Safely Exporting Keys from Secure Channels:
On the Security of EAP-TLS and TLS Key Exporters

Christina Brzuska[1], Håkon Jacobsen[2,⋆], and Douglas Stebila[3,4,⋆⋆]

[1] Hamburg University of Technology, Hamburg, Germany
brzuska@tuhh.de
[2] Norwegian University of Science and Technology, Trondheim, Norway
hakoja@item.ntnu.no
[3] Queensland University of Technology, Brisbane, Australia
[4] McMaster University, Hamilton, Ontario, Canada
douglas@stebila.ca

Abstract. We investigate how to safely export additional cryptographic keys from secure channel protocols, modelled with the *authenticated and confidential channel establishment (ACCE)* security notion. For example, the EAP-TLS protocol uses the Transport Layer Security (TLS) handshake to output an additional shared secret which can be used for purposes outside of TLS, and the RFC 5705 standard specifies a general mechanism for exporting keying material from TLS. We show that, for a class of ACCE protocols we call "TLS-like" protocols, the EAP-TLS transformation can be used to export an additional key, and that the result is a secure AKE protocol in the Bellare–Rogaway model. Interestingly, we are able to carry out the proof without looking at the specifics of the TLS protocol itself (beyond the notion that it is "TLS-like"), but rather are able to use the ACCE property in a semi black-box way. To facilitate our modular proof, we develop a novel technique, notably an encryption-based key checking mechanism that is used by the security reduction. Our results imply that EAP-TLS using secure TLS 1.2 ciphersuites is a secure authenticated key exchange protocol.

1 Introduction

Secure channel protocols are widely used in practice to allow two parties to authenticate each other and securely transmit data. A common design paradigm is to use an *authenticated key exchange (AKE)* protocol to authenticate parties based on public key certificates and to establish a session key, and then use a *stateful authenticated encryption* scheme to encrypt and authenticate the transmission of application data. Real-world secure channel protocols such as TLS, SSH, IPsec, Google's QUIC, the EMV chip-and-pin system, and IEEE 802.11i all follow this paradigm.

⋆ Håkon Jacobsen was supported by a STSM Grant from COST Action IC1306.
⋆⋆ Douglas Stebila was supported by Australian Research Council (ARC) Discovery Project grant DP130104304.

For theoreticians, this paradigm is desirable because it allows for modular proofs via composability. A classic result by Canetti and Krawczyk [11] shows how to provably construct a secure channel by running a key exchange protocol that satisfies standard key indistinguishability notions, and then using the key output by the AKE protocol as the symmetric key in authenticated encryption.

For practitioners, this paradigm is desirable because it is efficient and allows to use and combine simple software and hardware components in a variety of ways to form the overall system.

Despite the merits of modularity, most real-world designs are not as clean. In TLS versions up to 1.2, a key exchange protocol, the so-called handshake protocol, is used to establish a premaster secret, which is then used to derive a master secret, which is then used to derive session keys. The final messages of the handshake protocol are encrypted using the session keys, and then application data can be sent, encrypted using the same session keys. SSH has a similar design. In this design, the session keys do *not* satisfy the standard key indistinguishability notion for key exchange security: an adversary can decide whether they have been given the real session key or a random one simply by trial decrypting the encrypted handshake messages.

Early work on proving the security of TLS avoided this problem by showing that a modified version of the TLS handshake yields indistinguishable session keys [29], but this is unsatisfactory since it does not consider the TLS protocol as used in practice. In 2012, Jager, Kohlar, Schäge, and Schwenk (JKSS) [20] introduced the *authenticated and confidential channel establishment* (ACCE) security notion, which treats the key exchange and authenticated encryption as a single monolithic object, allowing them to prove security of the signed Diffie–Hellman ciphersuites in the unmodified TLS 1.2 protocol. ACCE has been applied or adapted to prove security of most other TLS ciphersuites [21,24,26], as well as SSH [4], QUIC [27,15], and the EMV chip-and-pin system [10].

The ACCE notion is not necessarily ideal to cryptographers; its monolithic nature can make modular analysis more difficult, and in particular individual components of ACCE-secure protocols cannot necessarily be used independently. For example, although TLS 1.2's signed Diffie–Hellman ciphersuite is ACCE-secure, one has no security assurance that the session key satisfies any independent security notion: we only have the assurance that the session key is safe to use with the corresponding authenticated encryption scheme in the manner described by the protocol.

Moreover, practitioners seem to like to use the TLS handshake in order to establish keying material for their own purposes. A prominent example is the EAP-TLS protocol [33], which uses the TLS handshake to derive a session key between two peers in the Extensible Authentication Protocol (EAP) [1]. More generally, the practice of *exporting* additional keys from the master secret in the TLS handshake has been formalized in the proposed IETF standard RFC 5705 on TLS key material exporters [31].

However, is it actually safe to use keys exported from the master secret in the TLS handshake? Solely assuming ACCE security of TLS does not at first

sight seem to say anything about the *internal* variables of TLS, such as the master secret. However, interestingly, inspired by Morrissey, Smart, and Warinschi (MSW) [29] we can show that the ACCE security of TLS implies that the master secret is *unpredictable*. If the master secret were predictable, then we would be able to break the security of the ACCE channel. This intuition lies at the heart of our proof which uses the ACCE property of TLS in a (semi-)black-box way.

Our contributions. In this paper we analyze the security of key exporters from ACCE protocols in the provable security setting. Concretely, for TLS we show that if one derives an additional exported key from the TLS master secret—independently of the other handshake messages—then TLS (outputting this additional exported key as the session key) constitutes a secure AKE protocol in the sense of Bellare and Rogaway [2]. However, while our starting point is the TLS protocol, our result is in fact more general, pertaining to a wider class of protocols which we call *TLS-like ACCE protocols*. Roughly speaking, these are protocols which satisfy the ACCE security notion and, like TLS, establish a master secret during the handshake, and from the master secret derive both the channel encryption key and the additional exported key. Apart from this requirement, our result has no other dependencies on the specifics of the protocol. In other words, our main result is a general theorem showing that the transformation specified by EAP-TLS as a key exporter turns any ACCE protocol which has a concept of a master secret into an AKE protocol.

An immediate application of our result is a proof of security in the Bellare-Rogaway model for TLS Key Material Exporters [31] and EAP-TLS [33]. The former has never been subject to a formal security analysis, while the latter has only been analyzed in the symbolic model by He et al. [17] who gave a proof in the context of IEEE 802.11.

Motivation for our approach. MSW [29] proved that a modified version of the TLS handshake yields indistinguishable session keys. Specifically, they considered a variant of TLS were the final messages are sent unencrypted. As an intermediate step in their analysis, they showed that the TLS master secret is unpredictable, i.e., that no adversary is able to output the full master secret of a fresh target session. They modeled the key derivation function (KDF) in TLS as a random oracle, and as the inputs to the random oracle are unpredictable, the session keys derived from the master secret are indistinguishable from random.

Similar to MSW, we want to use the fact that the master secret is unpredictable to show that *export* keys are indistinguishable from random. This should be possible even for the *unmodified* TLS protocol, because exported keys are not used to encrypt messages during the handshake phase. One obvious approach would be to reuse one of the existing security proofs which shows TLS to be ACCE secure. Specifically, in these proofs the master secret of a particular session is typically swapped out with a completely random value, allowing the rest of the proof to continue on the assumption that the master secret is completely hidden from the adversary. Due to the unpredictability of the master secret, the adversary will not be able to detect the switch. Using this truly random master

secret, we could extend the proof with one additional step where we derive the export key through a random oracle query. It would then follow that the derived export key is indistinguishable from random.

However, such a result could not be re-used across different TLS ciphersuites, nor hold for future versions of TLS. Instead, for every variant of TLS, one would have to copy-paste the corresponding security proof and augment it accordingly to account for the extra export key. This approach is of course inherently non-modular since it is tied to the innards of each particular proof. Still, it seems likely that most of these proofs would be fairly similar in terms of technique, and also reasonably independent of the specific details of the TLS protocol itself.

The question is whether we can isolate exactly those properties of the TLS protocol that these proofs rely on. If so, we could extract a generic proof of TLS key exporters that works across different versions unmodified. Moreover, it would be even better if we could have a result that is not tied to TLS at all, but rather one that targets an appropriate abstract security notion.

Essentially, this is what we do in this paper. We identify some features of the TLS protocol which, when added to a generic ACCE protocol, are sufficient to establish the indistinguishability of the export keys derived from the protocol. Note that, apart from the features that we identify, the result is completely independent of the internals of TLS. Below we describe these features.

Technical overview of our result. Surprisingly, the number of additional features we require in addition to a generic ACCE protocol is rather minimal and consists of the following three requirements (which we make more precise in Sect. 3). We call an ACCE protocol that satisfies these requirements *TLS-like*.

(i) The handshake includes a random *nonce* from each party.
(ii) Each party maintains a value called the *master secret* during the handshake.
(iii) The session key is derived from the master secret, the nonces, and possibly some other public information.

Our result can now be more precisely formulated as follows: starting from an ACCE secure TLS-like protocol Π, we create an AKE secure protocol Π^+, where Π^+ consists of first running protocol Π until a session accepts (according to Π), then deriving one additional key from the master secret and nonces of Π. This key—which is distinct from the session key in the underlying protocol Π—becomes the session key of Π^+. In our security proof the key derivation step will be modeled using a random oracle. The construction of Π^+ from Π precisely captures the definition used in TLS key exporters [31] and EAP-TLS [33].

Note that while we put no security requirements on the master secret of a TLS-like protocol, it is pivotal in our proof to relate the indistinguishability of the session keys in Π^+ to the ACCE security of Π. As mentioned previously, we build on the idea used by MSW [29] to show that unless the adversary queries the random oracle on the exact master secret of a party, it has no advantage in distinguishing the corresponding exported session key in Π^+. MSW proved that an application key agreement protocol (having indistinguishable session keys) could be built out of a master key agreement protocol (having unpredictable master

secrets). In their security reduction the simulator could simulate the application key agreement protocol since it had access to a *perfect* key-checking oracle, allowing it to test the validity of master secrets supplied to the random oracle. Our proof is complicated by the fact that we do a reduction to a (TLS-like) ACCE protocol for which there is no key-checking oracle available. The main technical novelty of our proof is to show that we can still create an approximation of the key-checking oracle as long as we allow a (small) one-sided error probability. This emulated key-checking oracle suffices to simulate the AKE experiment of protocol Π^+ in our reduction to the ACCE security of Π.

To give some intuition for our key-checking oracle within the ACCE setting, suppose we want to test whether the value ms' is the master secret of some session π. First, we use ms', the nonces π accepted with, and the KDF of Π (all available due to the TLS-like requirement on Π) to derive a *guess* on π's session key *in* Π. Next, we obtain a ciphertext C of a random message under π's *actual* session key in Π, using our access to a left-or-right encryption oracle in the ACCE game. Finally, we *locally* decrypt C using the guessed session key of Π, i.e., we do not use the decrypt oracle of the ACCE game. If this decryption gives back the random message we started with, we guess that ms' was the correct master secret of π; otherwise, we guess that it was incorrect.

In the above we tacitly assumed that different master secrets derive different session keys (using the same nonces). Normally, this would follow directly from the pseudorandomness of the KDF used in Π. However, since we do not require the master secrets to be independent and uniformly distributed, we cannot invoke this property of the KDF. Instead, we have to explicitly assume that different master secrets do not "collide" to the same session key. We expect this property to hold for most real-world KDFs. Concretely, we show in Theorem 2 (Appx. A) that the HMAC-based KDF used in TLS has this property.

Alternatives to using the ACCE security notion? The main reason for using the ACCE security notion in our analysis is that is has proved to be a very useful model for studying the security of two-stage channel establishment protocols. As already mentioned, it has been used repeatedly to analyze real-world protocols such as TLS, SSH, and QUIC. Since our result applies to *any* ACCE protocol that is TLS-like, it can be applied to all these protocols in a near black-box manner. In particular, we can plug in any existing ACCE result without having to re-do any of the steps carried out in the (ACCE) proof of the protocol itself. For example, our result applies unmodified to every ciphersuite version of TLS for which there exist an ACCE proof. Moreover, we can even apply our theorem to future versions of TLS, as long as these continue to be TLS-like and derive their channel keys using a collision resistant KDF.

Still, in the specific case of TLS, one might ask whether another approach could give a simpler, yet equally modular proof of the same result, namely that EAP-TLS (and more generally, TLS key exporters) constitutes a secure AKE.

Krawczyk, Paterson, and Wee (KPW) [24] showed that all the major handshake variants of TLS satisfy a security notion on its key encapsulation mechanism (KEM) called IND-CCCA [18]. If we could reduce the AKE security of

EAP-TLS to the IND-CCCA security of the TLS-KEM, then the results of KPW would give us all the major TLS ciphersuites "for free".

Unfortunately, it is not obvious how such a result can be obtained in a black-box manner from the KEM in [24]. Technically, in order to reduce the AKE security of EAP-TLS to the IND-CCCA security of the TLS-KEM, we need to be able to simulate the key derivation step in the AKE game of EAP-TLS. This requires knowledge about the sessions' master secrets. However, the KEM defined by KPW does not contain the TLS master secret. This means that an adversary against the TLS-KEM in the IND-CCCA game cannot simulate the Test-challenge for some adversary playing in the AKE game against EAP-TLS. Moreover, as remarked by KPW [24, Remark 4], if the KEM key actually *was* defined to be the TLS master secret, then the resulting scheme would be insecure for TLS-RSA provided that RSA PKCS#1v1.5 is re-randomizable[5].

Other modular approaches to analyzing TLS. Canetti and Krawczyk [11] presented a model that allows to analyze protocols in modular way. Unfortunately, since TLS does not meet the stringent requirement of key indistinguishability, it cannot be analyzed within their framework. Küsters and Thuengerthal [25] analyzed the core of TLS in their simulation-based universal composability model called IITM. Unlike some other UC models, the IITM model has the appealing feature that it does not rely on pre-established session identifiers. Brzuska et al. [8] introduced a framework that uses so-called key-independent reductions and allows to analyze protocols such as TLS. Their analysis is in a game-based setting and, up to some small technical differences between models, implies ACCE security of TLS. Kohlweiss et al. [22] recently used the abstract cryptography framework by Maurer and Renner [28] for a modular analysis of TLS.

2 Protocol Definitions

2.1 Execution Environment

Parties. A two-party protocol is carried out by a set of parties $\mathcal{P} = \{P_1, \ldots, P_{n_\mathcal{P}}\}$. Each party P_i has an associated long-term key pair (sk_i, pk_i). We presuppose the existence of a public key infrastructure (PKI) by assuming that every party has an authenticated copy of all the other parties' public keys pk_i. For simplicity we restrict to the setting of mutual authentication, but our results apply equally to the server-only authenticated setting.

Sessions. Each party can take part in multiple executions of the protocol, both concurrently and subsequently. Each run of the protocol is called a *session*. Let n_π denote the maximum number of sessions per party; for party P_i's sth session, we associate an oracle π_i^s which embodies this (local) session's execution of the

[5] On the other hand, Bhargavan et al. [6] conjecture that re-randomizing RSA PKCS#1v1.5 is infeasible, allowing the master secret to be used as the KEM key in TLS-RSA too. We forgo the issue by not reducing to the KEM-security of TLS.

Table 1. State variables for session oracle π_i^s.

Variable	Description
ρ	the role $\rho \in \{\mathsf{init}, \mathsf{resp}\}$ of the session in the protocol execution, being either the *initiator* or the *responder*
pid	the identity $\mathsf{pid} \in \mathcal{P}$ of the intended communication partner of π_i^s
pk	the public key of $\pi_i^s.\mathsf{pid}$
α	the state $\alpha \in \{\texttt{accepted}, \texttt{rejected}, \texttt{running}\}$ of the session oracle
T	the ordered transcript of all messages sent and received by π_i^s
k	the symmetric session-key $k \in \mathcal{K}$ derived by π_i^s
γ	the status $\gamma \in \{\perp, \texttt{revealed}\}$ of the session key $\pi_i^s.k$
sid	a session identifier $\mathsf{sid} \in \{0,1\}^*$ locally computable by π_i^s
b	a random bit $b \in \{0,1\}$ sampled at the initialization of π_i^s
st	additional auxiliary state that might be needed by the protocol

protocol, maintains the state specific to this session (as described in Table 1), and has access to the long-term secret key sk_i of the party. We put the following correctness requirements on the variables α, k, sid and pid:

$$\pi_i^s.\alpha = \texttt{accepted} \implies \pi_i^s.k \neq \perp \wedge \pi_i^s.\mathsf{sid} \neq \perp, \tag{1}$$

$$\pi_i^s.\alpha = \pi_j^t.\alpha = \texttt{accepted} \wedge \pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid} \implies \begin{cases} \pi_i^s.k = \pi_j^t.k \\ \pi_i^s.\mathsf{pid} = P_j \\ \pi_j^t.\mathsf{pid} = P_i \end{cases}. \tag{2}$$

Adverserial queries. The adversary is assumed to control the network, and interacts with the oracles by issuing queries to them. Below we describe the admissible queries.

- $\mathsf{NewSession}(P_i, \rho, \mathsf{pid})$: This query creates a new session π_i^s with at party P_i, having role ρ and intended partner pid. Based on pid, π_i^s sets the variable pk correspondingly. The session's state is set to $\pi_i^s.\alpha = \texttt{running}$ and, if $\rho = \mathsf{init}$, it also produces the first message of the protocol which is returned to the adversary.
- $\mathsf{Send}(\pi_i^s, m)$: This query allows the adversary to send any message m to the session oracle π_i^s. If $\pi_i^s.\alpha \neq \texttt{running}$ return $\perp$. Otherwise, the oracle responds according to the protocol specification, which depends on its role and current internal state.
- $\mathsf{Corrupt}(P_i)$: Return the private key $P_i.sk$ held by party P_i. If $\mathsf{Corrupt}(P_i)$ was the τ-th query issued by $\mathcal{A}$, then we say that P_i is τ*-corrupted*. For uncorrupted parties we define $\tau := \infty$.
- $\mathsf{Reveal}(\pi_i^s)$: This query returns the session key $\pi_i^s.k$ and sets $\pi_i^s.\gamma = \texttt{revealed}$.

2.2 AKE Protocols

An *authenticated key exchange protocol* (AKE) is a two-party protocol satisfying the syntactical requirement of (1) and (2), and where the security is defined in

terms of an AKE security experiment played between a challenger and an adversary. This experiment uses the execution environment described in Section 2.1, but has one additional query:

– $\mathsf{Test}(\pi_i^s)$: This query may be asked only once during the course of the game. If $\pi_i^s.\alpha \neq \texttt{accepted}$, then the oracle returns $\perp$. Otherwise, based on $b = \pi_i^s.b$, it returns k_b, where $k_0 \leftarrow \mathcal{K}$ is an independent uniformly sampled key and $k_1 := \pi_i^s.k$. The key k_b is called the Test-*challenge*.

The adversary can win in the AKE experiment in one of two ways: (i) by making a session accept maliciously or (ii) by guessing the secret bit of the Test-session. We formalize these winning conditions below. We simultaneously consider AKE protocols with and without *perfect forward secrecy* (PFS) [13].

Definition 1. *Two sessions π_i^s and π_j^t are* partners *if $\pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid}$.*

Definition 2. *A session π_i^s is said to be* fresh *(resp.* PFS-fresh*), with intended partner P_j, if*

(a) $\pi_i^s.\alpha = \texttt{accepted}$ and $\pi_i^s.\mathsf{pid} = P_j$ when $\mathcal{A}$ issued its τ_0-th query,
(b) $\pi_i^s.\gamma \neq \texttt{revealed}$ and P_i is uncorrupted (resp. τ-corrupted with $\tau_0 < \tau$)[6],
(c) for any partner oracle π_j^t of π_i^s, we have that $\pi_j^t.\gamma \neq \texttt{revealed}$ and P_j is uncorrupted (resp. τ'-corrupted with $\tau_0 < \tau'$).

Definition 3 (Entity authentication). *A session π_i^s is said to have* accepted maliciously *(resp.* accepted maliciously with PFS*) in the AKE security experiment with intended partner P_j, if*

(a) $\pi_i^s.\alpha = \texttt{accepted}$ and $\pi_i^s.\mathsf{pid} = P_j$ when $\mathcal{A}$ issued its τ_0-th query,
(b) P_i and P_j are uncorrupted (resp. τ- and τ'-corrupted with $\tau_0 < \tau, \tau'$), and
(c) there is no unique session π_j^t such that π_i^s and π_j^t are partners.

We let $\mathbf{Adv}_{\Pi}^{\mathsf{auth}}(\mathcal{A})$ (resp. $\mathbf{Adv}_{\Pi}^{\mathsf{auth\text{-}PFS}}(\mathcal{A})$) denote the probability that an adversary $\mathcal{A}$ gets a session to accept maliciously (resp. accepts maliciously with PFS) during the AKE security experiment.

Definition 4 (Key indistinguishability). *An adversary $\mathcal{A}$ that issued its Test-query to session π_i^s during the AKE security experiment,* answers the Test-challenge correctly *(resp.* answers the Test-challenge correctly with PFS*) if it terminates with output b', such that*

(a) π_i^s is fresh (resp. PFS-fresh) with some intended partner P_j, and
(b) $\pi_i^s.b = b'$.

We assign the following advantage measure to the event that $\mathcal{A}$ answers the Test-challenge correctly (resp. answers the Test-challenge correctly with PFS):

$$\mathbf{Adv}_{\Pi}^{\mathsf{key\text{-}ind(\text{-}PFS)}}(\mathcal{A}) := |\Pr[\pi_i^s.b = b'] - \frac{1}{2}| \ . \tag{3}$$

[6] For simplicity we do not model *key-compromise impersonation attacks* in this paper, which should allow P_i itself to be τ-corrupted, with $\tau < \tau_0$.

Definition 5 (AKE security). *An adversary $\mathcal{A}$* wins *(resp.* wins with PFS*) in the AKE security experiment if a session to accept maliciously (resp. accept maliciously with PFS) or it answers the* Test*-challenge correctly (resp. answers the* Test*-challenge correctly with PFS). We assign the following advantage measure to the event that $\mathcal{A}$ wins (resp. wins with PFS):*

$$\mathbf{Adv}_{\Pi}^{\mathsf{AKE(\text{-}PFS)}}(\mathcal{A}) := \mathbf{Adv}_{\Pi}^{\mathsf{auth(\text{-}PFS)}}(\mathcal{A}) + \mathbf{Adv}_{\Pi}^{\mathsf{key\text{-}ind(\text{-}PFS)}}(\mathcal{A}) \ . \tag{4}$$

2.3 ACCE Protocols

Jager et al. [20] introduced the notion of *authenticated and confidential channel establishment (ACCE)* protocols in order to model TLS. An ACCE protocol is a two-party protocol satisfying the syntactical requirement of equations (1) and (2) and where the session key k is used to key a *stateful length-hiding authenticated encryption scheme* (sLHAE) $\mathsf{stE} = (\mathsf{st.Gen}, \mathsf{stE.Init}, \mathsf{stE.Enc}, \mathsf{stE.Dec})$ (following the definition in [24]). For correctness, we require that if the *deterministic* algorithm $\mathsf{st.Init}$ produced initial states st_E^0, st_D^0, and the ACCE session key k was used to produce a sequence of encryptions $(C_i, st_E^{i+1}) \leftarrow \mathsf{stE.Enc}(k, \ell, m_i, H_i, st_E^i)$ where no C_i equal $\perp$, then the sequence of decryptions $(m_i', st_D^{i+1}) \leftarrow \mathsf{stE.Dec}(k, C_i, H_i, st_D^i)$ is such that $m_i' = m_i$ for each $i \geq 0$. For security, we define an ACCE security experiment based on the execution environment described in Sect. 2.1 that has the following two additional queries (note that there is no Test query).

- $\mathsf{Encrypt}(\pi_i^s, \ell, m_0, m_1, H)$: This query takes as input a ciphertext length specification ℓ, two messages m_0, m_1, and a header H. If $\pi_i^s.\alpha \neq \texttt{accepted}$, the query returns $\perp$. Otherwise, π_i^s has (by assumption) computed its session key k and run the $\mathsf{stE.Init}$ algorithm of a sLHAE scheme st_{E} to initiate states $\pi_i^s.st_E$ and $\pi_i^s.st_D$. Depending on the bit $\pi_i^s.b$, this call returns the encryption of either m_0 or m_1 using stE. For details, see Fig. 1.
- $\mathsf{Decrypt}(\pi_i^s, C, H)$: This query takes as input a ciphertext C and a header H. If $\pi_i^s.\alpha \neq \texttt{accepted}$, then the query returns $\perp$. Otherwise, it (statefully) decrypts (C, H) using the underlying sLHAE scheme stE. For details, see Fig. 1.

The adversary can win in the ACCE experiment in one of two ways: (i) by making a session accept maliciously according to Def. 3 (as in the AKE security experiment), or (ii) by breaking one of the sLHAE channels through guessing the corresponding session's secret bit, (we formally define this condition below). Partnering and freshness in the ACCE experiment are defined exactly like in the AKE experiment, i.e., according to Def. 1 and Def. 2, respectively.

Definition 6 (Channel security). *An adversary $\mathcal{A}$* breaks the channel *(resp.* breaks the channel with PFS*) in the ACCE security experiment if it terminates with output (π_i^s, b'), such that*

(a) π_i^s is fresh (resp. PFS-fresh) with some intended partner P_j, and
(b) $\pi_i^s.b = b'$.

$\underline{\mathsf{Encrypt}(\pi_i^s, \ell, m_0, m_1, H)}$:
1: **if** $\pi_i^s.\alpha \neq$ `accepted`**:**
2: **return** $\perp$:
3: $u \leftarrow u + 1$;
4: $(C^{(0)}, st_{\mathrm{E}}^{(0)}) \leftarrow \mathsf{stE.Enc}(k, \ell, m_0, H, st_{\mathrm{E}})$;
5: $(C^{(1)}, st_{\mathrm{E}}^{(1)}) \leftarrow \mathsf{stE.Enc}(k, \ell, m_1, H, st_{\mathrm{E}})$;
6: **if** $C^{(0)} = \perp$ or $C^{(1)} = \perp$**:**
7: **return** $\perp$;
8: $(\mathsf{C}[u], \mathsf{H}[u], st_{\mathrm{E}}) := (C^{(b)}, H, st_{\mathrm{E}}^{(b)})$
9: **return** $\mathsf{C}[u]$;

$\underline{\mathsf{Decrypt}(\pi_i^s, C, H)}$:
1: **if** $\pi_i^s.\alpha \neq$ `accepted`**:**
2: **return** $\perp$:
3: **if** $b = 0$**:**
4: **return** $\perp$;
5: $\pi_j^t \leftarrow \pi_i^s$'s partner or $\perp$;
6: $v \leftarrow v + 1$;
7: $(m, st_{\mathrm{D}}) \leftarrow \mathsf{stE.Dec}(k, C, H, st_{\mathrm{D}})$;
8: **if** $v > \pi_j^t.u$ or $C \neq \pi_j^t.\mathsf{C}[v]$ or $H \neq \pi_j^t.\mathsf{H}[v]$**:**
9: `in-sync` $\leftarrow$ `false`;
10: **if** `in-sync` $=$ `false`**:**
11: **return** m;
12: **return** $\perp$;

Fig. 1. The Encrypt and Decrypt queries of the ACCE security experiment. The variables $k, b, st_{\mathrm{E}}, st_{\mathrm{D}}, \mathsf{C}, \mathsf{H}, u$ and v all belong to the internal state of π_i^s. The variables C and H are lists, initially empty. The counters u and v are initialized to 0, and `in-sync` is set to `true` at the beginning of every session π_i^s. In case π_i^s does not have a partner when answering a Decrypt query, then `in-sync` $=$ `false`.

We assign the following advantage measure to the event that $\mathcal{A}$ breaks the channel (resp. breaks the channel with PFS):

$$\mathbf{Adv}_{\Pi}^{\mathsf{chan(\text{-}PFS)}}(\mathcal{A}) := \left| \Pr[\pi_i^s.b = b'] - \frac{1}{2} \right| . \tag{5}$$

Definition 7 (ACCE security). *An adversary $\mathcal{A}$* wins *(resp.* wins with PFS*) in the ACCE security experiment if it either gets a session to accept maliciously (resp. accept maliciously with PFS) or breaks the channel (resp. breaks the challenge with PFS). We assign the following advantage measure to the event that $\mathcal{A}$ wins (resp. wins with PFS) in the ACCE experiment:*

$$\mathbf{Adv}_{\Pi}^{\mathsf{ACCE(\text{-}PFS)}}(\mathcal{A}) := \mathbf{Adv}_{\Pi}^{\mathsf{auth(\text{-}PFS)}}(\mathcal{A}) + \mathbf{Adv}_{\Pi}^{\mathsf{chan(\text{-}PFS)}}(\mathcal{A}) . \tag{6}$$

3 TLS-like Protocols

Definition 8. *An ACCE protocol Π is said to be* TLS-like *if*

(i) each session uniformly at random generates and transmits a distinguished nonce value $n \xleftarrow{\$} \{0,1\}^\lambda$ *during its run of the protocol,*
(ii) each session holds a variable $\pi_i^s.ms \in \{0,1\}^\lambda \cup \{\perp\}$, called the master secret*,*
(iii) if n_1, n_2 are the two nonces on a session's transcript T, then the session key *is derived as*

$$k \leftarrow \mathsf{Kdf}(ms, n_1 \| n_2, F_\Pi(T)), \tag{7}$$

where $\mathsf{Kdf}\colon \{0,1\}^\lambda \times \{0,1\}^{2\lambda} \times \{0,1\}^* \to \{0,1\}^\lambda$ and $F_\Pi\colon \{0,1\}^* \to \{0,1\}^*$ are deterministic functions.

Remark 1. The function F_{Π} is protocol specific and meant to capture any additional input that might be used to derive the session keys. In TLS, $F_{\Pi}(T)$ is the empty string, while for example in IPSec (IKEv2), $F_{\Pi}(T)$ is the Security Parameter Index (SPI) of the initiator and responder.

Remark 2. Clearly TLS is TLS-like, but most other real-world protocols, like SSH, IPSec and QUIC, belong to this class as well.

4 Constructing an AKE protocol from a TLS-like ACCE Protocol

4.1 Construction

Let Π be a TLS-like ACCE protocol with key derivation function Kdf and let $G\colon \{0,1\}^{\lambda} \times \{0,1\}^{2\lambda} \times \{0,1\}^{*} \to \{0,1\}^{\lambda}$ be a random oracle. From Π and G we create an AKE protocol Π^{+} as follows. Protocol Π^{+} consists of first running protocol Π as usual until a session accepts, then it derives an additional key $ek \leftarrow G(ms, n_C \| n_S, aux)$, where ms is the master secret of Π, n_C and n_S are the nonces, and $aux \in \{0,1\}^{*}$ is an (optional) string containing selected values from the session's transcript T. The key ek becomes the *session key* in protocol Π^{+}. The session identifier in Π^{+} is inherited from Π.

By construction, a session in Π^{+} derives (at least) two keys: its "true" session key in the sense of the AKE-model, i.e., the key ek derived from G, and the channel encryption key derived in the underlying protocol Π using $\Pi.\mathsf{Kdf}$. To avoid confusion, we will call the former key the *export key*; while we will call the latter key the *channel key* and denote it ck. In particular, in the formal AKE security experiment the session key variable $\pi_i^s.k$ will store the export key ek, while the channel key ck will simply be part of π_i^s's internal state, written $\pi_i^s.ck$.

4.2 Main Result

Informally, our main result shows that the construction described above transforms a TLS-like ACCE protocol Π into an AKE protocol Π^{+}. However, in our proof we need to rely on two additional assumptions besides the ACCE-notion: (1) the key derivation function $\Pi.\mathsf{Kdf}$ used to derive the channel keys in Π^{+} is *collision resistant* in a particular sense (Def. 9) and (2) the session identifier allows for *public session matching* (Def. 10) and contains the sessions' nonces and $F_{\Pi}(T)$ value (q.v. equation (7)).

Definition 9 (KDF collision resistance). *Let* KDF *be an oracle implementing the key derivation function of a TLS-like ACCE protocol* Π*. Define the following advantage measure for an adversary* $\mathcal{A}$*:*

$$\mathbf{Adv}^{\mathsf{KDFcoll}}_{\Pi.\mathsf{Kdf}}(\mathcal{A}) := \Pr\left[((ms, ms'), n, s) \leftarrow \mathcal{A}^{\mathsf{KDF}} : \begin{array}{c}\mathsf{KDF}(ms, n, s) = \mathsf{KDF}(ms', n, s) \\ ms \neq ms'\end{array}\right]. \tag{8}$$

A triple $((ms, ms'), n, s)$ *satisfying the criteria in* (8) *is called a* (KDF) collision *for* $\Pi.\mathsf{Kdf}$*.*

Remark 3. Definition 9 is a variant of the more common notion of collision resistant *hash functions*. The difference is that KDF collision resistance is about collisions in the *keys*, not the messages.

Definition 10 (Public session matching). *A session identifier* sid *allows for* public session matching *in security experiment E, if there exists an efficient algorithm $\mathcal{M}$—having access to all the queries/responses exchanged between $\mathcal{A}$ and the challenger—that can always answer whether or not two accepted sessions are partners during the execution of E, i.e.:*

$$\forall k \in \mathbb{N}, \text{ and } \forall \pi_i^s, \pi_j^t \text{ having accepted before } \mathcal{A}\text{'s } \tau_k\text{-th query:}$$
$$\mathcal{M}(\pi_i^s, \pi_j^t) := \begin{cases} 1 & \text{if } \pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid}, \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Theorem 1. *Let Π be a TLS-like ACCE protocol having a session identifier that allows for public session matching and contains the sessions' nonces and $F_\Pi(T)$ values. Let Π^+ be the protocol derived from Π and random oracle G, using the construction described in Section 4.1. Then for any adversary $\mathcal{A}$ in the AKE security experiment against Π^+*

$$\mathbf{Adv}^{\mathsf{AKE(\text{-}PFS)}}_{\Pi^+}(\mathcal{A}) \leq 6 \cdot \mathbf{Adv}^{\mathsf{ACCE(\text{-}PFS)}}_{\Pi}(\mathcal{B}) + 3 \cdot \mathbf{Adv}^{\mathsf{KDFcoll}}_{\Pi.\mathsf{Kdf}}(\mathcal{C}) + \frac{6 q n_{\mathcal{P}} n_\pi}{2^{c\lambda}} + \frac{(n_{\mathcal{P}} n_\pi)^2}{2^{\lambda+1}}, \tag{10}$$

where λ is the security parameter, $n_{\mathcal{P}}$ is the number of parties, n_π is the number of sessions at each party, q is $\mathcal{A}$'s number of random oracle queries, and $c \in \mathbb{N}$ is an arbitrary constant.

The main idea behind the proof of Theorem 1 is to relate the security of the derived export keys to the security of the channel keys in the underlying ACCE protocol Π. Roughly speaking, by using the property that TLS-like protocols derive their channel keys from the master secret and nonces, we establish that two sessions derive the same export key if and only if they derive the same channel key (barring certain bad events which we bound). This fact will make it possible to derive the sessions' export keys in Π^+ independently of their master secrets, and still fully simulate the random oracle G.

4.3 Proof of Theorem 1

Let $\mathcal{A}$ be the adversary in an AKE security experiment against protocol Π^+. From $\mathcal{A}$ we construct an algorithm $\mathcal{B}$ against the ACCE security of the underlying protocol Π. Our proof proceeds through a *sequence of games* ([3,32]), where each consecutive game aims to reduce the challenger's dependency on the sessions' master secrets and the random oracle G, in order to derive the export keys in protocol Π^+. Eventually, in the final game, the random oracle G will have been completely replaced by a local list L_G, and the Π^+ export keys are derived independently of the sessions' master secrets. Thus, at this point, algorithm $\mathcal{B}$ will be able to simulate the game.

Game 0. This is the original AKE security experiment for protocol Π^+:

$$\mathbf{Adv}^{\mathsf{AKE(\text{-}PFS)}}_{\Pi^+}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{G}_0}_{\Pi^+}(\mathcal{A}) \ . \tag{11}$$

Game 1. Game 1 proceeds like in Game 0, but aborts if two sessions generate the same nonce value. Since there are $n_{\mathcal{P}} \cdot n_\pi$ generated nonces, the probability of there being at least one collision is bounded by $(n_{\mathcal{P}} n_\pi)^2 \cdot 2^{-(\lambda+1)}$. By the Difference Lemma ([32]) we have

$$\mathbf{Adv}^{\mathsf{G}_0}_{\Pi^+}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{G}_1}_{\Pi^+}(\mathcal{A}) + \frac{(n_{\mathcal{P}} n_\pi)^2}{2^{\lambda+1}}. \tag{12}$$

The remaining games are aimed at removing the challenger's dependency on the random oracle and enabling it to derive the Π^+ export keys without knowing the sessions' master secrets. To this end, the challenger will begin to maintain a list L_G which it will use to simulate the random oracle G and derive the sessions' export keys. The entries of L_G are tuples of the form $(ms, n, aux, ek, [*])$, where $ms \in \{0,1\}^\lambda \cup \{\perp\}$, $n \in \{0,1\}^{2\lambda}$, $ek \in \{0,1\}^\lambda$, $aux \in \{0,1\}^*$, and $[*]$ denotes a list that contains zero or more session oracles. Specifically, we use the notation "$[\,]$" to denote an empty list, "$[\pi_i^s]$" for a list containing exactly π_i^s, "$[\pi_i^s, *]$" for a list containing π_i^s plus zero or more (unspecified) sessions, and "$[*]$" for a list containing zero or more (unspecified) sessions. L_G is initially empty and is filled out either in response to $\mathcal{A}$'s random oracle queries or when a session reaches the `accepted` state.

All the remaining games either change the way export keys are derived for newly accepted sessions (which we call the "Send-code"), or how they answer random oracle calls (which we call the "G-code"). The evolution of the Send-code in Game 2 through Game 6 is shown in Fig. 2, while the corresponding G-code is shown in Fig. 3. We annotate the changes made to a game relative to the previous one using red boxes. Note that some games make changes to both the Send-code and G-code simultaneously.

Game 2. This game introduces the list L_G. When a session π_i^s accepts with master secret ms, nonces $n = n_C \| n_S$, and auxiliary data aux, the challenger uses the Send-code shown in the panel labeled "Game 2" in Fig. 2 to derive its export key. It uses the G-code shown in the panel labeled "Game 2" in Fig. 3 to answer the adversary's random oracle queries. We claim that

$$\mathbf{Adv}^{\mathsf{G}_1}_{\Pi^+}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{G}_2}_{\Pi^+}(\mathcal{A}) \ . \tag{13}$$

Since the challenger considers all of the input values to the random oracle when answering from L_G in this game—in particular, it explicitly looks at the master secrets of the sessions—and because a random oracle always returns the same value when given the same input twice, the answers in Game 2 are distributed exactly like in Game 1.

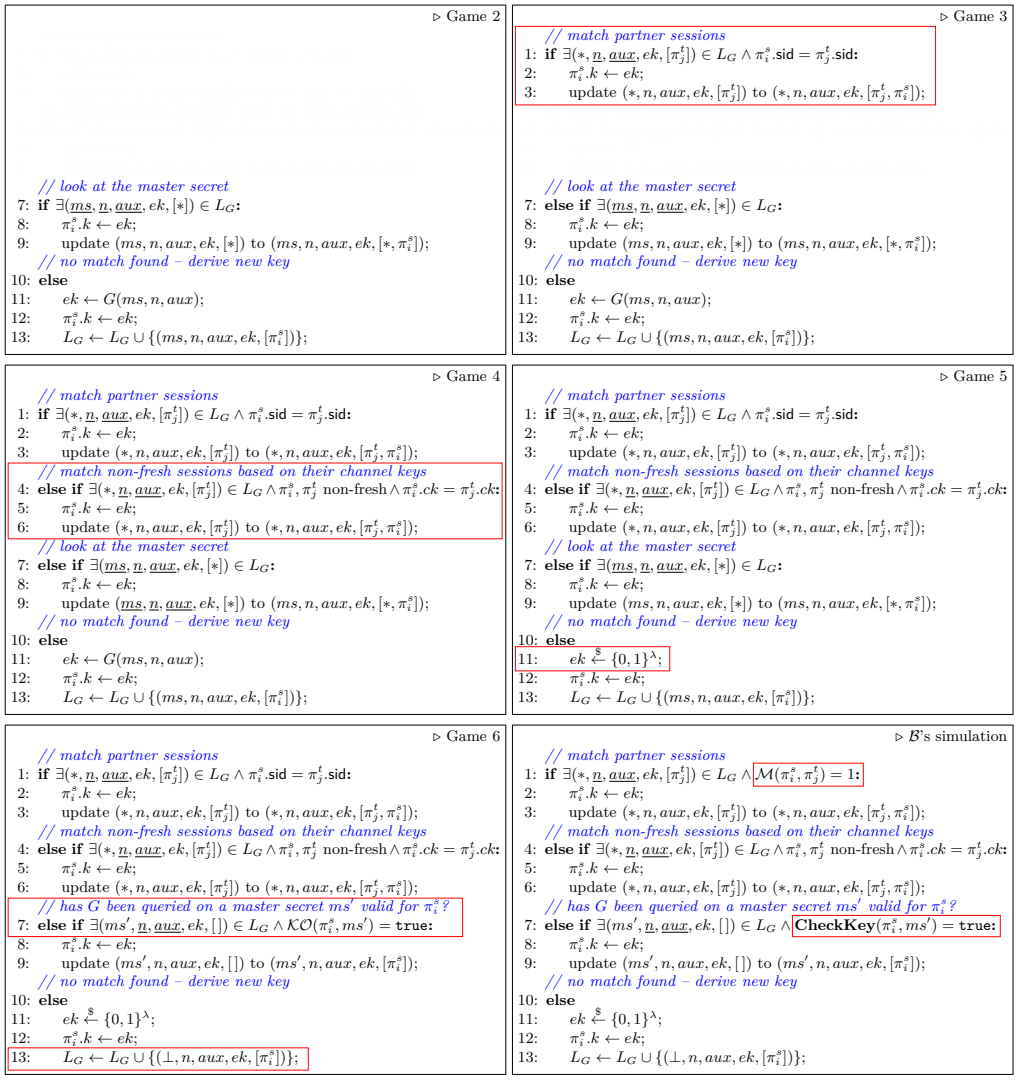

Fig. 2. How to derive the export key ek of a session π_i^s that accepted with master secret ms, nonces $n = n_C \| n_S$, and auxiliary data aux, in Game 2 to Game 6, and in $\mathcal{B}$'s simulation. Variables with underscores denote those that are 'pattern matched" against π_i^s's variables. For example, π_i^s is "matched" to $(a, \underline{b}, \underline{c}, ek, [*]) \in L_G$ only if $n_C \| n_S = b$, and $aux = c$. In particular, ms could be different from a.

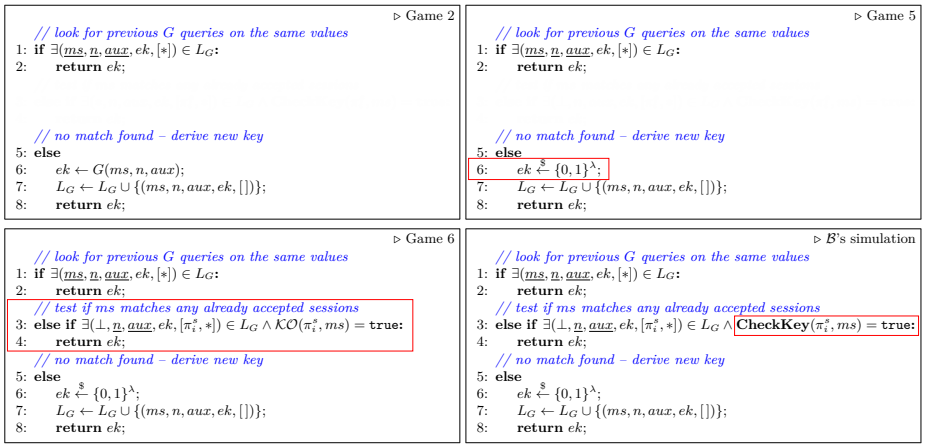

Fig. 3. How $\mathcal{A}$'s G queries, being of the form $G(ms, n, aux)$, are answered in Game 2 to Game 6, and in $\mathcal{B}$'s simulation.

In the remaining games, we define $ck\text{-}\mathsf{coll}_{\mathsf{i}}$ to be the event that during the run of Game i, the challenger calls the key derivation function $\Pi.\mathsf{Kdf}$ on two different master secrets $ms \neq ms'$, but with the same nonces $n = n_C \| n_S$ and additional input $F_\Pi(T)$, such that $\Pi.\mathsf{Kdf}(ms, n, F_\Pi(T)) = \Pi.\mathsf{Kdf}(ms', n, F_\Pi(T))$. We call event $ck\text{-}\mathsf{coll}_{\mathsf{i}}$ a *channel key collision*.

Game 3. In this game the Send-code is modified so that when a session accepts, the challenger first checks whether the session's partner is present in a tuple on L_G before deriving its export key (see the panel labeled "Game 3" in Fig. 2). The G-code remains unchanged. We claim that unless a channel key collision occurs, then Game 3 and Game 2 are identical.

To see this, suppose the if-check at line 1 of Game 3 matched two sessions π_i^s and π_j^t. This means that $\pi_i^s.\mathsf{sid} = \pi_j^t.\mathsf{sid}$, which by equation (2), implies that they have the same channel key. Then our assumption that no key collision occurs further implies that they must also have the same master secret. Hence, the else-if check at line 7 would also have matched π_i^s and π_j^t in Game 2. This shows that Game 2 and Game 3 matches exactly the same sessions when no channel key collision occurs, hence

$$\mathbf{Adv}^{\mathsf{G}_2}_{\Pi+}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{G}_3}_{\Pi+}(\mathcal{A}) + \Pr[ck\text{-}\mathsf{coll}_3] \ . \tag{14}$$

To bound $\Pr[ck\text{-}\mathsf{coll}_3]$ we create an algorithm $\mathcal{C}'$ that finds (KDF) collisions in $\Pi.\mathsf{Kdf}$ such that

$$\Pr[ck\text{-}\mathsf{coll}_3] \leq \mathbf{Adv}^{\mathsf{KDFcoll}}_{\Pi.\mathsf{Kdf}}(\mathcal{C}') \ . \tag{15}$$

Algorithm $\mathcal{C}'$ emulates adversary $\mathcal{A}$ and the challenger in an execution of Game 3 by instantiating all the parties' long-term keys and running all the

sessions according to the specification of the game. If event $ck\text{-}\mathsf{coll}_3$ happened during this run, say due to calls $\Pi.\mathsf{Kdf}(ms, n, F_\Pi(T))$ and $\Pi.\mathsf{Kdf}(ms', n, F_\Pi(T))$, then algorithm $\mathcal{C}'$ outputs $((ms, ms'), n, F_\Pi(T))$ as its collision for $\Pi.\mathsf{Kdf}$.

Since $\mathcal{C}'$ holds all the keys, it can simulate Game 3 perfectly. In particular, it can correctly simulate the random oracle G in those places where it is called inside of Game 3 (i.e., line 11 of the Send-code, and line 6 of the G-code). Thus, the probability that $\mathcal{C}'$ finds a collision in $\Pi.\mathsf{Kdf}$ is exactly the probability that event $ck\text{-}\mathsf{coll}_3$ occurs during its simulation of Game 3 for $\mathcal{A}$.

Remark 4. The reason we have to condition on there being no channel key collision in Game 3 is because we do not assume that equal session identifiers implies equal master secrets (cf. equation (2)). It is conceivable that two partner sessions might end up with the same channel key (and export key) even if their master secrets differ. This would lead to a discrepancy in how G queries are answered in Game 2 and Game 3.

Game 4. In this game the Send-code is augmented by matching non-fresh sessions based on their channel keys (see Fig. 2). That is, if two non-fresh sessions are found to have the same channel key (and the same nonces and auxiliary data), then they are given the same export key too. Again, as long as a channel key collision does not occur (event $ck\text{-}\mathsf{coll}_4$), then Game 4 and Game 3 are identical. Similarly, to bound $\Pr[ck\text{-}\mathsf{coll}_4]$ we build an algorithm $\mathcal{C}''$ against the collision resistance of $\Pi.\mathsf{Kdf}$ just like $\mathcal{C}'$ in Game 3. Thus

$$\mathbf{Adv}^{\mathsf{G}_3}_{\Pi^+}(\mathcal{A}) - \mathbf{Adv}^{\mathsf{G}_4}_{\Pi^+}(\mathcal{A}) \leq \Pr[ck\text{-}\mathsf{coll}_4] \leq \mathbf{Adv}^{\mathsf{KDFcoll}}_{\Pi.\mathsf{Kdf}}(\mathcal{C}'') \ . \tag{16}$$

Game 5. In this game the challenger replaces the calls to the random oracle (both in the Send-code and in the G-code) with strings drawn uniformly at random. We claim that this change does not affect $\mathcal{A}$'s view compared to Game 4 in any way, hence

$$\mathbf{Adv}^{\mathsf{G}_4}_{\Pi^+}(\mathcal{A}) = \mathbf{Adv}^{\mathsf{G}_5}_{\Pi^+}(\mathcal{A}) \ . \tag{17}$$

To prove (17) we show that the challenger in Game 4 never repeats a call to the random oracle on the same input. Thus, replacing these calls with uniformly drawn strings in Game 5 yields exactly the same distribution on the export keys.

Suppose at some point during Game 4 the challenger made the random oracle call $G(ms, n, aux)$ for the first time (either due to a session accepting, or because $\mathcal{A}$ made this exact G query). Suppose the random oracle responded with ek, and let $t = (ms, n, aux, ek, [*])$ be the tuple that was added to L_G in response to this call.

If the adversary later makes a G query on the same values, i.e. a query of the form $G(ms, n, aux)$, then line 1 of the G-code will be used to answer the query. Thus, the random oracle call on line 6 of the G-code would never be made on the same values twice in Game 4.

Likewise, if a session π_i^s accepts with the same values, i.e., master secret ms, nonces $n = n_C \| n_S$, and auxiliary data aux, *after* the initial G query was made,

then the else-if check on line 7 of the Send-code would match π_i^s to t. Thus, the random oracle call on line 11 of the Send-code would not be made on the same values twice in Game 4 either.

In the final game the challenger will derive the sessions' export keys independently of their master secrets. To do this, it will use a probabilistic *key-checking oracle* $\mathcal{KO}$ to test whether the adversary ever queried the random oracle at the correct master secret of a session. Oracle $\mathcal{KO}$ is defined as follows:

$$\mathcal{KO}(\pi_i^s, ms') := \begin{cases} \texttt{true} & \text{with probability 1 when } \pi_i^s.ms = ms', \\ \texttt{false} & \text{with probability } (1-\epsilon) \text{ when } \pi_i^s.ms \neq ms' \end{cases} . \tag{18}$$

Specifically, $\mathcal{KO}$ has a *one-sided error probability* since it can potentially return `true` (with probability ϵ) when $\pi_i^s.ms \neq ms$. Based on $\mathcal{KO}$ we define the following event, which will be important in our later analysis:

$$Q : \mathcal{KO} \text{ returns } \texttt{true} \text{ when called on a fresh session.} \tag{19}$$

We will later show that $\mathcal{A}$ has zero advantage in guessing the Test-challenge correctly unless Q happens (Lemma 1). Note that, if event Q happened, say due to a call $\mathcal{KO}(\pi_i^s, ms')$, then this does not necessarily imply that $\pi_i^s.ms = ms'$; namely, event Q also includes those cases where $\mathcal{KO}$ erroneously returns `true`.

Game 6. Game 6 modifies the else-if clause at line 7 of the Send-code in Game 5 to use the key-checking oracle $\mathcal{KO}$ instead of explicitly looking at a session's master secret. In addition, if a session accepts without a match on L_G, then Game 6 omits its master secret from the tuple that gets added to L_G (line 13). The G-code of Game 6 is also changed to use $\mathcal{KO}$, as shown in Fig. 3.

We claim that as long as $\mathcal{KO}$ does not make a mistake, then Game 6 and Game 5 are identical:

$$\mathbf{Adv}^{\mathsf{G}_5}_{\Pi^+}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{G}_6}_{\Pi^+}(\mathcal{A}) + \epsilon \ . \tag{20}$$

Let $t^\perp$ denote the tuple derived from $t = (ms, n, aux, ek, [*]) \in L_G$ by setting $ms = \perp$. To show (20) we prove the following three invariants.

(i) A session π_i^s accepts with master secret ms, nonces $n = n_C \| n_S$ and auxiliary data aux in Game 5 if and only if it accepts with the same master secret, nonces and auxiliary data, and at the same time instance, in Game 6.

(ii) A session π_i^s gets matched to a tuple $t \in L_G$ by one of the if/else-if clauses in the Send-code of Game 5 if and only if π_i^s gets matched to t or $t^\perp$ by the corresponding else/if-else clause in Game 6.

(iii) A G query is answered using tuple $t = (ms, n, aux, ek, [*]) \in L_G$ at line 1 of the G-code in Game 5 if and only if it is answered by $t \in L_G$ at line 1, or $t^\perp \in L_G$ at line 3, in Game 6.

We only show that (i) holds for the *first* accepting session since (ii) and (iii) implies that it also holds for all subsequent sessions.

(i) Fix a tape of random coins and some adversary $\mathcal{A}$, and consider a run of $\mathcal{A}$ in Game 5 and Game 6 using this tape as the source of randomness (both for the adversary and the challenger). Suppose π_i^s was the first session that accepted in this run of Game 5, say with values ms, $n = n_C \| n_S$, aux. If $\mathcal{A}$ made no G queries before π_i^s accepted, then π_i^s would have accepted with the same values (and at the same time) in the corresponding run in Game 6 too, since there are no differences between the two games up until this point. On the other hand, if $\mathcal{A}$ first made, say q_0, G queries before π_i^s accepted, then these queries would have been answered identically by the G-code in both Game 5 and Game 6 (in particular, by the else-clause at line 5). Hence, π_i^s would have accepted identically in both games also in the case where $\mathcal{A}$ made prior G queries.

(ii) Note that the first two if/else-if clauses in the Send-code do not look at the master secret explicitly (as indicated by the "$*$"). Thus, these two checks behave identically in Game 5 and Game 6.

Next, if π_i^s got matched to $t = (ms, n, aux, ek, [*])$ at line 7 in Game 5, then we claim that $[*] = [\,]$. To see this, suppose $[*] = [\pi_j^t, *]$. Clearly $\pi_i^s.\mathsf{sid} \neq \pi_j^t.\mathsf{sid}$, since otherwise the if-check at line 1 would already have matched π_i^s and t. Furthermore, since we can assume that π_i^s has not accepted maliciously (otherwise the game would already have ended), both π_i^s and π_j^t must be non-fresh by the assumption that the nonces are part of the session identifiers and are unique (Game 1). But then the else-if check at line 4 would have matched π_i^s and t, contradicting our assumption that π_i^s got matched to t at line 7. Hence $[*] = [\,]$. It follows that π_i^s would also have gotten matched to t at line 7 in Game 6 (by assumption, $\pi_i^s.ms = ms$, so $\mathcal{KO}$ is guaranteed to return `true`).

Conversely, if π_i^s got matched to $t = (ms', n, aux, ek, [\,])$ at line 7 in Game 6, it means that $\mathcal{KO}(\pi_i^s, ms') = \texttt{true}$. Since we have conditioned on $\mathcal{KO}$ not making a mistake, $\pi_i^s.ms = ms'$. Moreover, since line 7 is the only check that considers tuples having $[*] = [\,]$ in Game 5, it follows that π_i^s would have gotten matched to t at this line in Game 5 too.

(iii) Line 1 of the G-code ensures that the answers to G queries are consistent with respect to repeated queries in both Game 5 and Game 6, so we only consider non-repeated G-queries.

Suppose $t = (ms, n, aux, ek, [*]) \in L_G$ was used to answer a G query of the form $G(ms, n, aux)$ in Game 5. Note that if $[*] = [\,]$, then this was a repeated G query, so we assume $[*] = [\pi_i^s, *]$. By (i) and (ii), $t^\perp$ must have been on L_G prior to the G query being made in Game 6, and consequently line 3 would have been used to answer it in this game ($\mathcal{KO}$ is guaranteed to return `true` since $\pi_i^s.ms = ms$).

Conversely, if $t^\perp = (\perp, n, aux, [\pi_i^s, *]) \in L_G$ was used at line 3 to answer the query $G(ms', n, aux)$ in Game 6, then $\mathcal{KO}(\pi_i^s, ms') = \texttt{true}$. Since we have conditioned on $\mathcal{KO}$ not making a mistake, it follows that $\pi_i^s.ms = ms'$. Thus, when $\mathcal{A}$ makes the G query in Game 5, t would already be on L_G by (i) and (ii), yielding the right answer at line 1.

This establishes (20). We now turn to the analysis of Game 6.

Analyzing Game 6. It remains to bound the right-hand terms in equation (20). First we show that unless $\mathcal{A}$ manages to get event Q to happen (q.v. equation (19)), then it has zero advantage in guessing the Test-challenge correctly.

Lemma 1. *Suppose $\mathcal{A}$ issued its* Test*-query against session π_i^s during Game 6, and that it output b' as its answer to the* Test*-challenge. Then*

$$\Pr[\pi_i^s.b = b' \mid \overline{Q}] = \frac{1}{2} \ , \tag{21}$$

i.e. $\mathcal{A}$ has zero advantage in answering the Test*-challenge correctly if event Q did not happen during Game 6.*

Proof. That event Q did not happen means that $\mathcal{KO}$ never returned `true` for any fresh session during Game 6. Since $\mathcal{KO}$ is always correct when rejecting a key, i.e. when outputting `false`, this implies that $\mathcal{A}$ never queried the random oracle on the correct master secret of any fresh session. In particular, this means that the derived export key of the Test-session in Game 6 is distributed exactly like that of a random key. Thus, the hidden bit of the Test-session is independent of the derived export key from $\mathcal{A}$'s point of view. □

Lemma 1 implies that it is sufficient to bound the probability of event Q and the probability of a session accepting maliciously in order to bound $\mathcal{A}$'s advantage in Game 6. To this end, we construct an ACCE adversary $\mathcal{B}$ against the underlying protocol Π, which instantiates the key-checking oracle $\mathcal{KO}$ of Game 6 with a concrete procedure called **CheckKey**, such that

$$\mathbf{Adv}^{\mathsf{G_6\text{-}auth}}_{\Pi^+}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{auth\text{-}(PFS)}}_{\Pi}(\mathcal{B}), \tag{22}$$

$$\Pr[Q] \leq 2 \cdot \mathbf{Adv}^{\mathsf{chan\text{-}(PFS)}}_{\Pi}(\mathcal{B}) + \frac{2 \cdot qn_{\mathcal{P}}n_{\pi}}{2^{c\lambda}} \ . \tag{23}$$

Moreover, the **CheckKey** procedure will allow us to put a concrete bound on the failure probability ϵ in equation (20), specifically

$$\epsilon \leq 2 \cdot \Pr[Q] + \mathbf{Adv}^{\mathsf{KDFcoll}}_{\Pi.\mathsf{Kdf}}(\mathcal{C}''') \ . \tag{24}$$

We prove (22), (23), and (24), in Lemmas 3, 4, and 2, respectively.

Description of algorithm $\mathcal{B}$. Algorithm $\mathcal{B}$ plays in an ACCE security experiment against protocol Π and will use adversary $\mathcal{A}$ of Game 6 to win. Roughly speaking, algorithm $\mathcal{B}$ will simulate Game 6 for $\mathcal{A}$ by "embedding" the sessions in its own ACCE experiment into Game 6 and outfitting them with export keys. To derive these export keys, $\mathcal{B}$ maintains the list L_G which it fills out, and answers from, according to the Send and G-code shown in the last panels of Fig. 2 and Fig. 3, respectively (both labeled "$\mathcal{B}$'s simulation"). The difference between Game 6 and $\mathcal{B}$'s simulation is that $\mathcal{B}$ has to "implement" the key-checking oracle $\mathcal{KO}$ and also be able to correctly match partnered sessions.

To match partnered sessions, $\mathcal{B}$ uses one of the public session matching algorithms $\mathcal{M}$ guaranteed to exist for sid (since Π is TLS-like).

To instantiate the $\mathcal{KO}$ oracle, $\mathcal{B}$ uses the aforementioned procedure called **CheckKey**, which is formally defined in Algorithm 1 below. We will later show that **CheckKey** has the same properties as the key-checking oracle $\mathcal{KO}$ (as defined in equation (18)), but first we describe $\mathcal{B}$'s simulation in detail.

At the beginning of its ACCE security experiment, $\mathcal{B}$ receives the public keys of all the parties from its challenger E which it forwards to $\mathcal{A}$. Then $\mathcal{B}$ initializes L_G to an empty list and runs $\mathcal{A}$, answering its queries as follows:

- $\mathsf{NewSession}(P_i, \rho, \mathsf{pid})$: $\mathcal{B}$ forwards the query to its own ACCE challenger and, if $\rho = \texttt{init}$, returns the corresponding response back to $\mathcal{A}$.
- $\mathsf{Send}(\pi_i^s, m)$: $\mathcal{B}$ forwards the query to its own challenger and returns its response back to $\mathcal{A}$. Additionally, if m caused π_i^s to accept then $\mathcal{B}$ derives its export key by running the Send-code shown in the last panel of Fig. 2.
- $\mathsf{Corrupt}(P_i)$: $\mathcal{B}$ issues $\mathsf{Corrupt}(P_i)$ to its own challenger to obtain the secret key of P_i which it returns back to $\mathcal{A}$.
- $G(ms, n, aux)$: $\mathcal{B}$ answers this query by running the G-code shown in the last panel of Fig. 3.
- $\mathsf{Reveal}(\pi_i^s)/\mathsf{Test}(\pi_i^s)$: If $\pi_i^s.\alpha \neq \texttt{accepted}$, then $\mathcal{B}$ returns $\perp$. Otherwise, there will be an entry $(*, n, aux, ek, [\pi_i^s, *]) \in L_G$, and $\mathcal{B}$ returns ek.

In addition to the above, $\mathcal{B}$ stops and outputs a guess (π_i^s, b') to its ACCE challenger if one of the following events happen.

- *Two sessions generated the same nonce*: select π_i^s arbitrarily among the fresh sessions and draw b' randomly.
- *Event Q happened due to a call to* **CheckKey**(π_i^s, ms): if the ciphertext C decrypted to m_0 at line 20 of Algorithm 1, output $(\pi_i^s, 0)$, otherwise, if it decrypted to m_1, output $(\pi_i^s, 1)$.
- $\mathcal{A}$ *outputs a guess for the* Test*-challenge*: select π_i^s arbitrarily among the fresh sessions and draw b' randomly[7].

This ends the description of algorithm $\mathcal{B}$. Note that the only thing that differs between $\mathcal{B}$'s simulation and Game 6 is $\mathcal{B}$'s usage of the **CheckKey** procedure and the algorithm $\mathcal{M}$ for matching sessions. By definition, the latter is always correct, so $\mathcal{B}$'s simulation is sound given that **CheckKey** correctly implements the $\mathcal{KO}$ oracle.

Analysis of CheckKey. We need to show that **CheckKey** has the same properties as the key-checking oracle $\mathcal{KO}$ used in Game 6, i.e. that it always returns `true` if called on the right master secret of a session and returns `false` (with high probability) when not. The idea of **CheckKey** is to derive from the supplied master secret a guess on the session's channel key and then compare this to the channel key actually held by the session.

[7] By Lemma 1 it is immaterial whether $\mathcal{B}$ outputs a random bit or uses $\mathcal{A}$'s guess on the Test-challenge, since Q did not happen.

Algorithm 1 CheckKey(π_i^s, ms)

Note: The procedure is parameterized by $c \in \mathbb{N}$. Calls on the same input always return the same value, i.e. **CheckKey** records its results for every input combination. To simplify the presentation, we leave out the code that deals with this below.

Precondition: $(C_1, H_1), (C_2, H_2), \ldots, (C_k, H_k)$, are the encrypted handshake messages (if any) output by π_i^s during the run of Π^+, together with the corresponding additional data.

$x, y \xleftarrow{\$} \{0,1\}^\lambda$;
$(m_0, m_1) := (0\|x, 1\|y)$;

// n_C, n_S are the nonces, and T the transcript, π_i^s accepted with.
$ck' \leftarrow \Pi.\mathsf{Kdf}(ms, n_C\|n_S, F_\Pi(T))$;

if π_i^s is non-fresh**:**
    $ck \leftarrow \mathsf{Reveal}(\pi_i^s)$;
    return $ck \stackrel{?}{=} ck'$;
else
    $C \leftarrow \mathsf{Encrypt}(\pi_i^s, \ell, m_0, m_1, H)$; ▷ *obtain an encryption of $m_{\pi_i^s.b}$ under $\pi_i^s.ck$.*

    // "recreate" a decrypt state st'_D matching the encrypt state used to create C.
    $(*, st'_D) \leftarrow \mathsf{stE.Init}$;
    for all (C_r, H_r) **do**
        $(*, st'_D) \leftarrow \mathsf{stE.Dec}(ck', C_r, H_r, st'_D)$;
    for all C's from previous calls to **CheckKey**$(\pi_i^s, *)$ **do**
        $(*, st'_D) \leftarrow \mathsf{stE.Dec}(ck', C, H, st'_D)$;

    $(m', *) \leftarrow \mathsf{stE.Dec}(ck', C, H, st'_D)$; ▷ *locally decrypt C using ck' and st'_D.*
    return $m' \stackrel{?}{\in} \{m_0, m_1\}$;

For non-fresh sessions this is straightforward since $\mathcal{B}$ can just make a Reveal query in order to obtain their channel keys and make the comparison directly (line 9 in Algorithm 1). On the other hand, issuing a Reveal query to a fresh session would "destroy" its status as a valid target in the ACCE game, preventing $\mathcal{B}$ from capitalizing on the event where $\mathcal{A}$ queries the random oracle on the master secret of a fresh session.

For fresh sessions **CheckKey** instead tests the validity of a derived channel key *indirectly* by trying to (locally) decrypt a ciphertext that was legitimately created with the actual channel key of the session. To obtain this ciphertext, **CheckKey** exploits $\mathcal{B}$'s access to a left-or-right encryption oracle for every session in the ACCE game (i.e., the Encrypt query). However, **CheckKey** is complicated by the statefulness of the sLHAE scheme. That is, before attempting to (locally) decrypt the ciphertext at line 20 of Algorithm 1, **CheckKey** first needs to "recreate" a valid decryption state. This is done as follows: starting from the initial state of the sLHAE scheme, **CheckKey** chronologically decrypts each encrypted message output by the session during the handshake (if any). Then it decrypts all ciphertext messages created in prior calls to **CheckKey** (because these advance the session's encrypt state st_E). Finally, it attempts the decryption of C. If the correct channel key was used, then this process is guaranteed to generate a decryption state st'_D that "matches"[8] the encrypt state st_E which was used to create the ciphertext C (due to the correctness of the sLHAE scheme).

Since $\Pi.\mathsf{Kdf}$ is deterministic, the above shows that $\mathbf{CheckKey}(\pi_i^s, ms)$ will always return `true` if ms is equal to the master secret of π_i^s, since the derived channel key ck' will then equal $\pi_i^s.ck$.

Conversely, if **CheckKey** is called on a wrong master secret, then it does indeed have a one-sided error probability. In particular, let fresh (resp. non-fresh) denote that **CheckKey** was called on a fresh (resp. non-fresh) session, and let CKerror denote that a call to **CheckKey** erroneously returned `true`. Note that for a fresh session π_i^s and master secret ms', this requires that the decryption of C at line 20 of Algorithm 1 returned one of the two messages (m_0, m_1) associated to the pair (π_i^s, ms'). Letting $b = \pi_i^s.b$, we write correctDec for the event that C decrypted to m_b, and wrongDec for the event that it decrypted to $m_{\bar{b}}$[9]. Consequently, CKerror can be partitioned as follows, depending on whether the session was fresh or not.

$$\begin{aligned} \mathsf{CKerror} &= (\mathsf{CKerror} \cap \mathsf{fresh}) \cup (\mathsf{CKerror} \cap \mathsf{non\text{-}fresh}) && (25) \\ &= (\mathsf{CKerror} \cap (\mathsf{correctDec} \cup \mathsf{wrongDec})) \cup (\mathsf{CKerror} \cap \mathsf{non\text{-}fresh}) \ . && (26) \end{aligned}$$

By the above we have shown that **CheckKey** correctly implements the key-checking oracle $\mathcal{KO}$. Moreover, we can now provide concrete bounds on the error probability ϵ in (20) by bounding CKerror.

[8] The recreated state st'_D does not necessarily have to be *equal* to the decryption state held by π_i^s — it only needs to yield a valid decryption.

[9] Event correctDec can either happen legitimately ($\pi_i^s.ms = ms'$), or because of an error ($\pi_i^s.ms \neq ms'$). On the other hand, event wrongDec can only happen due to an error.

Lemma 2.

$$\Pr[\mathsf{CKerror}] = \Pr[\mathsf{CKerror} \cap \mathsf{fresh}] + \Pr[\mathsf{CKerror} \cap \mathsf{non\text{-}fresh}] \tag{27}$$

$$\leq \Pr[\mathsf{correctDec}] + \Pr[\mathsf{wrongDec}] + \Pr[ck\text{-}\mathsf{coll}_6] \tag{28}$$

$$\leq 2 \cdot \Pr[Q] + \mathbf{Adv}^{ck\text{-}\mathsf{coll}}_{\Pi.\mathsf{Kdf}}(\mathcal{C}'''). \tag{29}$$

Proof. For $\mathsf{CKerror} \cap \mathsf{fresh}$, note that $\mathsf{correctDec}$ and $\mathsf{wrongDec}$ are mutually exclusive since $\mathcal{B}$ aborts as soon as one of them happens. Also, in the context of **CheckKey**, they are both sub-events of Q. Thus, $\Pr[\mathsf{CKerror} \cap \mathsf{fresh}] = \Pr[\mathsf{correctDec}] + \Pr[\mathsf{wrongDec}] \leq 2 \cdot \Pr[Q]$.

If $\mathsf{CKerror} \cap \mathsf{non\text{-}fresh}$ happens in Game 6, then event $ck\text{-}\mathsf{coll}_6$ must by definition have happened too. Hence $\Pr[\mathsf{CKerror} \cap \mathsf{non\text{-}fresh}] \leq \Pr[ck\text{-}\mathsf{coll}_6]$. Furthermore, the bound $\Pr[ck\text{-}\mathsf{coll}_6] \leq \mathbf{Adv}^{ck\text{-}\mathsf{coll}}_{\Pi.\mathsf{Kdf}}(\mathcal{C}''')$ follows from the same strategy used in the game hop from Game 2 to 3, and from Game 3 to 4. That is, we construct an algorithm $\mathcal{C}'''$ that plays the challenger in Game 6; once $ck\text{-}\mathsf{coll}_6$ occurs in this game, then $\mathcal{C}'''$ has found a collision in $\Pi.\mathsf{Kdf}$. □

Analysis of $\mathcal{B}$. Having shown that $\mathcal{B}$'s simulation of Game 6 is sound, we now turn to bounding $\mathcal{A}$'s advantage in Game 6 in terms of $\mathcal{B}$'s advantage in the ACCE security experiment.

Lemma 3.

$$\mathbf{Adv}^{\mathsf{G}_6\text{-}\mathsf{auth}}_{\Pi^+}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{auth}\text{-}(\mathsf{PFS})}_{\Pi}(\mathcal{B}) \ . \tag{30}$$

Proof. Since $\mathcal{B}$'s simulation of Game 6 is sound, and because the protocols Π^+ and Π have the same session identifier, it follows that $\mathcal{A}$ gets a session to accept maliciously in Game 6, if and only if the session accepts maliciously in the underlying ACCE security experiment in $\mathcal{B}$'s simulation. □

Lemma 4.

$$\mathbf{Adv}^{\mathsf{G}_6\text{-}\mathsf{chan}}_{\Pi^+}(\mathcal{A}) \leq \Pr[Q] \leq 2 \cdot \mathbf{Adv}^{\mathsf{chan}\text{-}(\mathsf{PFS})}_{\Pi}(\mathcal{B}) + \frac{2qn_{\mathcal{P}}n_{\pi}}{2^{c\lambda}} \ . \tag{31}$$

Proof. The first inequality follows from Lemma 1. The proof of the second inequality amounts to a direct calculation based on conditional probabilities. Suppose $\mathcal{B}$ halted with output (π_i^s, b') in its ACCE security experiment, where π_i^s is some fresh session. By conditioning on whether event Q happened or not during $\mathcal{B}$'s simulation of Game 6 for $\mathcal{A}$, we get that $\mathcal{B}$'s probability of breaking the

ACCE channel is:

$$\Pr[\pi_i^s.b = b'] = \Pr[\pi_i^s.b = b' \mid Q] \cdot \Pr[Q] + \Pr[\pi_i^s.b = b' \mid \overline{Q}] \cdot \Pr[\overline{Q}] \tag{32}$$

$$\stackrel{a)}{=} \Pr[\pi_i^s.b = b' \mid Q] \cdot \Pr[Q] + \frac{1}{2}(1 - \Pr[Q]) \tag{33}$$

$$\stackrel{b)}{=} \Big(\overbrace{\Pr[\pi_i^s.b = b' \mid Q \cap \mathsf{correctDec}]}^{=1} \cdot \Pr[\mathsf{correctDec} \mid Q] + \overbrace{\Pr[\pi_i^s.b = b' \mid Q \cap \mathsf{wrongDec}]}^{=0} \cdot \Pr[\mathsf{wrongDec} \mid Q] \Big) \cdot \Pr[Q] + \frac{1}{2}(1 - \Pr[Q]) \tag{34}$$

$$= \Pr[\mathsf{correctDec} \mid Q] \cdot \Pr[Q] + \frac{1}{2}(1 - \Pr[Q]) \tag{35}$$

$$= \Pr[\mathsf{correctDec} \cap Q] - \frac{1}{2} \cdot \Pr[Q] + \frac{1}{2} \tag{36}$$

$$\stackrel{c)}{=} \big(\Pr[Q] - \Pr[\mathsf{wrongDec} \cap Q] \big) - \frac{1}{2} \Pr[Q] + \frac{1}{2} \tag{37}$$

$$= \frac{1}{2} \Pr[Q] - \Pr[\mathsf{wrongDec} \cap Q] + \frac{1}{2} \tag{38}$$

$$\stackrel{d)}{=} \frac{1}{2} \Pr[Q] - \Pr[\mathsf{wrongDec}] + \frac{1}{2} \tag{39}$$

$$\geq \frac{1}{2} \Pr[Q] - \frac{q n_{\mathcal{P}} n_{\pi}}{2^{c\lambda}} + \frac{1}{2} \ . \tag{40}$$

In a) we used the fact that $\mathcal{B}$ outputs a random bit when Q does not happen, b) and c) used that $Q = \mathsf{correctDec} \cup \mathsf{wrongDec}$ and $\mathsf{correctDec} \cap \mathsf{wrongDec} = \emptyset$, and d) used that $\mathsf{wrongDec} \subseteq Q$. We prove the final inequality as follows.

Let $\overline{b} = 1 - \pi_i^s.b$ and let (m_0, m_1) be the two messages associated to the pair (π_i^s, ms) in **CheckKey**. Since $m_{\overline{b}}$ is independent of the ciphertext C produced at line 11 of Algorithm 1, the probability that C decrypts to $m_{\overline{b}}$ at line 20 is statistically bounded by $2^{-c\lambda}$ for any key k. By taking the union bound over all parties, the number of sessions per party, and the number of random oracle calls, we get that $\Pr[\mathsf{wrongDec}] \leq q n_{\mathcal{P}} n_{\pi} / 2^{c\lambda}$.

Solving (40) for $\Pr[Q]$ yields the second inequality in Lemma 4. □

Concluding the proof of Theorem 1. Applying Lemmas 2, 3, and 4, we get that the right-hand side of equation (20) is bounded by

$$\mathbf{Adv}_{\Pi}^{\mathsf{auth\text{-}(PFS)}}(\mathcal{B}) + 6 \cdot \mathbf{Adv}_{\Pi}^{\mathsf{chan\text{-}(PFS)}}(\mathcal{B}) + \frac{6 q n_{\mathcal{P}} n_{\pi}}{2^{c\lambda}} + \mathbf{Adv}_{\Pi.\mathsf{Kdf}}^{\mathsf{KDFcoll}}(\mathcal{C}''') \ . \tag{41}$$

By collecting all the probabilities from Game 0 to Game 6, and letting $\mathcal{C} = \max_{\mathbf{Adv}_{\Pi.\mathsf{Kdf}}^{\mathsf{KDFcoll}}} \{\mathcal{C}', \mathcal{C}'', \mathcal{C}'''\}$, the theorem follows.

4.4 Application to EAP-TLS and TLS Key Material Exporters

EAP [1] is a widely used authentication framework which defines a set of generic message formats and message flows. EAP is not a specific authentication mechanism on its own, but is instead used to encapsulate another concrete authentication protocol, like TLS, IKEv2 or IEEE 802.1X, known as a *method*. Each EAP method can additionally specify a way of generating keying material, known as *export keys*, both for internal and external use. For example, in EAP-TLS [33] the export key ek is derived as follows:

$$ek := \mathsf{tls.PRF}(ms, \text{“}\texttt{client EAP encryption}\text{”}, n_C \| n_S) \ , \tag{42}$$

where ms is the master secret and n_C, n_S the nonces established during the TLS handshake. How export keys should be derived from the TLS handshake in settings outside of EAP is defined in RFC 5705: “Keying Material Exporters for Transport Layer Security (TLS)” [31]. Besides a different constant label string, RFC 5705 defines ek almost exactly as in (42). The only difference is that it also allows an extra context value aux to be added into the key derivation together with the nonces. For both EAP-TLS and RFC 5705 the security requirement on ek is that it be indistinguishable from random.

In order to apply Theorem 1 to EAP-TLS, we have to show that TLS is in fact a TLS-like ACCE protocol, using a session identifier that satisfies the requirements of the theorem. Since several works have already proven TLS to be ACCE secure, it only remains to demonstrate that the session identifier used in these prior analyses allowed for public session matching and contained the sessions' nonces.

As an example, in their analysis of TLS, Krawczyk, Paterson, and Wee [24] defined their session identifier to consist of the two first flows between the client and the server, in addition to the client's KEM-value (either a Diffie-Hellman share or the pre-master secret encrypted with the server's public RSA key). This session identifier includes the parties' nonces, and allows for public session matching since it only consists of public values. Thus, using the TLS analysis of Krawczyk et al. [24], we can apply Theorem 1 with Π = TLS, and Π^+ = EAP-TLS, in order to get the following result.

Corollary 1 (AKE security of EAP-TLS).

$$\mathbf{Adv}^{\mathsf{AKE}}_{\mathsf{EAP\text{-}TLS}}(\mathcal{A}) \leq 6 \cdot \mathbf{Adv}^{\mathsf{ACCE}}_{\mathsf{TLS}}(\mathcal{B}) + 3 \cdot \mathbf{Adv}^{\mathsf{KDFcoll}}_{\mathsf{tls.PRF}}(\mathcal{C}) + \frac{6 q n_{\mathcal{P}} n_{\pi}}{2^{c\lambda}} + \frac{(n_{\mathcal{P}} n_{\pi})^2}{2^{\lambda+1}} \ , \tag{43}$$

where $\Pi.\mathsf{Kdf} = \mathsf{tls.PRF}$*, and all other quantities are defined as stated in Theorem 1.*

Remark 5. The KDF used in TLS is based on HMAC [23], and its KDF collision resistance follows from the (hash function) collision resistance of the underlying hash function used in HMAC (see Theorem 2, Appx. A).

Remark 6. JKSS [20] used *matching conversations* as their partnering mechanism in their analysis of TLS. Since matching conversations contain the parties' nonces and trivially allow for public session matching, it would seem like JKSS's analysis could also be used with Theorem 1 in order to establish Corollary 1.

However, there is a subtle technical difference between the ACCE model as defined in this paper and the ACCE model as defined by JKSS, stemming from the difference in choice of partnering mechanism. Specifically, in JKSS's definition of ACCE [19, Def. 11] one must forbid the adversary from issuing a Reveal query towards the server after it sent out its last message, but *before* the client to which it has a matching conversation received it. This is to avoid a trivial attack whereby the adversary re-encrypts the final message towards the client, getting it to accept maliciously (see [19, Remark 6] for further details)[10].

In contrast, the definition of ACCE used in this paper (in particular, Def. 3) allows all Reveal queries. It should be noted that the trivial attack in JKSS's model does not imply any actual weakness in TLS, but rather highlights a peculiarity of using matching conversations as the partnering mechanism when defining ACCE.

Remark 7. Brzuska et al. [8] defined their session identifier to consist of the parties' nonces and identities, together with the TLS *pre-master secret*. Unfortunately, basing the session identifier upon secret values does not in general allow for public session matching. For instance, if the KEM used in the TLS handshake was a *re-randomizable* encryption scheme [12,30], then the choice of Brzuska et al. [8] would not allow for public session matching (see also [9] for further details).

Remark 8. Bhargavan et al. [5] showed that the *full* TLS protocol, including resumption and renegotiation, is vulnerable to an *unknown key-share* attack [7]. The attack allows an adversary to synchronize the master secret and nonces of two non-partnered sessions, leading them to derive the same channel key. While the attack carries over to EAP-TLS, it does not invalidate Corollary 1, since our model does not consider resumption nor renegotiation. However, it should be noted that this has been done for the sake of simplicity, not because of an essential limitation in our analysis. Our result can be extended to incorporate features like renegotiation, resumption or ciphersuite and version negotiation, either by using the *multi-phase* ACCE model of Giesen et al. [16] or the *multi-ciphersuite* ACCE model of Bergsma et al. [4]. The former has been used to prove results on TLS with renegotiation [16], while the latter has been used to prove results on SSH and TLS with ciphersuite and version negotiation [4,14]. Since our proof uses the underlying ACCE protocol in an almost black-box way, by adopting one of the above models we would essentially "inherit" their corresponding results for EAP-TLS as well.

[10] This extra requirement was not included in the original published version [20], but was later added to the online version [19].

Acknowledgments

We would like to thank Colin Boyd and Britta Hale for helpful comments and discussions. Part of this work was done while Christina Brzuska was working for Microsoft Research, Cambridge, UK. Christina Brzuska is grateful to NXP Semiconductors for supporting her chair for IT Security Analysis. Håkon Jacobsen was hosted by Microsoft Research, Cambridge, UK, for parts of this work. Some of this work performed while Douglas Stebila was hosted by the Norwegian University of Science and Technology.

A KDF Collision Resistance of the TLS KDF

Let H be a hash function, and let $\overline{H}$ denote the HMAC function using H as its underlying hash function, namely

$$\overline{H}(k,m) \stackrel{\text{def}}{=} H\left(k \oplus \texttt{opad} \| H(k \oplus \texttt{ipad} \| m)\right) \ , \tag{44}$$

where `ipad` and `opad` are distinct constants.

The TLS 1.2 KDF is defined as follows, where the variable t depends on how much keying material is needed.

$$\mathsf{tls.PRF}(ms, L, n) \stackrel{\text{def}}{=} \Big|\Big|_{i=1}^{t} \overline{H}(ms, A(i) \| L \| n) \tag{45}$$

$$A(1) = \overline{H}(ms, n) \tag{46}$$

$$A(i) = \overline{H}(ms, A(i-1)) \tag{47}$$

In TLS 1.2, $L =$ "`key expansion`" and $n = n_C \| n_S$, where and n_C, n_S are the client and server nonces, respectively. For simplicity, we write $S = L \| n$.

Theorem 2. *A KDF collision (Def. 9) in* tls.PRF *implies a collision in* H.

Proof. Suppose $\mathsf{tls.PRF}(ms, L, n) = \mathsf{tls.PRF}(ms', L, n)$, with $ms \neq ms'$. By (45) we specifically have that

$$\overline{H}(ms, A(1) \| S) = \overline{H}(ms', A'(1) \| S), \tag{48}$$

where $A'(1) = \overline{H}(ms', n)$. Expanding (48) using (44) we get:

$$\begin{gathered} H\left(ms \oplus \texttt{opad} \| H\left(ms \oplus \texttt{ipad} \| A(1) \| S\right)\right) \\ = \\ H\left(ms' \oplus \texttt{opad} \| H\left(ms' \oplus \texttt{ipad} \| A'(1) \| S\right)\right) \ . \end{gathered} \tag{49}$$

Letting $X = H\left(ms \oplus \texttt{ipad} \| A(1) \| S\right)$ and $Y = H\left(ms' \oplus \texttt{ipad} \| A'(1) \| S\right)$ denote the "inner" hash function values, (49) becomes:

$$H(ms \oplus \texttt{opad} \| X) = H(ms' \oplus \texttt{opad} \| Y) \ . \tag{50}$$

Since $ms \oplus \texttt{opad} \neq ms' \oplus \texttt{opad}$, it follows that $ms \oplus \texttt{opad} \| X$ and $ms' \oplus \texttt{opad} \| Y$ constitute a collision in H. □

Remark 9. The construction of tls.PRF in TLS 1.0/1.1 is different from that in TLS 1.2 (shown in equation (45)). In versions prior to TLS 1.2, tls.PRF is defined as $P_{\text{MD5}} \oplus P_{\text{SHA1}}$, where P_{MD5} and P_{SHA1} are equal to the right-hand side of equation (45) with $\overline{H}$ using MD5 and SHA1, respectively. Theorem 2 only applies to the construction used in TLS 1.2.

References

1. Aboba, B., Blunk, L.J., Vollbrecht, J.R., Carlson, J., Levkowetz, H.: Extensible Authentication Protocol. RFC 3748, RFC Editor (June 2004), `https://tools.ietf.org/html/rfc3748`
2. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology. pp. 232–249. CRYPTO '93, Springer-Verlag New York, Inc., New York, NY, USA (1994)
3. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques. pp. 409–426. EUROCRYPT'06, Springer-Verlag, Berlin, Heidelberg (2006)
4. Bergsma, F., Dowling, B., Kohlar, F., Schwenk, J., Stebila, D.: Multi-ciphersuite security of the Secure Shell (SSH) Protocol. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 369–381. CCS '14, ACM, New York, NY, USA (2014)
5. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P.Y.: Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In: Security and Privacy (SP), 2014 IEEE Symposium on. pp. 98–113 (May 2014)
6. Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P., Béguelin, S.Z.: Proving the TLS handshake secure (as it is). In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8617, pp. 235–255. Springer (2014)
7. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the station-to-station (STS) protocol. In: Imai, H., Zheng, Y. (eds.) Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1560, pp. 154–170. Springer (1999)
8. Brzuska, C., Fischlin, M., Smart, N.P., Warinschi, B., Williams, S.C.: Less is more: relaxed yet composable security notions for key exchange. International Journal of Information Security 12(4), 267–297 (2013)
9. Brzuska, C., Fischlin, M., Warinschi, B., Williams, S.C.: Composability of Bellare-Rogaway key exchange protocols. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. pp. 51–62. CCS '11, ACM, New York, NY, USA (2011)
10. Brzuska, C., Smart, N.P., Warinschi, B., Watson, G.J.: An analysis of the EMV channel establishment protocol. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013. pp. 373–386. ACM (2013)

11. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding. Lecture Notes in Computer Science, vol. 2045, pp. 453–474. Springer (2001)
12. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 565–582. Springer (2003)
13. Diffie, W., Van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Cryptography 2(2), 107–125 (Jun 1992)
14. Dowling, B., Stebila, D.: Modelling ciphersuite and version negotiation in the TLS protocol. In: Foo, E., Stebila, D. (eds.) Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9144, pp. 270–288. Springer (2015)
15. Fischlin, M., Günther, F.: Multi-stage key exchange and the case of Google's QUIC protocol. In: Ahn, G., Yung, M., Li, N. (eds.) Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014. pp. 1193–1204. ACM (2014)
16. Giesen, F., Kohlar, F., Stebila, D.: On the security of TLS renegotiation. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013. pp. 387–398. ACM (2013)
17. He, C., Sundararajan, M., Datta, A., Derek, A., Mitchell, J.C.: A modular correctness proof of IEEE 802.11i and TLS. In: Proceedings of the 12th ACM Conference on Computer and Communications Security. pp. 2–15. CCS '05, ACM, New York, NY, USA (2005)
18. Hofheinz, D., Kiltz, E.: Secure hybrid encryption from weakened key encapsulation. In: Menezes, A. (ed.) Advances in Cryptology - CRYPTO 2007, Lecture Notes in Computer Science, vol. 4622, pp. 553–571. Springer Berlin Heidelberg (2007)
19. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. Cryptology ePrint Archive, Report 2011/219 (2011), `https://eprint.iacr.org/2011/219`
20. Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 273–293. Springer (2012)
21. Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367 (2013), `https://eprint.iacr.org/2013/367`
22. Kohlweiss, M., Maurer, U., Onete, C., Tackmann, B., Venturi, D.: (De-)constructing TLS. Cryptology ePrint Archive, Report 2014/020 (2014), `https://eprint.iacr.org/2014/020`
23. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational) (February 1997), `https://tools.ietf.org/html/rfc2104`

24. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J. (eds.) Advances in Cryptology – CRYPTO 2013, Lecture Notes in Computer Science, vol. 8042, pp. 429–448. Springer Berlin Heidelberg (2013)
25. Küsters, R., Tuengerthal, M.: Composition theorems without pre-established session identifiers. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011. pp. 41–50. ACM (2011)
26. Li, Y., Schäge, S., Yang, Z., Kohlar, F., Schwenk, J.: On the security of the pre-shared key ciphersuites of TLS. In: Krawczyk, H. (ed.) Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8383, pp. 669–684. Springer (2014)
27. Lychev, R., Jero, S., Boldyreva, A., Nita-Rotaru, C.: How secure and quick is quic? provable security and performance analyses. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 214–231. IEEE Computer Society (2015)
28. Maurer, U., Renner, R.: Abstract cryptography. In: Chazelle, B. (ed.) Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings. pp. 1–21. Tsinghua University Press (2011)
29. Morrissey, P., Smart, N.P., Warinschi, B.: A modular security analysis of the TLS handshake protocol. In: Pieprzyk, J. (ed.) Advances in Cryptology - ASIACRYPT 2008, Lecture Notes in Computer Science, vol. 5350, pp. 55–73. Springer Berlin Heidelberg (2008)
30. Prabhakaran, M., Rosulek, M.: Rerandomizable RCCA encryption. In: Menezes, A. (ed.) Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4622, pp. 517–534. Springer (2007)
31. Rescorla, E.: Keying Material Exporters for Transport Layer Security (TLS). RFC 5705, RFC Editor (March 2010), `https://tools.ietf.org/html/rfc5705`
32. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), `https://eprint.iacr.org/2004/332`
33. Simon, D., Aboba, B., Hurst, R.: The EAP-TLS Authentication Protocol. RFC 5216, RFC Editor (March 2008), `https://tools.ietf.org/html/rfc5216`