_{\text{ke}}}, sk_{A,n_{\text{ke}}})$. We assume that each party P_A is uniquely identified by any one of its public keys $pk_{A,*}$. In practice, there may be other identities that are bound to these public keys, e.g. by using certificates, but this is out of scope of this paper.

Sessions. Each party P_A can participate in up to n_{se} 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. Each session $s \in [1, n_{\text{se}}]$ is presented to the environment as an oracle π_A^s. Each oracle π_A^s records in a variable $\pi_A^s.d$ the oracle corresponding to the intended communication partner, e.g. $\pi_A^s.d = \pi_B^t$. 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 consist of up to n_{ph} phases. 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 $\pi_A^s.\texttt{phases}$ of different phase states is maintained; we sometimes use the notation $\pi_A^{s,\ell}$ for $\pi_A^s.\texttt{phases}[\ell]$. There can be at most n_{ph} phases per oracle. The last entry of $\pi_A^s.\texttt{phases}$ contains the state of the current phase, which may still be in progress. Each entry $\pi_A^s.\texttt{phases}[\ell]$ in the log contains:

- pk, the public key used by π_A^s in that phase,

- pk', the public key that π_A^s observed as being used for its peer in that phase[2],
- $\omega \in \{0,1\}$, denoting the authentication mode used, where 0 indicates that server-only authentication is used in that phase and 1 indicates mutual authentication,
- Δ, a counter used to keep track of the current status of the protocol execution,
- α, either `accept`, `reject`, or $\emptyset$ (for in-progress),
- k, the encryption and/or MAC key(s) established by π_A^s in that phase,
- T, the transcript of all (plaintext) messages sent and received by π_A^s during the pre-accept stage of that phase,
- RT_s and RT_r, the transcripts of all ciphertexts sent and received (respectively) in the post-accept phase by π_A^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 internal state is initialized to $d \leftarrow \emptyset$, $pk \leftarrow \emptyset$, $pk' \leftarrow \emptyset$, $\omega \leftarrow \emptyset$, $\Delta \leftarrow 1$, $\alpha \leftarrow \emptyset$, $k \leftarrow \emptyset$, $T \leftarrow \emptyset$, $RT \leftarrow \emptyset$, $b \xleftarrow{\$} \{0,1\}$, and $st \leftarrow \emptyset$. When describing a protocol, we will enumerate the protocol messages. The oracles keep track of the protocol execution by setting the counter state equal to the message number that the oracles expect to receive next, and update the counter on each message sent ($\Delta \leftarrow \Delta + 1$). Once a phase of a protocol accepts (that is, an encryption key has been negotiated and authentication is believed to hold), then α is set to `accept`. If the protocol rejects and the oracle wishes to discontinue operation, the counter Δ can be set to the special symbol `reject`. Whenever a new handshake initialization message is received, the oracle adds a new entry to its `phases` list. The variable ω is set at some point during (or before) the protocol execution, depending on the protocol specification (e.g. in case of TLS, the server can send the message `CertificateRequest` to request client, i.e. mutual, authentication, otherwise server-only authentication is used). 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 1. 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 equivalent to the original ACCE model of Jager *et al.* [16].

Adversarial interaction. The adversary interacts with oracles by issuing the following queries, which allow her to control (forward/alter/create/drop) all communication on the public channel (Send), learn parties' long-term secret keys (Corrupt), learn session keys (Reveal), and control sending and receiving of arbitrary messages on the encrypted record layer (Encrypt/Decrypt) using a stateful symmetric encryption scheme StE (Appendix A.2).

- $\mathsf{Send}(\pi_A^s, m)$: The adversary can use this query to send any (plaintext) message m of its choosing to (the current phase of) oracle π_A^s. The oracle will respond according to the protocol specification, depending on its internal state. Some distinguished control messages have special behaviour:
  - $m = (\texttt{newphase}, pk, \omega)$ triggers an 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 `ServerHelloRequest` message.
  - $m = (\texttt{ready}, pk, \omega)$ activates a (server) 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 ω 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 instead learned

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

$\mathsf{Encrypt}(\pi_A^s, ctype, m_0, m_1, len, head)$:
1. $u \leftarrow u + 1$
2. If ($ctype = \texttt{control}$) AND caller is not π_A^s, then return $\perp$
3. $(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, len, head, ctype \| m_0, st_e)$
4. $(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, len, head, ctype \| m_1, st_e)$
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

$\mathsf{Decrypt}(\pi_A^s, C, head)$:
1. $v \leftarrow v + 1$
2. $(ctype \| m, st_d) = \mathsf{StE.Dec}(k, head, C, st_d)$
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. Else $r' \leftarrow \perp$
7. Return (m', r')

where $(k, b, \mathsf{diverge}) = \pi_A^s.\texttt{phases}[\ell^*].(k, b, \mathsf{diverge})$ and ℓ^* is the last phase π_A^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.
Note: the "protocol response for m" may be encrypted by the party internally making an Encrypt call.

Figure 2: Encrypt and Decrypt oracles for the multi-phase/renegotiable ACCE security experiments.

during the run of the protocol: we are using a post-specified peer model [7]. 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 $(\texttt{ready}, \cdot)$.

- $\mathsf{Corrupt}(P_A, pk)$: Oracle π_A^1 responds with the long-term secret key $sk_{A,i}$ corresponding to public key $pk = pk_{A,i}$ of party P_A, or $\perp$ if there is no i such that $pk = pk_{A,i}$. This is the *weak corruption model*, meaning we do not allow the adversary to obtain the party's internal state nor register rogue keys.
- $\mathsf{Reveal}(\pi_A^s, \ell)$: Oracle π_A^s responds with the key(s) $\pi_A^s.\texttt{phases}[\ell].k$ used in phase ℓ, or $\emptyset$ if no such value exists. Since the TLS record layer is unidirectional, there are both encryption and decryption keys, and for most ciphersuites also MAC keys; in our case, all 4 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 this query, though one could imagine refinements if desired.
- $\mathsf{Encrypt}(\pi_A^s, ctype, m_0, m_1, len, head)$: This query depends on the random bit b sampled by π_A^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 $head$. Content type `control` is used for handshake messages. The adversary cannot query this oracle with $ctype = \texttt{control}$. Through an abuse of notation, we allow the party itself to call this oracle with `control` to encrypt protocol messages that must be sent encrypted; 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 2.
- $\mathsf{Decrypt}(\pi_A^s, C, head)$: This query takes as input a ciphertext C and header data $head$. If π_A^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 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, for which we give the definition and an explanation in Appendix A.2.

Let us review how an adversary would use the oracles to carry out a normal TLS negotiation and renegotiation. First the adversary uses the Send query to deliver `newphase` and `ready` messages to the client and server. The client responds to the Send query with a `ClientHello` message; the server responds with $\perp$. The adversary delivers the first message from the client to the server by calling the server's Send oracle, which returns the next message from the server to the client (`ServerHello`, `ServerKeyExchange`, etc.). The adversary delivers these to the client via a Send query. The client responds with several plain text messages (such as `ClientKeyExchange`) as well as a `ChangeCipherSpec` message. There is one more message, the client's `Finished` message, which the client first encrypts using an internal Encrypt call. The plaintext messages are delivered by the adversary to the server using Send and the encrypted message is

delivered using Decrypt. The response by the server from Send will be a `ChangeCipherSpec` message and the response by the server from Decrypt will the server's `Finished` message, which the server first encrypts using an internal Encrypt oracle call. The encrypted message is delivered by the adversary using Decrypt. The parties set $\alpha = \texttt{accept}$ and are now ready to use the record layer, which the adversary can make use of by matching Encrypt/Decrypt queries. When the adversary wants to trigger client-initiated renegotiation, it sends a `newphase` message via a Send query to the client, who responds with a `ClientHello` message encrypted via an internal Encrypt call. 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 transcript RT. When the parties accept in the phase, they begin using the encryption keys for the new phase.

2.3 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_A^s reaches the post-accept state only if there exists another party π_B^t such that π_A^s has a matching conversation (in the sense of [16], reproduced in Appendix A.1) to π_B^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 the introduction to Section 2.

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 $\{\pi_A^s : A \in [1, n_{\text{pa}}], s \in [1, n_{\text{se}}]\}$. At the beginning of the game, the challenger generates n_{ke} long-term key pairs $(pk_{A,1}, sk_{A,1}), \ldots, (pk_{A,n_{\text{ke}}}, sk_{A,n_{\text{ke}}})$ for each party P_A; we assume that, within a party, all public key pairs are distinct. (That distinct parties have distinct key pairs comes as a consequence of the protocol being secure.) The adversary receives all parties' public keys as input. The adversary may issue Send, Corrupt, Reveal, Encrypt, and Decrypt queries to the oracles and eventually terminates.

Table 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 1 (Correct multi-phase ACCE)**.** *We say* Π *is a* correct multi-phase ACCE protocol *if, for all oracles* π_A^s *with destination address* $\pi_A^s.d = \pi_B^t$*, and for all* $\ell, \ell' \in [1, n_{\text{ph}}]$ *for which* $\pi_A^s.\texttt{phases}[\ell].T$ *and* $\pi_B^t.\texttt{phases}[\ell'].T$ *are matching conversations, it holds that* $\pi_A^s.\texttt{phases}[\ell].\alpha = \pi_B^t.\texttt{phases}[\ell'].\alpha = \texttt{accept}$*,* $\pi_A^s.\texttt{phases}[\ell].\omega = \pi_B^t.\texttt{phases}[\ell].\omega$ *and* $\pi_A^s.\texttt{phases}[\ell].k = \pi_B^t.\texttt{phases}[\ell'].k$*.*

2.3.1 Confidentiality.

All of our notions for secure 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 Encrypt/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. As with the ACCE notion of Jager *et al.* [16], this notion ensures forward security: corrupting long-term secret keys after completion of a session should not impact confidentiality/integrity of messages. (We discuss how these definitions could be extended to the non-forward-secure case in Appendix B.)

Definition 2 (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* (A, s, ℓ, b')*. If*

C1. $\pi_A^s.\texttt{phases}[\ell].\alpha = \texttt{accept}$*; and*
C2. $\mathcal{A}$ *did not query* $\mathsf{Corrupt}(P_A, \pi_A^s.\texttt{phases}[\ell].pk)$ *before* π_A^s *accepted in phase* ℓ*; and*
C3. $\mathcal{A}$ *did not query* $\mathsf{Corrupt}(P_B, \pi_A^s.\texttt{phases}[\ell].pk')$ *before* π_A^s *accepted in phase* ℓ*, where* $\pi_A^s.d = \pi_B^t$*; and*
C4. $\mathcal{A}$ *did not query* $\mathsf{Reveal}(\pi_A^s, \ell)$*; and*
C5. $\mathcal{A}$ *did not query* $\mathsf{Reveal}(\pi_B^t, \ell')$*, where* $\pi_B^t = \pi_A^s.d$ *is* π_A^s*'s intended communication partner, and* ℓ' *is any phase for which* $\pi_B^t.\texttt{phases}[\ell'].T$ *is a matching conversation to* $\pi_A^s.\texttt{phases}[\ell].T$*; and*

C6. $|\Pr[\pi_A^s.\texttt{phases}[\ell].b = b'] - 1/2| \geq \epsilon$,
then we say that $\mathcal{A}$ (τ, ϵ)-breaks confidentiality/integrity of Π.

2.3.2 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 result in Section 4. 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).

Definition 3 (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_A^s *with* $\pi_A^s.d = \pi_B^t$ *and a phase* ℓ *such that*

A1. $\pi_A^s.\texttt{phases}[\ell].\alpha = \texttt{accept}$*; and*
A2. $\mathcal{A}$ *did not query* $\mathsf{Corrupt}(P_A, \pi_A^s.\texttt{phases}[\ell].pk)$ *before* π_A^s *accepted in phase* ℓ*; and*
A3. $\mathcal{A}$ *did not query* $\mathsf{Corrupt}(P_B, \pi_A^s.\texttt{phases}[\ell].pk')$ *before* π_A^s *accepted in phase* ℓ*; and*
A4. *if* $\pi_A^s.\texttt{phases}[\ell].\omega = 0$ *then* $\pi_A^s.\rho = \mathsf{Client}$*; and*
A5. $\mathcal{A}$ *did not query* $\mathsf{Reveal}(\pi_B^t, \ell')$ *before* π_A^s *accepted in phase* ℓ*, for any* ℓ' *such that* $\pi_B^t.\texttt{phases}[\ell'].T$ *is a matching conversation to* $\pi_A^s.\texttt{phases}[\ell].T$*; and*
M. *there is no* ℓ' *such that* $\pi_B^t.\texttt{phases}[\ell'].T$ *is a matching conversation to* $\pi_A^s.\texttt{phases}[\ell].T$

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

A protocol Π *is a* (τ, ϵ)-secure multi-phase ACCE protocol *if there exists no algorithm* $\mathcal{A}$ *that* (τ, ϵ)*-breaks confidentiality/integrity (Def. 2) or authentication (as defined above) of* Π.

In **A1** and **M** we redefine the `NoMatch`-condition from [2]. In **A2** we exclude leaking of the secret long-term keys of the accepting party (necessary for example to counter KCI attacks [20]). 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.

The secure multi-phase ACCE definition when limited to a phase per session and a single key pair per party ($n_{\text{ph}} = n_{\text{ke}} = 1$) collapses to an extension of the original ACCE definition, the extension being support for server-only authentication.

2.3.3 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 and all previous phases' handshake and record layer transcripts match, provided she has not impersonated the parties in the current phase. We will show in Section 5 that TLS with our proposed countermeasure satisfies this definition.

Definition 4 (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_A^s *with* $\pi_A^s.d = \pi_B^t$ *and a phase* ℓ^* *such that*

A1–A5 *as in Definition 3 with* ℓ^**, and either*
M′(a) $\pi_B^t.\texttt{phases}[\ell^*].T$ *is not a matching conversation to* $\pi_A^s.\texttt{phases}[\ell^*].T$ *or*
M′(b) *for some* $\ell < \ell^*$, $\pi_A^s.\texttt{phases}[\ell].T \| RT_s \| RT_r \neq \pi_B^t.\texttt{phases}[\ell].T \| RT_r \| RT_s$*;*

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

A protocol Π *is a* (τ, ϵ)-secure renegotiable ACCE protocol *if there exists no algorithm* $\mathcal{A}$ *that* (τ, ϵ)*-breaks confidentiality/integrity (Def. 2) or renegotiation authentication (as defined above) of* Π.

2.3.4 Weakly secure renegotiable ACCE.

Unfortunately, TLS, when combined with SCSV/RIE countermeasures, does not satisfy Def. 4 because, as we will see in Section 4.1, 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 5 (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_A^s with $\pi_A^s.d = \pi_B^t$ and a phase ℓ^* such that all conditions from Def. 4, as well as the following additional conditions are satisfied:*

A6. *$\mathcal{A}$ did not issue a $\mathsf{Reveal}(\pi_A^s, \ell)$ query before π_A^s accepted in phase $\ell + 1$, for every $\ell < \ell^*$, and*
A7. *$\mathcal{A}$ did not issue a $\mathsf{Reveal}(\pi_B^t, \ell)$ query before π_A^s accepted in phase $\ell + 1$, for every $\ell < \ell^*$;*

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

*A protocol Π is a (τ, ϵ)-*weakly secure renegotiable ACCE protocol *if there exists no algorithm $\mathcal{A}$ that (τ, ϵ) breaks confidentiality/integrity (Def. 2) or weak renegotiation authentication (as defined above) of Π.*

Remark 2. 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 Def. 5 still places no such restriction on previous encryption keys being revealed.

3 Renegotiation (In)security of TLS

In this section we discuss how the original TLS protocol, without SCSV/RIE countermeasures, fits into our model, and show how the attack of Ray and Dispensa is captured in the model.

Jager *et al.* [16] in the full version [17, Fig. 3] described how to map TLS into the ACCE model. We highlight a few components of that mapping, and the alterations due to our addition of renegotiation.

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 [10, §6.2.1]:

Packets with `ContentType=change_cipher_spec (20)` or `handshake (22)` are considered in our model to have *ctype* = `control` and packets with `ContentType=application_data (23)` are considered in our model to have *ctype* = `data`. We do not explicitly handle `ContentType=alert (21)` messages. The Reveal query reveals the encryption and MAC keys derived from the master secret key, *not* the master secret key itself.

3.1 TLS without countermeasures is not a (weakly) secure renegotiable ACCE protocol

Recall the TLS renegotiation attack by Ray and Dispensa [24], as described previously in Figure 1 on page 4. 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 will eventually establish a new record layer to which Eve has no access. Alice then transmits

	Secure multi-phase ACCE (Defn. 3)	**Weakly secure renegotiable ACCE** (Defn. 5)	**Secure renegotiable ACCE** (Defn. 4)
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. 2)	implied	implied	implied
`TLS_*` without countermeasures	—	× (Sect. 3.1)	× (Sect. 3.1)
Tagged-ACCE-fin-secure `TLS_*` with SCSV/RIE countermeasure	—	✓ (Thm. 1)	—
`TLS_RSA_` with SCSV/RIE countermeasure	?[1]	× (App. B.2) / ?[1]	× (Sect. 4.1)
`TLS_DHE_DSS_` with SCSV/RIE countermeasure	✓ (Cor. 2)	✓ (Cor. 2)	× (Sect. 4.1)
Secure multi-phase `TLS_*` with new (Sect. 5) countermeasure	—	—	✓ (Thm. 4)
`TLS_RSA_` with new (Sect. 5) countermeasure	?[1]	× (App. B.2) / ?[1]	× (App. B.2) / ?[1]
`TLS_DHE_DSS_` with new (Sect. 5) countermeasure	✓ (Thm. 4)	✓ (Thm. 4)	✓ (Thm. 4)

Table 1: Summary of security notions and results on TLS

[1] `TLS_RSA_` key transport ciphersuites may be able to be shown secure under notions with suitable restrictions on forward security; see discussion in Appendix B.

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 5 as follows. Assume Alice and Bob each have a single oracle instance, and Eve has carried out the above attack. 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 attacker has broken renegotiation authentication at both Alice and Bob's instances. At Alice by satisfying condition **M′(a)** (Alice's first handshake transcript does not match Bob's first handshake transcript), and at Bob by satisfying both **M′(a)** (Bob's second handshake transcript does not match Alice's second handshake transcript) and **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.

4 Renegotiation Security of TLS with SCSV/RIE Countermeasures

In this section we analyze the security of TLS with the SCSV/RIE countermeasures proposed in RFC 5746 [26]. We first see, in Section 4.1, that the SCSV/RIE countermeasures are not enough to prove that TLS satisfies our strongest notion, a secure renegotiable ACCE (Defn. 4).

Our goal, then, will be to show that TLS with the SCSV/RIE countermeasures is a weakly secure renegotiable ACCE. 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

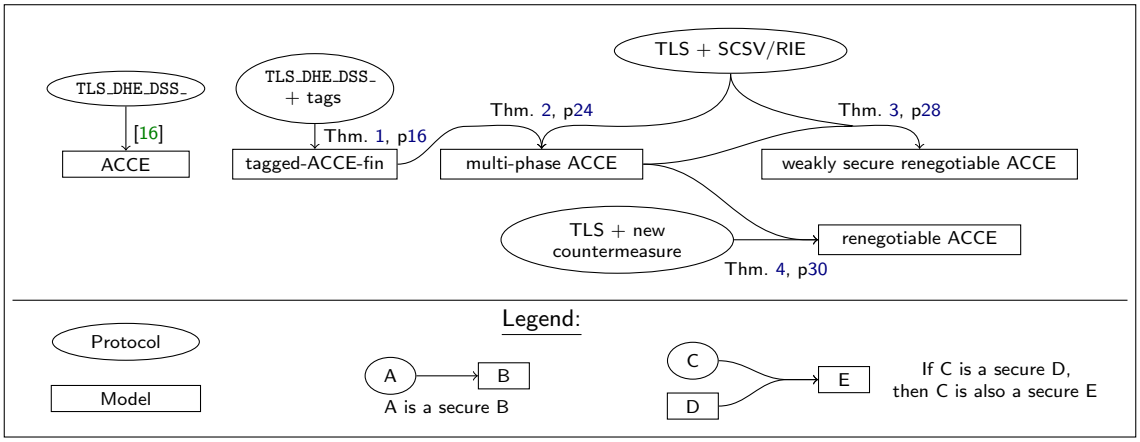

Figure 3: Summary of results on TLS and renegotiation

ACCE." As noted in the introduction, we 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, a white-box examination of the proof details of the only known proof of ACCE security of a TLS ciphersuite [16] observes that only small changes are needed to the proof to achieve this.

We will provide a sequence of definitions and results that justifies the security of the SCSV/RIE countermeasure. Figure 3 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.
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_DSS_` is ACCE secure can be modified in a very minor way to show that tagged `TLS_DHE_DSS_` is tagged-ACCE-fin-secure. For completeness, we give a full proof of this fact in the appendix.
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_DSS_` with SCSV/RIE countermeasures is a weakly secure renegotiable ACCE.

4.1 `TLS_*` with SCSV/RIE is not a secure renegotiable ACCE

Definition 4 requires that, even when the adversary can reveal previous phases' session keys, the parties will not successfully renegotiate if the attacker 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 message authentication codes, but Definition 4 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.

4.2 Tagged-ACCE security model and tagged TLS

In this section we introduce a variant of the ACCE model from which we can prove a generic result on the renegotiable security of TLS with countermeasures.

4.2.1 Tagged-ACCE security model

The *tagged-ACCE* security model is an extension of the ACCE security model to allow arbitrary tags as follows. Since the original ACCE definition of Jager *et al.* [16] does not support server-only authentication, while our definition allows both authentication modes, we extend the ACCE definition implied by limiting multi-phase ACCE (Definition 3) to a single phase ($n_{\text{ph}} = 1$) and at most one public key per party ($n_{\text{ke}} = 1$).

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

- $\mathsf{Send}(\pi_A^s, m)$. The adversary can specify an arbitrary tag during session initialization.
  - If $m = (\texttt{newphase}, \omega, tag)$, the party sets its internal variable $\rho \leftarrow \mathsf{Client}$, sets authentication mode ω, stores tag, and responds with the first protocol message.
  - If $m = (\texttt{ready}, \omega, tag)$, the party sets $\rho \leftarrow \mathsf{Server}$, authentication mode ω, stores tag, and responds with the next protocol message, if any.

The freshness and winning conditions of tagged-ACCE are unchanged from ACCE.

4.2.2 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. It adds the following query:

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

The following queries are modified:

- $\mathsf{Encrypt}(\pi_A^s, ctype, m_0, m_1, len, head)$: The adversary is not prevented from making queries with $ctype = \texttt{control}$.
- $\mathsf{Decrypt}(\pi_A^s, C, head)$: No semantic meaning is associated with $ctype = \texttt{control}$ messages. In other words, line 5 of Figure 2 is removed.

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

Remark 3. 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.

4.2.3 Tagged TLS

Figure 6 shows a generic TLS ciphersuite, along with the SCSV/RIE extensions denoted in green with a dagger. By *tagged TLS*, we mean the generic TLS ciphersuite from Figure 6, 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_DSS_`.

For completeness, we will show that TLS_DHE_DSS_ is a secure tagged-ACCE-fin protocol. The proof follows almost exactly the proof by Jager *et al.* [16] that TLS_DHE_DSS_ is a secure ACCE protocol. The intuition that leaking the Finished messages does not affect security is as follows. The ACCE proof of TLS_DHE_DSS_ 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 1 (Tagged TLS_DHE_DSS_ 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 $(\tau, \epsilon_{\mathsf{prf}})$-secure, the signature scheme is $(\tau, \epsilon_{\mathsf{sig}})$-secure, the* DDH*-problem is $(\tau, \epsilon_{\mathsf{ddh}})$-hard in the group G used to compute the TLS premaster secret, the hash function is $(\tau, \epsilon_{\mathsf{H}})$-collision resistant, and the* PRFODH*-problem is $(\tau, \epsilon_{\mathsf{prfodh}})$-hard with respect to G and* PRF*. Suppose that the stateful symmetric encryption scheme is $(\tau, \epsilon_{\mathsf{sLHAE}})$-secure.*

For any adversary that $(\tau', \epsilon_{\mathsf{tls}})$-breaks the tagged TLS_DHE_DSS_ *in the sense of Definition 3 in the tACCE execution environment with $\tau \approx \tau'$ it holds that*

$$\epsilon_{\mathsf{tls}} \leq \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^{\lambda-2}} + n_{\mathrm{pa}} n_{\mathrm{se}} \cdot \left(4 n_{\mathrm{pa}} \epsilon_{\mathsf{sig}} + 3\epsilon_{\mathsf{ddh}} + (n_{\mathrm{pa}} n_{\mathrm{se}} + 2)\left(\epsilon_{\mathsf{PRFODH}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^{\mu}} + \epsilon_{\mathsf{sLHAE}}\right)\right)$$

Recall that n_{pa} and n_{se} are the maximum number of parties and sessions per party; in tagged-ACCE-fin, the number of phases n_{ph} and number of keypairs n_{ke} are both at most 1*.*

To prove Theorem 1, we closely follow the approach of Jager *et al.* [17] and 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*.

Note that our proof proceeds exactly as the proof of Jager *et al.* to enable comparison, thus we also prove Theorem 1 by two lemmas. Lemma 1 bounds the probability ϵ_{auth} that an authentication-adversary succeeds, Lemma 2 bounds the probability ϵ_{enc} that an encryption-adversary succeeds. Then we have

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

Lemma 1. *For any adversary running in time $\tau' \approx \tau$, the probability that there exists an oracle π_i^s that accepts maliciously is at most*

$$\epsilon_{\mathsf{auth}} \leq \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^{\lambda-1}} + n_{\mathrm{pa}} n_{\mathrm{se}} \cdot \left(2 n_{\mathrm{pa}} \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + (n_{\mathrm{pa}} n_{\mathrm{se}} + 2)\left(\epsilon_{\mathsf{PRFODH}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^{\mu}} + \epsilon_{\mathsf{sLHAE}}\right)\right)$$

where all quantities are defined as stated in Theorem 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, 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

$$\epsilon_{\mathsf{client}} \leq \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^{\lambda}} + n_{\mathrm{pa}} n_{\mathrm{se}} \cdot \left(n_{\mathrm{pa}} \epsilon_{\mathsf{sig}} + n_{\mathrm{pa}} n_{\mathrm{se}}\left(\epsilon_{\mathsf{PRFODH}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^{\mu}} + \epsilon_{\mathsf{sLHAE}}\right)\right)$$

$$\epsilon_{\mathsf{server}} \leq \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^{\lambda}} + n_{\mathrm{pa}} n_{\mathrm{se}} \cdot \left(n_{\mathrm{pa}} \epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^{\mu}} + \epsilon_{\mathsf{sLHAE}}\right)$$

and thus

$$\begin{aligned}\epsilon_{\mathsf{auth}} &\leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} \\ &\leq \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^{\lambda-1}} + n_{\mathrm{pa}} n_{\mathrm{se}} \cdot \left(2n_{\mathrm{pa}}\epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + (n_{\mathrm{pa}} n_{\mathrm{se}} + 2)\left(\epsilon_{\mathsf{PRFODH}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^{\mu}} + \epsilon_{\mathsf{sLHAE}}\right)\right).\end{aligned}$$

4.2.4 Proof of Lemma 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 is negligible. The proof proceeds in a *sequence of games*, following [3, 27]. 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 with $\rho = \mathsf{Client}$ in Game δ.

Game 0. This game equals the multi-phase ACCE security experiment used in Section 2.2. 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 $n_{\mathrm{pa}} n_{\mathrm{se}}$ 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 $(n_{\mathrm{pa}} n_{\mathrm{se}})^2 \cdot 2^{-\lambda}$, which implies

$$\Pr[\mathsf{break}_0^{(2)}] \leq \Pr[\mathsf{break}_1^{(2)}] + \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^{\lambda}} \ .$$

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 $(i^*, s^*) \xleftarrow{\$} [n_{\mathrm{pa}}] \times [n_{\mathrm{se}}]$. 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 $1/(n_{\mathrm{pa}} \cdot n_{\mathrm{se}})$ we have $(i,s) = (i^*, s^*)$, and thus

$$\Pr[\mathsf{break}_1^{(2)}] = n_{\mathrm{pa}} n_{\mathrm{se}} \cdot \Pr[\mathsf{break}_2^{(2)}] \ .$$

Note that in this game the attacker 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 attacker 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 $r_C \| r_S \| p \| g \| T_S$, 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 not be corrupted when $\pi_{i*}^{s^*}$ accepts (as otherwise $\pi_{i*}^{s^*}$ does not accept *maliciously*). To show that $\Pr[\mathsf{abort}_{\mathsf{sig}}] \leq n_{\mathrm{pa}} \cdot \epsilon_{\mathsf{sig}}$, 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 $\phi \xleftarrow{\$} [n_{\mathrm{pa}}]$, 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 $1/(n_{\mathrm{pa}})$, 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 $\Pr[\mathsf{abort}_{\mathsf{sig}}]/(n_{\mathrm{pa}}) \leq \epsilon_{\mathsf{sig}}$; if $\Pr[\mathsf{abort}_{\mathsf{sig}}]$ is not negligible, then ϵ_{sig} is not negligible as well and we have

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

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 attacker. Note also that there may be multiple oracles that issued a signature σ_S containing r_C, since the attacker 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 $(j^*, t^*) \xleftarrow{\$} [n_{\mathrm{pa}}] \times [n_{\mathrm{se}}]$. It aborts, if the attacker 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

$$\Pr[\mathsf{break}_3^{(1)}] \leq n_{\mathrm{pa}} n_{\mathrm{se}} \Pr[\mathsf{break}_4^{(1)}] \ .$$

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 PRFODH 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 attacker 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 Server `Finished` message still cannot be predicted by an attacker. Remember that the Server `Finished` is computed as

$$fin_C^* = F(label_4 || \mathsf{H}(m_1 || \cdots || m_{12})),$$

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

$$\Pr[\mathsf{break}_8^{(1)}] \leq \Pr[\mathsf{break}_9^{(1)}] + \epsilon_{\mathsf{sLHAE}} = \epsilon_{\mathsf{sLHAE}} \ .$$

□

4.2.5 Proof of Lemma 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 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 with $\rho = \mathsf{Server}$ in Game δ.

Game 0. This game equals the ACCE security experiment described in Definition 3. 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

$$\Pr[\mathsf{break}_0^{(2)}] \leq \Pr[\mathsf{break}_1^{(2)}] + \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^\lambda} \ .$$

Game 2. This game is identical, except for the following. The challenger guesses three random indices $(i^*, s^*) \xleftarrow{\$} [n_{\mathrm{pa}}] \times [n_{\mathrm{se}}]$. 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 $1/(n_{\mathrm{pa}} n_{\mathrm{se}})$ we have $(i, s) = (i^*, s^*)$, and thus

$$\Pr[\mathsf{break}_1^{(2)}] = (n_{\mathrm{pa}} n_{\mathrm{se}}) \cdot \Pr[\mathsf{break}_2^{(2)}] \ .$$

Note that in this game the attacker 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 $\mathsf{abort}_{\mathsf{sig}}$, if oracle $\pi_{i*}^{s^*}$ ever receives as input a message $m_7 = cert_C$ indicating intended partner $\Pi = j$ and message $m_9 = \sigma_C = \mathsf{SIG.Sign}(sk_C, m_1 \| \ldots, \| m_8)$ such that σ_C is a valid signature but there exists no oracle π_j^t which has previously output σ_C. Clearly we have

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

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 not be corrupted when $\pi_{i*}^{s^*}$ accepts. To show that $\Pr[\mathsf{abort}_{\mathsf{sig}}] \leq (n_{\mathrm{pa}}) \cdot \epsilon_{\mathsf{sig}}$, 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 $\phi \xleftarrow{\$} [n_{\mathrm{pa}}]$, 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 $1/(n_{\mathrm{pa}})$, 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 $\Pr[\mathsf{abort}_{\mathsf{sig}}]/(n_{\mathrm{pa}}) \leq \epsilon_{\mathsf{sig}}$. Therefore if $\Pr[\mathsf{abort}_{\mathsf{sig}}]$ is not negligible, then ϵ_{sig} is not negligible as well and we have

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

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 attacker. 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 Client `Finished` message still cannot be predicted by an attacker. Remember that the Client `Finished` is computed as

$$fin_S^* = F(label_3 || \mathsf{H}(m_1 || \cdots || m_{10})),$$

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 Client `Finished` 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 1. We obtain that

$$\begin{aligned}\epsilon_{\mathsf{auth}} &\leq \epsilon_{\mathsf{client}} + \epsilon_{\mathsf{server}} \\ &\leq \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^{\lambda-1}} + n_{\mathrm{pa}} n_{\mathrm{se}} \cdot \left(2n_{\mathrm{pa}}\epsilon_{\mathsf{sig}} + \epsilon_{\mathsf{ddh}} + (n_{\mathrm{pa}} n_{\mathrm{se}} + 2)\left(\epsilon_{\mathsf{PRFODH}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^\mu} + \epsilon_{\mathsf{sLHAE}}\right)\right).\end{aligned}$$

4.2.6 Proof of Confidentiality

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

$$\epsilon_{\mathsf{enc}} \leq \epsilon_{\mathsf{auth}} + n_{\mathrm{pa}} n_{\mathrm{se}} \left(\epsilon_{\mathsf{ddh}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}\right) \ ,$$

where ϵ_{auth} is an upper bound on the probability that there exists an oracle that accepts maliciously in the sense of Definition 3 (cf. Lemma 1) and all other quantities are defined as stated in Theorem 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 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 4.2.1. 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 any phase in the sense of Definition 5. 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 (cf. Lemma 1).

Recall that we assume that $\mathcal{A}$ always outputs (i, s, b') such that all conditions in Property 2 of Definition 3 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 $\pi_i^s.\texttt{phases}[1].T$ has a matching conversation to $\pi_j^t.\texttt{phases}[1].T$, as the game is aborted otherwise.

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

$$\mathsf{Adv}_1 \leq n_{\mathrm{pa}} n_{\mathrm{se}} \mathsf{Adv}_2 \ .$$

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 attacker $\mathcal{A}$ 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 $\mathsf{Encrypt}$ 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 attacker $\mathcal{A}$ having advantage ϵ' yields an attacker $\mathcal{B}$ against the sLHAE security of the encryption scheme with success probability at least $1/2 + \epsilon'$.

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

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

□

Adding up probabilities from Lemmas 1 and 2, we obtain that

$$\begin{aligned}
\epsilon_{\mathsf{tls}} &\leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{enc}} \\
&\leq 2\epsilon_{\mathsf{auth}} + n_{\mathrm{pa}} n_{\mathrm{se}} \left(\epsilon_{\mathsf{ddh}} + 2\epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{sLHAE}}\right) \\
&\leq \frac{(n_{\mathrm{pa}} n_{\mathrm{se}})^2}{2^{\lambda-2}} + n_{\mathrm{pa}} n_{\mathrm{se}} \cdot \left(4n_{\mathrm{pa}}\epsilon_{\mathsf{sig}} + 3\epsilon_{\mathsf{ddh}} + (n_{\mathrm{pa}} n_{\mathrm{se}} + 2)\left(\epsilon_{\mathsf{PRFODH}} + \epsilon_{\mathsf{prf}} + \epsilon_{\mathsf{H}} + \frac{1}{2^{\mu}} + \epsilon_{\mathsf{sLHAE}}\right)\right)
\end{aligned}$$

which yields Theorem 1.

Note, that we do lose some tightness compared to the original ACCE proof of `TLS_DHE_DSS_`. 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.

4.3 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 2 (TLS with SCSV/RIE is a secure-multi-phase ACCE)**.** *Let* Π *be a generic tagged TLS ciphersuite as described in Section 4.2. Assume that* Π *is* $(\tau, \epsilon_{\mathsf{tagged}})$*-tagged-ACCE-fin-secure. Let* Π' *denote* Π *with the SCSV/RIE countermeasure as described in Figure 6. For any adversary that* $(\tau', \epsilon_{\mathsf{mp}})$*-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* n_{pa} *in* ϵ *with* $n_{\mathrm{pa}} \cdot n_{\mathrm{ke}}$ *and replacing all instances of* n_{se} *in* ϵ *with* $n_{\mathrm{se}} \cdot n_{\mathrm{ph}}$*. (Recall that* $n_{\mathrm{pa}}, n_{\mathrm{se}}, n_{\mathrm{ph}}$*, and* n_{ke} *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 (A, pk) in the multi-phase ACCE experiment, $\mathcal{S}$ will allocate a distinct party, abstractly denoted $A|pk$, 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^s_A, let pk denote $\pi^s_A.\texttt{phases}[\ell].pk$, pk^* denote $\pi^s_A.\texttt{phases}[\ell+1].pk$, and suppose $\pi^s_A.d = \pi^t_B$.

- $\mathsf{Send}(\pi^s_A, m)$: 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 $m = (\texttt{newphase}, pk, \omega)$ and $\pi^s_A.\texttt{phases}$ is empty: $\mathcal{S}$ issues a $\mathsf{Send}(\pi^{s|1}_{A|pk}, (\texttt{newphase}, \omega, \texttt{empty}))$ query to $\mathcal{C}$ and returns the result.
  - If $m = (\texttt{ready}, pk, \omega)$ and $\pi^s_A.\texttt{phases}$ is empty: $\mathcal{S}$ issues a $\mathsf{Send}(\pi^{s|1}_{A|pk}, (\texttt{ready}, \omega, \texttt{empty})$ query to $\mathcal{C}$; no result is received or returned.
  - If $m = m_1 = (r_C, \texttt{cs-list}, ext_C)$ in Figure 6, $\mathcal{S}$ aborts if $ext_C \neq \texttt{empty}$. Otherwise, $\mathcal{S}$ sends issues a $\mathsf{Send}(\pi^{s|1}_{A|pk}, (r_C, \texttt{cs-list}))$ query to $\mathcal{C}$ and returns the result.
  - If $m = m_2 \| \ldots \| m_6$ and $m_2 = (r_C, \texttt{cs-list}, ext_S)$ in Figure 6, $\mathcal{S}$ aborts if $ext_S \neq \texttt{empty}$. Otherwise, $\mathcal{S}$ sets $m'_2 = (r_C, \texttt{cs-list})$ and issues a $\mathsf{Send}(\pi^{s|1}_{A|pk}, m'_2 \| \ldots \| m_6)$ query to $\mathcal{C}$ and returns the result.
  - If $m = m_7 \| \ldots \| m_{11}$ 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 $m = (\texttt{newphase}, pk^*, \omega)$, $\pi^s_A.\texttt{phases}$ is not empty, and $\pi^s_A.\rho = \mathsf{Client}$: $\mathcal{S}$ issues a $\mathsf{RevealFin}(\pi^{s|\ell}_{A|pk})$ query to $\mathcal{C}$ to obtain $fin_C \| fin_S$. $\mathcal{S}$ starts in `phases` a new phase $\ell + 1$ of π^s_A with authentica-

tion mode ω and public key pk^*. $\mathcal{S}$ obtains handshake message m_1^* by issuing a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, (\texttt{newphase}, \omega, fin_C))$ query to $\mathcal{C}$. $\mathcal{S}$ then issues an $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, \texttt{control}, m_1^*, m_1^*, len, head)$ query to $\mathcal{C}$ and returns the result.
  - If $m = (\texttt{newphase}, pk^*, \omega)$, $\pi_A^s.\texttt{phases}$ is not empty, and $\pi_A^s.\rho = \mathsf{Server}$: $\mathcal{S}$ starts in $\texttt{phases}$ a new phase $\ell+1$ of π_A^s with authentication mode ω and public key pk^*, sets $m_0^* = \texttt{ServerHelloRequest}$, issues an $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, \texttt{control}, m_0^*, m_0^*, len, head)$ query to $\mathcal{C}$, and returns the result.
  - If $m = (\texttt{ready}, pk^*, \omega)$ and $\pi_A^s.\texttt{phases}$ is not empty: $\mathcal{S}$ starts in $\texttt{phases}$ a new phase $\ell+1$ of π_A^s with authentication mode ω and public key pk^*. $\mathcal{S}$ issues a $\mathsf{RevealFin}(\pi_{A|pk}^{s|\ell})$ query to $\mathcal{C}$ to obtain $fin_C \| fin_S$. $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, (\texttt{ready}, \omega, fin_S))$ query to $\mathcal{C}$; no result is received or returned.

  The actual handshake messages in renegotiation handshakes are delivered using $\mathsf{Decrypt}$ queries.
- $\mathsf{Corrupt}(P_A, pk)$: $\mathcal{S}$ issues a $\mathsf{Corrupt}(P_{A|pk})$ query to $\mathcal{C}$ and returns the result.
- $\mathsf{Reveal}(\pi_A^s, \ell)$: $\mathcal{S}$ issues a $\mathsf{Reveal}(\pi_{A|pk}^{s|\ell})$ query to $\mathcal{C}$ and returns the result.
- $\mathsf{Encrypt}(\pi_A^s, ctype, m_0, m_1, len, head)$: $\mathcal{S}$ aborts if $ctype = \texttt{control}$. Otherwise, $\mathcal{S}$ issues an $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, m_0, m_1, len, head)$ query to $\mathcal{C}$ and returns the result.
- $\mathsf{Decrypt}(\pi_A^s, C, head)$: First suppose that all of conditions **C2**–**C5** of Definition 2; 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 $\pi_A^s.\rho = \mathsf{Server}$ and the last query that π_A^s received was $\mathsf{Send}(\pi_A^s, (\texttt{ready}, \dots))$ or $\mathsf{Send}(\pi_A^s, (\texttt{newphase}, \dots))$. If C does not equal the last ciphertext that was sent by π_B^t, then abort. Otherwise, $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C, head)$ query to $\mathcal{C}$. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, m_1^*)$ query to $\mathcal{C}$, where m_1^* is the handshake message obtained by $\mathcal{S}$ from $\mathcal{C}$ in the $\mathsf{Send}(\pi_B^t, (\texttt{newphase}, \dots))$ query above. $\mathcal{S}$ receives $m_2^* \| \dots \| m_6^*$ from $\mathcal{C}$, encrypts them by issuing an $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, \texttt{control}, m_2^* \| \dots \| m_6^*, \dots)$ query, and returns the resulting ciphertext C'.[3]
  - Suppose $\pi_A^s.\rho = \mathsf{Client}$ and the last query that π_A^s received was $\mathsf{Send}(\pi_A^s, (\texttt{ready}, \dots))$. If C does not equal the last ciphertext that was sent by π_B^t, then abort. Otherwise, $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C, head)$ query to $\mathcal{C}$. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, m_2^* \| \dots \| m_6^*)$ query to $\mathcal{C}$, where $m_2^* \| \dots \| m_6^*$ are the handshake messages obtained by $\mathcal{S}$ from $\mathcal{C}$ in the $\mathsf{Decrypt}(\pi_B^t, \dots)$ query in the bullet point immediately preceding this one. $\mathcal{S}$ receives $m_7^* \| \dots \| m_{11}^*$ from $\mathcal{C}$, encrypts $m_7^* \| \dots \| m_{10}^*$ by issuing an $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, \texttt{control}, m_7^* \| \dots \| m_{10}^*, \dots)$ query, and returns the resulting ciphertext C' along with m_{11}^*.
  - Suppose $\pi_A^s.\rho = \mathsf{Server}$ and the last query that π_A^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_B^t, then abort. Otherwise, $\mathcal{S}$ splits the ciphertext as $C^* \| m_{11}^*$ and issues a $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C^*, head)$ query. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, m_7^* \| \dots \| m_{11}^*)$ query to $\mathcal{C}$, where $m_7^* \| \dots \| m_{10}^*$ are the handshake messages obtained by $\mathcal{S}$ from $\mathcal{C}$ in the $\mathsf{Decrypt}(\pi_B^t, \dots)$ 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 $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, \texttt{control}, m_{12}^*, \dots)$ query, and returns the resulting ciphertext C' along with m_{13}^*.
  - Suppose $\pi_A^s.\rho = \mathsf{Client}$ and the last query that π_A^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_B^t, then abort. Otherwise, $\mathcal{S}$ splits the ciphertext as $C^* \| m_{13}^*$ and issues a $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C^*, head)$ query. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, m_{12}^* \| \dots \| m_{13}^*)$ query to $\mathcal{C}$, where m_{12}^* is the handshake message obtained by $\mathcal{S}$ from $\mathcal{C}$ in the $\mathsf{Decrypt}(\pi_B^t, \dots)$ query in the bullet point immediately preceding this one.
  - All other $\mathsf{Decrypt}(\pi_A^s, \dots)$ queries: $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C, head)$ query to $\mathcal{C}$ and returns the result.

[3] 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.

Now suppose otherwise, namely that one or more of conditions **C2**–**C5** of Defintion 2 is violated, so either the phase owner's public key pk or the peer's public key pk' has been corrupted, or either $\mathsf{Reveal}(\pi_B^t, \ell')$ or $\mathsf{Reveal}(\pi_B^t, \ell')$ has been called.

If it has not already done so, $\mathcal{S}$ issues a $\mathsf{Reveal}(\pi_{A|pk}^{s|\ell})$ 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 appropriate encryption key, $\mathcal{S}$ steps through all previous calls to $\mathsf{Decrypt}(\pi_A^s, \dots)$ to bring the decryption state st_d up-to-date. Now we explain each part of the renegotiation handshake:

- Suppose $\pi_A^s.\rho = \mathsf{Server}$ and the last query that π_A^s received was $\mathsf{Send}(\pi_A^s, (\texttt{ready}, \dots))$ or $\mathsf{Send}(\pi_A^s, (\texttt{newphase}, \dots))$. $\mathcal{S}$ 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 $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C, head)$ query to $\mathcal{C}$. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, m_1^*)$ query to $\mathcal{C}$. $\mathcal{S}$ receives $m_2^*\|\dots\|m_6^*$ from $\mathcal{C}$, encrypts them by issuing an $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, \texttt{control}, m_2^*\|\dots\|m_6^*, \dots)$ query, and returns the resulting ciphertext C'.
- Suppose $\pi_A^s.\rho = \mathsf{Client}$ and the last query that π_A^s received was $\mathsf{Send}(\pi_A^s, (\texttt{ready}, \dots))$. $\mathcal{S}$ 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 $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C, head)$ query to $\mathcal{C}$. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, m_2^*\|\dots\|m_6^*)$ query to $\mathcal{C}$. $\mathcal{S}$ receives $m_7^*\|\dots\|m_{11}^*$ from $\mathcal{C}$, encrypts $m_7^*\|\dots\|m_{10}^*$ by issuing an $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, \texttt{control}, m_7^*\|\dots\|m_{10}^*, \dots)$ query, and returns the resulting ciphertext C' along with m_{11}^*.
- Suppose $\pi_A^s.\rho = \mathsf{Server}$ and the last query that π_A^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 $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C^*, head)$ query to $\mathcal{C}$. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, m_7^*\|\dots\|m_{11}^*)$ query to $\mathcal{C}$. $\mathcal{S}$ receives $m_{12}^*\|m_{13}^*$ from $\mathcal{C}$, encrypts m_{12}^* by issuing an $\mathsf{Encrypt}(\pi_{A|pk}^{s|\ell}, \texttt{control}, m_{12}^*, \dots)$ query, and returns the resulting ciphertext C' along with m_{13}^*.
- Suppose $\pi_A^s.\rho = \mathsf{Client}$ and the last query that π_A^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 $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C^*, head)$ query. (Note this returns $\perp$.) Then $\mathcal{S}$ issues a $\mathsf{Send}(\pi_{A|pk^*}^{s|\ell+1}, m_{12}^*\|\dots\|m_{13}^*)$ query to $\mathcal{C}$.
- All other $\mathsf{Decrypt}(\pi_A^s, \dots)$ queries: $\mathcal{S}$ issues a $\mathsf{Decrypt}(\pi_{A|pk}^{s|\ell}, C, head)$ 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 $\pi_A^s.pk$ 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 2 were satisfied. And under the assumption that C is a valid ciphertext, if $\mathcal{S}$ was to make a query $\mathsf{Decrypt}(\pi_A^s, C, head)$ to $\mathcal{C}$, then $\mathcal{S}$ would receive either $\perp$ if $\mathcal{C}$'s secret challenge bit b in $\pi_{A|pk}^{s|\ell}$ is 0, or $m \neq \perp$ if the secret challenge bit $b = 1$. In other words, $\mathcal{S}$ can (τ', 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 2), namely by outputting a tuple (A, s, ℓ, b') for which $b' = \pi_A^s.\texttt{phases}[\ell].b$, and that adversary never violated conditions **C1**–**C6** of Definition 2. Let pk denote $\pi_A^s.\texttt{phases}[\ell].pk$. Since $\mathcal{S}$ only violates conditions **C1**–**C6** for $\pi_{A|pk}^{s|\ell}.\texttt{phases}[1]$ when the adversary violates them for $\pi_A^s.\texttt{phases}[\ell]$, $\mathcal{S}$ has not violated these conditions either. The simulator $\mathcal{S}$ outputs the tuple $((A|pk), (s|\ell), 1, b')$; it holds that $b' = \pi_{A|pk}^{s|\ell}.\texttt{phases}[1].b$ in Π. Thus, we have that $\mathcal{S}$ $(\tau', \epsilon_{\mathsf{c}})$-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 n_{pa} in ϵ with $n_{\mathrm{pa}} \cdot n_{\mathrm{ke}}$ and replacing all instances of n_{se} in ϵ with $n_{\mathrm{se}} \cdot n_{\mathrm{ph}}$.

Authentication. Now suppose that the multi-phase ACCE adversary for Π' succeeds in breaking authentication (as in Definition 3, namely by causing to exist a phase $\pi_A^s.\mathtt{phases}[\ell]$ 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 $\pi_{A|pk}^{s|\ell}.\mathtt{phases}[1]$ when the adversary violates them for $\pi_A^s.\mathtt{phases}[\ell]$, $\mathcal{S}$ has not violated these conditions either. Moreover, condition **A4** for $\pi_{A|pk}^{s|\ell}.\mathtt{phases}[1]$ is satisfied precisely when $\pi_A^s.\mathtt{phases}[\ell]$ is. Thus, we have that $\mathcal{S}$ $(\tau', \epsilon_{\mathsf{a}})$-breaks the (tagged-ACCE-fin) authentication of Π, where $\tau \approx \tau'$ and ϵ_{a} is obtained from ϵ by replacing n_{pa} and n_{se} in ϵ as in the previous paragraph.

The result follows. □

Remark 4. 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.

4.4 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 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. If* Π *is a* $(\tau, \epsilon_{\mathsf{mp}})$*-secure multi-phase ACCE protocol, and* PRF *is a* $(\tau, \epsilon_{\mathsf{prf}})$*-secure pseudorandom function, then* Π *is a* (τ, 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 — M′(a). Suppose $\mathcal{A}$ wins the weak renegotiable ACCE security experiment for Π using condition **M′(a)**. Either there is no ℓ at all such that $\pi_B^t.\texttt{phases}[\ell].T$ matches $\pi_A^s.\texttt{phases}[\ell^*].T$, or there is such an ℓ but $\ell \neq \ell^*$.

First consider the case where there is no ℓ at all such that $\pi_B^t.\texttt{phases}[\ell].T$ matches $\pi_A^s.\texttt{phases}[\ell^*].T$. That meets condition **M** of Definition 3 for Π.

Now consider the case where there is an ℓ such that $\pi_B^t.\texttt{phases}[\ell].T$ matches $\pi_A^s.\texttt{phases}[\ell^*].T$ but $\ell \neq \ell^*$. Assume without loss of generality $\ell < \ell^*$ (otherwise we could swap the oracles).

There must exist some value $j \in [1, \ell - 1]$ such that $\pi_A^s.\texttt{phases}[\ell^* - j].T \neq \pi_B^t.\texttt{phases}[\ell - j].T$. In particular, $j \leq \ell - 1$, since in π_B^t's first phase its outgoing message m_1 contains $ext_C = \texttt{empty}$ but π_A^s received a message m_1 with $ext_c \neq \texttt{empty}$. Let j be minimal. Then $\pi_B^t.\texttt{phases}[\ell - j + 1].T$ matches $\pi_A^s.\texttt{phases}[\ell^* - j + 1].T$. 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 $fin_C^{(-1)}$ and $fin_S^{(-1)}$, and since π_A^{s,ℓ^*-j+1} accepted, both π_A^{s,ℓ^*-j+1} and $\pi_B^{t,\ell-j+1}$ used the same $fin_C^{(-1)}$ and $fin_S^{(-1)}$ values. But at each party, $fin_C^{(-1)}$ and $fin_S^{(-1)}$ are the hash (using a PRF) of the handshake transcripts from phases π_A^{s,ℓ^*-j} and $\pi_B^{t,\ell-j}$, 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 **M′(a)** and **A1**–**A7**.

Authentication — M′(b). Now suppose $\mathcal{A}$ wins the weak renegotiable ACCE security experiment for Π using condition **M′(b)** but not **M′(a)**. In particular, for every $\ell' < \ell^*$, $\pi_A^s.\texttt{phases}[\ell'].T = \pi_B^t.\texttt{phases}[\ell'].T$ *but* there is some $\ell < \ell^*$ such that $\pi_A^s.\texttt{phases}[\ell].RT_s \| RT_r \neq \pi_B^t.\texttt{phases}[\ell].RT_r \| RT_s$. Choose ℓ minimal. Let v be the smallest index such that the vth ciphertext C_v of $\pi_A^s.\texttt{phases}[\ell].RT_s \| RT_r$ is not equal to the vth ciphertext of $\pi_B^t.\texttt{phases}[\ell].RT_r \| RT_s$.

Assume without loss of generality that C_v was received by π_A^s as the vth ciphertext but was not sent by π_B^t as the vth ciphertext. (The alternative is that C_v was *sent* by π_A^s as the vth ciphertext but was not received by π_B^t as the vth ciphertext. However, we could then focus on everything from π_B^t's perspective and apply the same argument.)

This means that when $\mathcal{A}$ called $\mathsf{Decrypt}(\pi_A^s, C_v, head)$, if $b = 0$ then Decrypt returned $(\perp, \cdot)$, whereas if $b = 1$ then Decrypt returned $(m', \cdot)$ where $m' \neq \perp$. Our simulator can thus output (A, 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. We need to show that conditions **C1**–**C5** also hold for (A, s, ℓ).

Since $\mathcal{A}$ wins the weak renegotiable ACCE experiment using condition **M′(b)**, we have that **A1**–**A7** all hold. We want to show that, at the time that π_A^s accepted in phase $\ell + 1$, conditions **C1**–**C5** also hold for (A, s, ℓ).

- **C1**: **A1** directly implies **C1**, since if π_A^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 5 do not imply that $\mathcal{A}$ did not ask Corrupt queries prohibited by **C2** and **C3**. However, we do have that $\pi_A^s.\texttt{phases}[\ell].T = \pi_B^t.\texttt{phases}[\ell].T$; 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 Corrupt queries for public keys used in phase ℓ until after π_A^s accepts in phase ℓ.
- **C4**: **A6** directly implies **C4**, at the time that π_A^s accepted.
- **C5**: Since π_A^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 $\pi_A^s.\texttt{phases}[\ell'].T = \pi_A^s.\texttt{phases}[\ell].T$. By **A7**, $\mathcal{A}$ did not issue $\mathsf{Reveal}(\pi_B^t, \ell)$ before π_A^s accepted in phase $\ell + 1$. Thus at the time that π_A^s accepted, $\mathcal{A}$ did not issue $\mathsf{Reveal}(\pi_B^t, \ell')$ to any phase with $\pi_B^t.\texttt{phases}[\ell'].T = \pi_A^s.\texttt{phases}[\ell].T$, satisfying condition **C5**.

Thus, assuming Π is a secure multi-phase ACCE no $\mathcal{A}$ can achieve conditions **M′(b)** and **A1**–**A7**. □

We can combine Theorems 2 and 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 *as described in Section 4.2 is a secure tagged-ACCE-fin protocol and* PRF *is a secure pseudorandom function,*

then that TLS ciphersuite Π *with SCSV/RIE countermeasures as described in Figure 6 is a weakly secure renegotiable ACCE.* □

For concreteness, we can combine this with Theorem 1 to obtain:

Corollary 2 (`TLS_DHE_DSS_` with SCSV/RIE is a weakly secure renegotiable ACCE)**.** *Under the same assumptions on the building blocks as in Theorem 1,* `TLS_DHE_DSS_` *with SCSV/RIE countermeasures is a weakly secure renegotiable ACCE protocol.* □

5 Renegotiation Security of TLS with a New Countermeasure

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 4. 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` ($fin_C^{(-1)}$) and `server_verify_data` ($fin_S^{(-1)}$) 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:

$$ext_C \leftarrow fin_C^{(-1)} \parallel \mathsf{PRF}(ms^{(-1)}, label_5 \| H(RT_s^{(-1)} \| RT_r^{(-1)})) \ , \tag{1}$$

where $ms^{(-1)}$ is the previous phase's master secret, H is a collision-resistant hash function, and $RT_s^{(-1)} \| RT_r^{(-1)}$ is the client's view of the previous phase's record layer transcript; the server uses $RT_r^{(-1)} \| RT_s^{(-1)}$ 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 5. 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 3 would go through if only ext_C contained $fin_S^{(-1)}$. However, if the security model is altered to allow Corrupts 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 [15].

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 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 but using* ext_C *as in equation (1). If* Π *is a* $(\tau, \epsilon_{\mathsf{mp}})$*-secure multi-phase ACCE protocol,* H *is a* $(\tau, \epsilon_{\mathsf{h}})$*-collision-resistant hash function, and* PRF *is a* $(\tau, \epsilon_{\mathsf{prf}})$*-secure pseudorandom function, then* Π *is a* (τ, ϵ)*-secure renegotiable ACCE, where* $\epsilon = \epsilon_{\mathsf{mp}} + \epsilon_{\mathsf{h}} + \epsilon_{\mathsf{prf}}$*.*

The proof proceeds similarly to that of Theorem 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 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 the PRF, 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 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 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 3.

If $\mathcal{A}$ wins using condition $\mathbf{M'(b)}$ but not $\mathbf{M'(a)}$, then, for every $\ell' < \ell^*$, $\pi_A^s.\texttt{phases}[\ell'].T = \pi_B^t.\texttt{phases}[\ell'].T$. But there is some $\ell < \ell^*$ such that $\pi_A^s.\texttt{phases}[\ell].RT_s \| RT_r \neq \pi_B^t.\texttt{phases}[\ell].RT_r \| RT_s$. Choose ℓ *maximal*. Then $\pi_B^t.\texttt{phases}[\ell+1].T$ matches $\pi_A^s.\texttt{phases}[\ell+1].T$. 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 contains $\mathsf{PRF}(ms^{(-1)}, label_5 \| H(RT_s^{(-1)} \| RT_r^{(-1)}))$. Since $\pi_A^{s,\ell+1}$ accepted, both $\pi_A^{s,\ell+1}$ and $\pi_B^{t,\ell+1}$ 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 $\pi_A^{s,\ell}$ and $\pi_B^{t,\ell}$, 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**. □

6 Conclusion

Although two-party protocols for establishing secure communication have been extensively studied in the literature and are widely used in practice, this is the first work to consider the important practical issue of renegotiation, in which parties update one or more aspects of their connection — authentication credentials, cryptographic parameters, or simply refresh their session key. The importance of correctly implementing renegotiation was highlighted by the 2009 attack of Ray and Dispensa on how certain applications process data from renegotiable TLS connections.

We have developed a formal model for describing the security of renegotiable cryptographic protocols, focussing on authenticated and confidential channel establishment (ACCE) protocols. We have specifically analyzed renegotiation in the TLS protocol, identifying the original attack of Ray and Dispensa in our model. We have provide a generic proof that the SCSV/RIE countermeasure offers good protection against renegotiation attacks, and give a new countermeasure that provides renegotiation security even in the face of slightly stronger adversaries.

Renegotiation, reauthentication, and rekeying are important features of many other applied cryptographic protocols. Future applied work includes examining the security of rekeying in protocols such as SSH or IKEv2 in our model. Open theoretical questions include how to adapt our approach for defining secure renegotiation to other primitives, in particular authenticated key exchange protocols. The overall security of TLS still has many important open questions, including the security of other TLS ciphersuites and the formal analysis of other complex functionality such as alerts and error messages. TLS session resumption [10, §F.1.4] is another important functionality of TLS, and it appears that our multi-phase ACCE model may be the right model in which to analyze its security, another interesting open problem. Given that attacks continue to be found

outside the core key agreement component of TLS, further research into modelling the security of TLS in increasingly realistic scenarios is well-motivated.

Acknowledgements

The authors gratefully acknowledge helpful discussions with Colin Boyd, Cas Cremers, Kenny Paterson, and Jörg Schwenk, and the advice of anonymous reviewers.

References

[1] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology – CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, April 2001.

[2] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, August 1994.

[3] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, May / June 2006.

[4] Simon Blake-Wilson, Magnus Nystroem, David Hopwood, Jan Mikkelsen, and Tim Wright. Transport Layer Security (TLS) extensions, June 2003. RFC 3546.

[5] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, August 1998.

[6] 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.

[7] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161. Springer, August 2002. http://eprint.iacr.org/2002/120/.

[8] Tim Dierks and Christopher Allen. The TLS protocol version 1.0, January 1999. RFC 2246.

[9] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) protocol version 1.1, April 2006. RFC 4346.

[10] Tim Dierks and Eric Rescorla. The Transport Layer Security (TLS) protocol version 1.2, August 2008. RFC 5246.

[11] Stephen Farrell. Why didn't we spot that? *IEEE Internet Computing*, 14(1):84–87, Jan.–Feb. 2010.

[12] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The Secure Sockets Layer (SSL) protocol version 3.0, August 2011. RFC 6101; republication of original SSL 3.0 specification by Netscape of November 18, 1996.

[13] Sebastian Gajek, Mark Manulis, Olivier Pereira, Ahmad-Reza Sadeghi, and Jörg Schwenk. Universally composable security analysis of TLS. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *Second International Conference on Provable Security (ProvSec) 2008*, volume 5324 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2008. Full version available as http://eprint.iacr.org/2008/251.

[14] Rati Gelashvili. Attacks on re-keying and renegotiation in key exchange protocols, April 2012. Bachelor's thesis, ETH Zurich.

[15] Vipul Gupta, Douglas Stebila, Stephen Fung, Sheueling Chang Shantz, Nils Gura, and Hans Eberle. Speeding up secure web transactions using elliptic curve cryptography. In *ISOC Network and Distributed System Security Symposium – NDSS 2004*. The Internet Society, February 2004.

[16] 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*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, August 2012.

[17] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model (full version). Cryptology ePrint Archive, Report 2011/219, last revised 2013. http://eprint.iacr.org/2011/219.

[18] Jakob Jonsson and Burton S. Kaliski Jr. On the security of RSA encryption in TLS. In Moti Yung, editor, *Advances in Cryptology – Proc. CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 183–199. Springer, 2002.

[19] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 310–331. Springer, August 2001.

[20] Hugo Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 546–566. Springer, August 2005.

[21] 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.

[22] Paul Morrissey, Nigel P. Smart, and B. Warinschi. A modular security analysis of the TLS handshake protocol. In Josef Pieprzyk, editor, *Advances in Cryptology – Proc. ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2008. Full version available as http://eprint.iacr.org/2008/236.

[23] Kenneth G. Paterson, Thomas Ristenpart, and Thomas Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 372–389. Springer, December 2011.

[24] Marsh Ray and Steve Dispensa. Renegotiating TLS, November 2009. http://extendedsubset.com/Renegotiating_TLS.pdf.

[25] Eric Rescorla. HTTP over TLS, May 2000. RFC 2818.

[26] Eric Rescorla, Marsh Ray, Steve Dispensa, and Nasko Oskov. Transport Layer Security (TLS) renegotiation indication extension, February 2010. RFC 5746.

[27] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, Nov 2004.

[28] Thierry Zoller. TLS & SSLv3 renegotiation vulnerability. Technical report, G-SEC, 2009. http://www.g-sec.lu/practicaltls.pdf.

A Additional Definitions

A.1 Matching Conversations

Bellare and Rogaway [2] introduced the notion of *matching conversations* to help define correctness and security of an AKE protocol. Later Jager *et al.* [16, 17] modified the original definition to take into account which party sent the last message. The asymmetry of the definition — the fact that we have to distinguish which party has sent the last message — is necessary since protocol messages may be sent sequentially. For instance, in the TLS handshake protocol (see Figure 5) the last message the client sends is the 'client finished' message fin_C, and then it waits for the 'server finished' message fin_S before acceptance. The server, however, 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. Since an active adversary may simply drop the last protocol message, this must be considered in the definition of matching conversations.

Definition 6 (Matching conversations)**.** *Let* transcripts T_A *and* T_B *be two sequences of messages sent and received, in chronological order, by parties* P_A *and* P_B *respectively. We say that* T_A *is a* prefix *of* T_B*, if* T_A *contains at least one message, and the messages in* T_A *are identical to and in the same order as the first* $|T_A|$ *messages of* T_B*. We say that transcript* T_A *is a* matching conversation *to* T_B *if (i)* T_B *is a prefix of* T_A *and party* P_A *has sent the last message(s), or (ii)* $T_A = T_B$ *and party* P_B *has sent the last message(s).*

We say that T_A *and* T_B *are matching conversations, if* T_A *is a* matching conversation *to* T_B *and vice versa. Likewise we say that* P_A *and* P_B have *matching conversations, if* T_A *and* T_B are *matching conversations.*

A.2 Stateful Length-Hiding Authenticated Encryption (sLHAE)

Length-hiding authenticated encryption (LHAE) was originally introduced by Paterson *et al.* [23]. Here we describe the variant stateful LHAE (sLHAE), stated by Jager *et al.* [16].

A *stateful symmetric encryption scheme* is a pair of algorithms $\mathsf{StE} = (\mathsf{StE.Enc}, \mathsf{StE.Dec})$. Algorithm $(C, st'_e) \xleftarrow{\$} \mathsf{StE.Enc}(k, len, head, m, st_e)$ takes as input a secret key $k \in \{0,1\}^\kappa$, an output ciphertext length

$len \in \mathbb{N}$, some header data $head \in \{0,1\}^*$, a plaintext $m \in \{0,1\}^*$, and the current state $st_e \in \{0,1\}^*$, and outputs a ciphertext $C \in \{0,1\}^*$ and an updated state st'_e. Algorithm $(m', st'_d) = \mathsf{StE.Dec}(k, head, C, st_d)$ takes as input a key k, header data $head$, a ciphertext C, and the current state $st_d \in \{0,1\}^*$, and returns an 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. Both encryption state st_e and decryption state st_d are initialized to the empty string $\emptyset$. Algorithm $\mathsf{StE.Enc}$ may be probabilistic, while $\mathsf{StE.Dec}$ is always deterministic.

Definition 7. *A stateful symmetric encryption scheme* $\mathsf{StE} = (\mathsf{StE.Enc}, \mathsf{StE.Dec})$ *is* $(\tau, \epsilon_{\mathsf{sLHAE}})$*-secure, if* $\Pr[b = b'] \leq \epsilon_{\mathsf{sLHAE}}$ *for all adversaries* $\mathcal{A}$ *running in time at most* τ *in the following experiment:*

1. *Choose* $b \xleftarrow{\$} \{0,1\}$ *and* $k \xleftarrow{\$} \{0,1\}^\kappa$*, and set* $st_e \leftarrow \emptyset$ *and* $st_d \leftarrow \emptyset$*.*
2. *Run* $b' \xleftarrow{\$} \mathcal{A}^{\mathsf{Encrypt},\mathsf{Decrypt}}$*.*

Oracle $\mathsf{Encrypt}(m_0, m_1, len, head)$ *takes as input two messages* m_0 *and* m_1 *and a length* len*, and keeps a counter* i *which is initialized to* 0*. Oracle* $\mathsf{Decrypt}(C, head)$ *takes as input a ciphertext* C *and header* $head$*, and keeps a counter* j *and a switch* $\mathsf{diverge}$*, both initialized to* 0*. Both oracles proceed as defined in Figure 4.*

This behaviour of the $\mathsf{Decrypt}$ oracle in this combined definition for confidentiality and integrity can be somewhat difficult to understand, so we give a brief explanation. From a high level, when $b = 0$, the adversary always receives $\perp$ from $\mathsf{Decrypt}$ queries. When $b = 1$, if the adversary successfully manipulates the integrity of the scheme — either by injecting a new valid ciphertext or messing around with the order of ciphertexts but still getting them to successfully decrypt — then $\mathsf{Decrypt}$ outputs the message m which is not equal to $\perp$, so the adversary learns that $b = 1$ not 0.

Note that $\mathsf{diverge}$ is stateful, so if it is ever set to 1 it remains 1. If the adversary ever calls $\mathsf{Decrypt}$ with "not just the next ciphertext sent by the other party", then $\mathsf{diverge}$ is set. In other words, if the adversary tries to inject a new ciphertext, or mess around with the order in which ciphertexts are delivered, $\mathsf{diverge}$ is set. In an ideal stateful encryption scheme, decryption should now fail. But if decryption succeeds, the adversary receives something other than $\perp$ (when $b = 1$), allowing it to learn the challenge bit b, and thus the scheme should be considered insecure.

$\underline{\mathsf{Encrypt}(m_0, m_1, len, head)}$:	$\underline{\mathsf{Decrypt}(C, head)}$:
1. $i \leftarrow i + 1$	1. $j \leftarrow j + 1$
2. $(C^{(0)}, st_e^{(0)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, len, head, m_0, st_e)$	2. $(m, st_d) = \mathsf{StE.Dec}(k, head, C, st_d)$
3. $(C^{(1)}, st_e^{(1)}) \xleftarrow{\$} \mathsf{StE.Enc}(k, len, head, m_1, st_e)$	3. If $(j > i)$ OR $(C \neq C_j)$, then $\mathsf{diverge} \leftarrow 1$
4. If $(C^{(0)} = \perp)$ OR $(C^{(1)} = \perp)$, then return $\perp$	4. If $(b = 1)$ AND $(\mathsf{diverge} = 1)$, then return m
5. $(C_i, st_e) \leftarrow (C^{(b)}, st_e^{(b)})$	5. Else return $\perp$
6. Return C_i	

Figure 4: $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ oracles in the stateful LHAE security experiment.

A.3 The PRF-Oracle-Diffie-Hellman Assumption

The PRF-Oracle-Diffie-Hellman (PRFODH) assumption presented by Jager *et al.* [16] is a variant of the ODH assumption introduced by Abdalla, Bellare and Rogaway [1], adapted from hash functions to PRFs. Jager *et al.* allow a single oracle query, in contrast to a polynomial number of queries as in the original assumption [1]. Abdalla *et al.* point out that the ODH assumption is heuristically reasonable: in the random oracle model, the strong DH assumption implies the ODH assumption.

Let G be a group with generator g. Let PRF be a deterministic function $\mathsf{PRF} : G \times \{0,1\}^* \to \{0,1\}^\mu$. 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 chooses $u, v \in [1, q]$, sets $z_0 = \mathsf{PRF}(g^{uv}, m)$ and samples $z_1 \in \{0,1\}^\mu$ uniformly at random. Then it tosses a coin $b \in \{0,1\}$ and returns (z_b, g^u, 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 8. *We say that the* PRFODH *problem is* $(\tau, \epsilon_{\mathsf{prfodh}})$-hard *with respect to* G *and* PRF*, if for all adversaries* $\mathcal{A}$ *that run in time* τ *it holds that* $|\Pr[b = b'] - 1/2| \leq \epsilon_{\mathsf{prfodh}}$.

B Protocols without Forward Security

The ACCE notion of Jager *et al.* [16] requires confidentiality/integrity of ciphertexts to hold even when the long-term secret keys are corrupted after the handshake completes. As a result, RSA key transport-based ciphersuites cannot be proven secure in this model: they do not have forward security, since corrupting the long-term secret keys allows the adversary to compute the session key.

It is plausible that a non-forward-secure notion of ACCE could be defined in which an RSA key transport-based ciphersuite could be proven secure. While that is beyond the scope of this document, we do comment on how the notions we have introduced in this paper may be modified to consider protocols without forward security.

B.1 Model

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 Def. 5 to consider protocols without forward security:

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

A8. $\mathcal{A}$ did not query $\mathsf{Corrupt}(P_B, \pi_A^s.\mathtt{phases}[\ell].pk')$ before π_A^s accepted in phase $\ell + 1$, for every $\ell < \ell^*$, where $\pi_A^s.d = \pi_B^t$.

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 Def. 2 to consider protocols without forward security:

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

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

B.2 On Renegotiation Security of TLS_RSA_ with SCSV/RIE

As mentioned in Section 2, TLS ciphersuites without forward security, such as RSA key transport, cannot be proven to be secure ACCE protocols, since the compromise of a party's long-term secret key after the phase accepts allows the adversary to compute the master secret. Hence RSA key transport-based ciphersuites can also not be shown to be secure or weakly secure renegotiable ACCE protocols. However, it is plausible that the ACCE definition can be modified to consider protocols without forward security, 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 5, also satisfy a non-forward-secure notion of renegotiation security.

C TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA Protocol with Renegotiation Extensions

Figure 5: TLS handshake for TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA ciphersuite with client authentication †and SCSV / RIE renegotiation countermeasures

D Generic TLS Protocol with Renegotiation Extensions

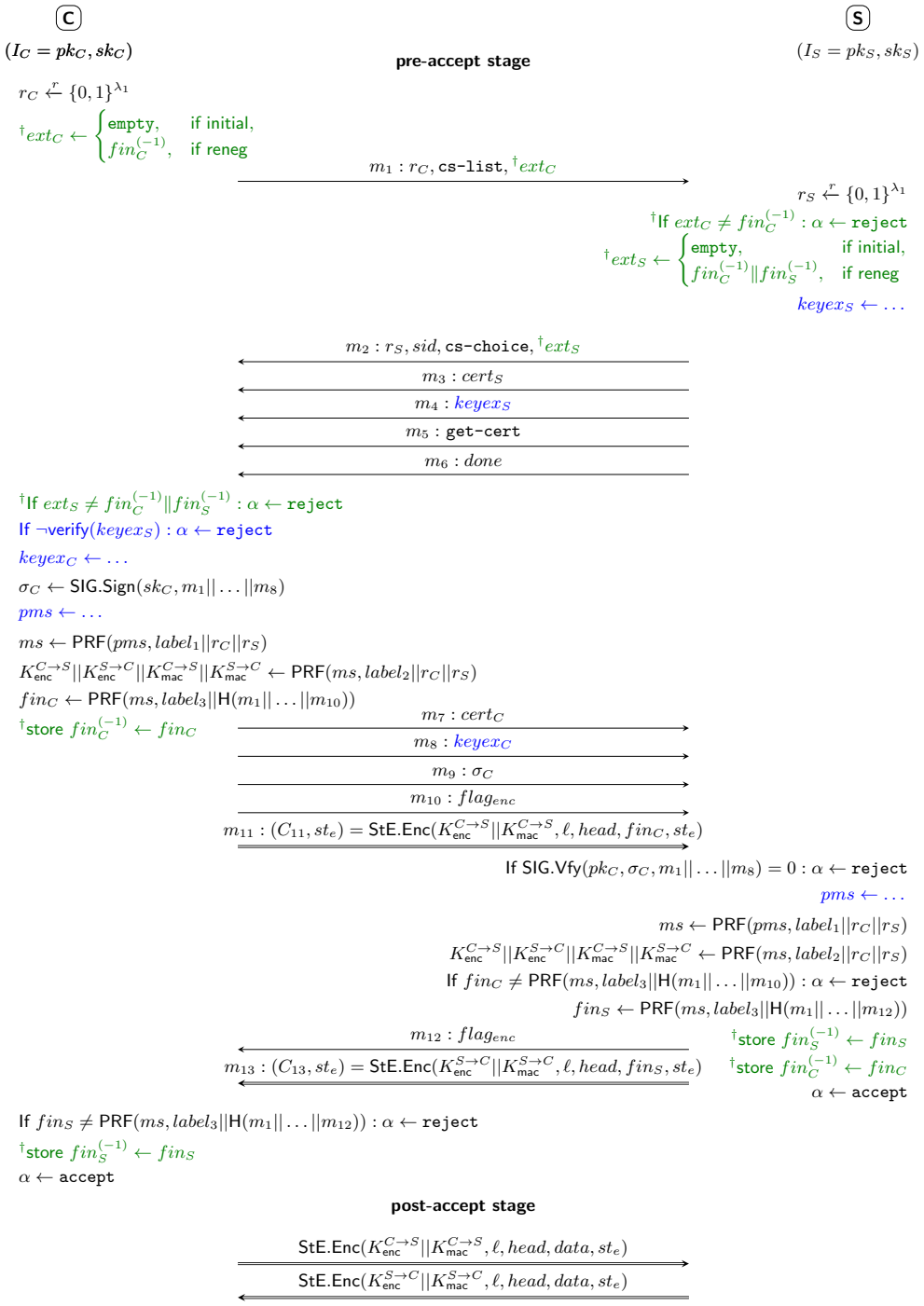

Figure 6: Generic TLS handshake protocol with †SCSV / RIE renegotiation countermeasures