An abridged version of this paper appears in the proceedings of PKC 2017, DOI: 10.1007/978-3-662-54388-7_12. This is the full version.

A Modular Security Analysis of EAP and IEEE 802.11

Chris Brzuska[1] and Håkon Jacobsen*,[2]

[1]Hamburg University of Technology, Hamburg, Germany
brzuska@tuhh.de
[2]Norwegian University of Science and Technology, Trondheim, Norway
hakoja@item.ntnu.no

March 19, 2017

Abstract

We conduct a reduction-based security analysis of the Extensible Authentication Protocol (EAP), a widely used three-party authentication framework. EAP is often found in enterprise networks where it allows a client and an authenticator to establish a shared key with the help of a mutually trusted server. Considered as a three-party authenticated key exchange protocol, we show that the general EAP construction achieves a security notion we call 3P-AKE^w. The 3P-AKE^w security notion captures the idea of *weak forward secrecy* and is a simplified three-party version of the well-known eCK model in the two-pass variant. Our analysis is modular and reflects the compositional nature of EAP.

Additionally, we show that the security of EAP can easily be upgraded to provide *full* forward secrecy simply by adding a subsequent key-confirmation step between the client and the authenticator. In practice this key-confirmation step is often carried out in the form of a 2P-AKE protocol which uses EAP to bootstrap its authentication. A concrete example is the extremely common IEEE 802.11 protocol used in WLANs. In enterprise settings EAP is often used in conjunction with IEEE 802.11 in order to allow the wireless client to authenticate itself to a wireless access point (the authenticator) through some centrally administrated server. Building on our modular results for EAP, we get as our second major result the first reduction-based security result for IEEE 802.11 combined with EAP.

*Håkon Jacobsen was supported by a STSM Grant from COST Action IC1306.

Contents

1 Introduction

The Extensible Authentication Protocol (EAP), specified in RFC 3748 [4], is a widely used authentication framework for network access control. It is particularly common in wireless networks, being used by protocols like IEEE 802.11 (Wi-Fi), IEEE 802.16 (WiMAX) and various 3G/4G mobile networks. The typical use case of EAP is in settings where a *client* seeks to gain access to a network controlled by an *authenticator*, but where the client and authenticator do not share any common credentials. EAP allows the client and authenticator to authenticate each other based on a mutually trusted *server*. Technically, EAP is not a specific authentication mechanism on its own, rather it specifies a generic three-party framework for composing other concrete authentication protocols. This provides applications of EAP the freedom to choose whatever concrete instantiation is suitable for their own specific setting. The success of this approach is apparent by the huge and diverse set of real-life deployments using the EAP framework.

Surprisingly then, given its prevalence and importance, there has been no formal reduction-based provable security analysis of EAP. One reason for this might be due to the general nature of EAP itself. As mentioned above, EAP is not a single protocol on its own, but relies on other sub-protocols to instantiate it. As such, many things in the EAP specification are left unspecified or considered out of scope. On the other hand, in order to conduct a formal security analysis of EAP, these details matter and require a careful treatment. More generally, the need to make assumptions on protocols outside of the EAP standard makes it harder to analyze as described by Hoeper and Chen [23].

Another reason for the lack of reductionist-based security results on EAP might be due to the fact that it is a three-party protocol. As pointed out by Schwenk in his recent work on Kerberos [41], apart from a few papers like [10, 3, 37, 5, 41] relatively little work has been done on three-party protocols[1] in the computational setting compared to the huge literature on two-party protocols.

In this paper we aim to remedy this state-of-affairs by providing a formal reductionist analysis of EAP in the computational setting. Our result is modular and reflects the compositional nature of EAP. Building upon this result we extend our analysis to cover a very common application of the EAP framework: network authentication and access control in enterprise and university networks. In particular, we focus on wireless networks based on the IEEE 802.11 standard [2] when combined with EAP for centralized authentication. This setting is often referred to as WPA2-Enterprise. Current results on IEEE 802.11 have so far only focused on the much simpler WPA2-PSK setting where the client and access point (authenticator) already share a pre-established long-term key. WPA2-PSK is typically used in wireless home-networks and small offices where sharing a single long-term key among many users is feasible, while WPA2-Enterprise is used in larger organizations and businesses where central authentication is necessary. Based on our result on EAP we can now provide a

[1] Considered distinct from *group-key exchange* protocols.

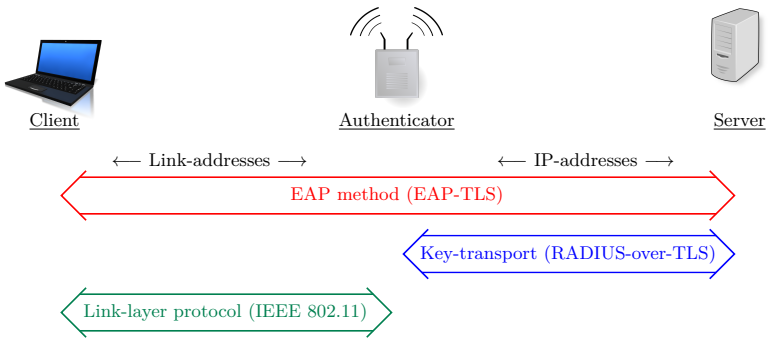

Figure 1: The three-party EAP architecture. Concrete example protocols shown in parenthesis.

reduction-based analysis of WPA2-Enterprise as well.

Review of EAP and IEEE 802.11. The general EAP architecture is shown in Fig. 1. The exchange begins when a client wants to connect to some access-controlled service protected by an authenticator. For most practical purposes the service is simply getting network access (e.g., to the Internet), and the authenticator is a wireless access point or link-layer switch. The client and authenticator do not share any common secrets a-priori but instead share credentials with a mutually trusted server. The purpose of EAP is to allow the client to authenticate itself to the server using whatever authentication protocol they like, for instance TLS or IPsec, and then having the server "vouch" for the client to the authenticator. In order to do this in a generic and uniform manner across different authentication protocols, EAP defines a frame format as well as a set of generic messages known as Request/Response messages. The Request and Response messages are used to encapsulate the concrete authentication protocol being used between the client and the server. This frees the authenticator—which often operates in so-called *pass-through* mode between the client and the server, meaning that all their messages pass through it—from having to support all the concrete authentication mechanisms itself. Instead, the authenticator only needs to inspect the generic EAP messages. This is valuable in settings where it can be difficult to update the authenticator(s).

The combination of a concrete authentication protocol, like TLS, together with the EAP encapsulation is called an *EAP method*. Numerous EAP methods have been defined, with EAP-TLS [44] being one of the most widely supported. In EAP-TLS the client and server mutually authenticate each other based on certificates. Besides authentication, the EAP method usually also supports the derivation of a shared key between the client and the server. In this paper we will assume that all EAP methods derive keys. The server will forward this key to the authenticator over some separate channel, where the choice of channel protocol is orthogonal to the choice of EAP method used between the client

and the server. While the EAP standard does not specify the protocol to use between the server and the authenticator, the de-facto standard in practice is RADIUS [39].[2]

Once the key is transported from the server to the authenticator the EAP exchange is technically complete. Still, at this point the client does not actually have any guarantee that the authenticator is in possession of the same key as itself, since the communication between the server and the authenticator happens over a completely separate channel. Thus, in practice the client and the authenticator now typically run some link-layer specific protocol in order to prove mutual possession of the key distributed by EAP. Additionally, this also serves to implicitly authenticate the client and the authenticator to each other, since in order to get the same key they must have been able to authenticate themselves to the mutually trusted server.

Again, the subsequent link-layer protocol run between the client and the authenticator is outside the scope of EAP and could in principal be any one of a number of different protocols. In this paper we are going to focus particularly on the a very common setting of wireless LANs provided by the IEEE 802.11 protocol[2]. In this case the "key-confirmation" protocol run between the client and the authenticator (i.e., access point) is known to as the "4-Way-Handshake" (4WHS). The 4WHS is both an authentication protocol as well as a key-exchange protocol, meaning that the client and the access point also derive fresh session keys. This session key is used to protect the bulk data transfer between the client and the access point on the WLAN. Although technically incorrect, the security part of the IEEE 802.11 wireless standard is also commonly referred to by the name "WPA2". When EAP and IEEE 802.11 are combined, then the entire exchange is referred to as "IEEE 802.11 with upper-layer authentication" or WPA2-Enterprise.

On the difficulty of modeling EAP. In this paper we analyze the security of EAP both when considered on its own as well as when combined with IEEE 802.11 in a formal reduction-based setting. We do this in a modular way: first considering the security properties provided by EAP and IEEE 802.11 in isolation, then using a composition theorem to link them together. However, since EAP inherently depends on other protocols, assessing the exact security guarantees it provides is in one sense harder than for "standalone" protocols like TLS, IKE and SSH. While the EAP specification defines the security requirements of each EAP method ([4, §7]), this only covers the communication between the client and the server. It leaves unspecified how, for example, the derived key should be transferred from the server to the authenticator. Hence, solely using the security claims from RFC 3748 is not sufficient to decide the security of EAP considered as a three-party protocol. In fact, it is impossible to talk about "the" EAP and its security without making further assumptions

[2]Within the EAP standard lingo, the protocol run between the server and authenticator is generally referred to as an *Authentication, Authorization and Accounting (AAA)* protocol. Besides RADIUS is Diameter [18] another common AAA protocol.

on the various protocols that make up EAP. Consequently, in order to be able to analyze EAP, we will have to make some assumptions on these protocols.

Firstly, in this paper we are going to assume that the communication between the authenticator and the server takes place over a secure channel. Specifically, we model the link as a two-party authenticated channel establishment protocol (2P-ACCE) based on symmetric long-term pre-shared keys[3] (see Section 2.4 for a formal definition). Since most key-transport protocols used between the server and the authenticator support to be run inside a secure channel (see e.g. RADIUS-over-TLS [45] and Diameter [18]), this assumption seems reasonable.

Second, since the authenticator often works in pass-through mode, a well-known issue with the EAP architecture is the so-called "lying authenticator problem". Namely, a malicious authenticator may present false or inconsistent identity information to the client and the server. Unless the EAP method provides a feature known as *channel binding* [21], there is no way for the client and server to verify that they are in fact talking to the same authenticator (see [21, §3] for examples of attacks that this may enable). Hence, in this paper we are generally going to assume that EAP provides channel binding. However, we will also briefly explore the (in)security of EAP without channel binding in Section 4.3. While there are a couple of suggested ways to achieve channel binding in EAP (see [21, §4.1]), here we are only going to focus on the cryptographically simplest one, described in RFC draft `draft-ohba-eap-channel-binding-02` [38]. In this approach, the client and authenticator identities are being input to the key-derivation step of EAP, cryptographically binding the session key to the right pair of identities (see Section 4.2 for details).

Our contributions. The main contributions of this paper are the following.

- We provide the first reduction-based security result for EAP assuming it employs channel binding.
- We show how the security guarantees of EAP can be upgraded by adding an additional key-confirmation step (modeled as a 2P-AKE). This corresponds to the common scenario where EAP is first used to bootstrap the establishment of a common key among the client and the authenticator, then some link-layer specific protocol is run between the client and the authenticator in order to prove mutual possession of that key (in addition to establishing session keys for the lower-layer link).
- Our technical means for obtaining the above results are two modular composition theorems which may be of separate interest. Namely, the two theorems consider a fairly generic way of constructing a 3P-AKE protocol, using generic 2P-AKEs and secure channels as building blocks. For

[3]There is nothing fundamental about our assumption on symmetric PSKs here. The choice is made simply because the trust-relationship between the server and authenticator is commonly based on symmetric PSKs in practice. Our results work just as well for certificate-based authentication.

instance, both Kerberos and the AKA protocol used within the UMTS and LTE mobile networks, fit the description of our 3P-AKE construction. In particular, for the latter protocol, our theorems might enable a more general and modular analysis than the one recently provided by Alt et al. [5].

- As a stepping stone towards our final result, we provide a reduction-based security result for the IEEE 802.11 4-Way-Handshake protocol in the pre-shared key setting without the use of EAP (i.e., WPA2-PSK). This corresponds to the setting typically found in home WLANs. To the best of our knowledge, we are the first do to such an analysis using standard game-based definitions of AKE. Previous analyses of WPA2-PSK have either been based on formal methods [22] or on universal composition frameworks [32, 33].

- Finally, the results above combine to provide the first reduction-based security result for EAP combined with IEEE 802.11 (WPA2-Enterprise). This corresponds to the setting usually found in enterprise and university WLANs. For instance, the *eduroam* network[4], which is used to provide wireless roaming services to university and research institutions, uses EAP and IEEE 802.11.

The structure of our paper is as follows. In Section 2 we provide our formal security definitions, including 2P/3P-AKE, ACCE (secure channels) and explicit entity authentication. In Section 3 we prove our two composition results for two generic protocol constructions. In Section 4 we show how the security of standalone EAP follows directly from our first composition result by making appropriate assumptions on the concrete protocols used to instantiate the EAP framework. Finally, in Section 5, we prove the security of the IEEE 802.11 4WHS protocol. Combined with our result on EAP in Section 4 and our second composition result, this immediately yields a result for IEEE 802.11 combined with EAP.

Technical overview of our results. The main technical contributions of this paper are two fairly generic composition theorems which correspond to the "cryptographic core" of EAP with or without a subsequent key-confirmation step, respectively. To obtain these theorems we first have to provide an appropriate security model. Our starting point is the original 3P-AKE model of Bellare and Rogaway [10]. However, due the different security guarantees provided by standalone EAP, EAP combined with IEEE 802.11 and standalone IEEE 802.11, we in fact define *three* different models of varying strength. The main difference between these models lays in the level of adaptivity afforded to the adversary in terms of long-term key leakage, capturing full, weak and no forward secrecy, respectively. The distinction between full and weak forward secrecy follows the definition given in the eCK model[5] [34].

[4] `https://www.eduroam.org`

[5] We do not consider ephemeral key-leakage in this paper however.

Briefly, the only difference between the strongest security model (full forward secrecy) and the intermediate one (weak forward secrecy) depends on what happens if the test-session does not have a partner. When this happens in the strongest model the adversary is still allowed to learn the long-term keys of the parties involved, provided this happens after the test-session accepted. On the other hand, in the intermediate model, if the test-session does not have a partner then the adversary is forbidden from learning these long-term keys. Finally, if the test-session *does* have a partner, then there is no difference between the two models: the adversary is allowed to learn any long-term key at any time. The formal definitions of these models are provided in Section 2.3.

Preempting our own results a bit, we show that standalone EAP can achieve weak forward secrecy, while IEEE 802.11 *without* upper-layer authentication achieves no forward secrecy at all (this is natural since the 4WHS relies exclusively on symmetric primitives). However, when EAP and IEEE 802.11 are *combined*, the security is upgraded to achieve full forward secrecy in our strongest corruption model.

Intuitively, the reason why standalone EAP does not achieve full forward secrecy is because it does not provide *key confirmation*. Namely, after completing the EAP method with the server, the client has no guarantee that the key-transport protocol between the server and the authenticator actually took place. Specifically, the following attack illustrates why EAP does not provide full forward secrecy. Suppose that *after* the client accepted, but *before* the key-transport protocol between the server and authenticator starts running, an adversary learns the long-term PSK of the server and the authenticator. Now the adversary simply impersonates the authenticator towards the server and have it send over the session key it previously established with the client. According to the full forward secrecy model this attack would be valid since the exposure of the PSK happened after the client accepted. On the other hand, in the weak forward secrecy model the attack is not considered valid because client session doesn't have a partner, hence the PSK cannot be exposed.

Essentially, the purpose of the link-layer protocol is to provide key-confirmation to the standalone EAP protocol, which ensures that the client will always have a partner before it accepts. This is similar to how the security of the two-flow variant of HMQV can be upgraded from only providing weak forward secrecy to providing full forward secrecy simply by adding a third flow to it (see [28, §3]). While the property of key confirmation was recently formalized in [19], in this paper we model the key-confirmation step by assuming that the link-layer protocol provides *explicit* entity authentication (formally defined in Section 2.5).

Besides the introduction of the three different corruption models, we only provide a few other changes to the original 3P-AKE model of Bellare and Rogaway [10]. For example, we support both asymmetric and symmetric long-term keys, and dispense with the explicit SendS query to the server (now modeled simply as a regular Send query).

One thing we *do* keep from [10] however, is the concept of *partner functions*. Interestingly, the use of partner functions has seen rather limited adoption when

compared to partnering based on matching conversations [9] or session identifiers (SIDs) [8]. However, when modeling EAP, we are in the peculiar situation that the parties that we need to partner (the client and the authenticator) do not have any messages in common! Naturally, this makes partnering based on matching conversations more difficult, but it also severely limits our choice of SIDs: we are essentially forced to pick their session keys as the SID. While using the session key as the SID is reasonable in many settings (cf. [25]), it does not necessarily guarantee *public* partnering (see [13]). This is important for modular composition proofs like our own. While partnering functions have been criticized for being non-intuitive and hard to work with (even by Rogaway himself [40, §6]), they generalize more naturally to the three-party setting than SIDs. Essentially, this is because partner functions can take global transcript information into consideration rather than only the local views of the two partners.

2 Formal models

2.1 Notation

For $m, n \in \mathbb{N}$ and $m \leq n$, let $[m, n] \stackrel{\text{def}}{=} \{m, m+1, \ldots n\}$. We use $v \leftarrow x$ to denote the assignment of x to the variable v, and $x \twoheadleftarrow X$ to denote that x is assigned a value randomly according to the distribution X. If S is a finite set, then $x \twoheadleftarrow S$ means to sample x uniformly at random from S. Algorithms are in general randomized, and we let $y \twoheadleftarrow A(x_1, \ldots, x_n)$ denote running A on inputs $x_1, \ldots, x_n$ with random coins r, and assigning its output to the variable y. A function $g\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if for every $c \in \mathbb{N}$ there is an integer n_c such that $g(n) \leq n^{-c}$ for all $n \geq n_c$.

2.2 A unified execution model

Protocol participants. A protocol is carried out by a set of *parties* $U \in \mathcal{P}$. Each party U can either take on the role of *initiator*, *responder* or *server*, i.e., $\mathcal{P}$ is partitioned into three disjoint sets $\mathcal{I}$, $\mathcal{R}$ and $\mathcal{S}$, consisting of the initiators, responders and servers, respectively. In the two-party case there are no servers, in which case $\mathcal{S} = \emptyset$.

Our model includes both long-term asymmetric keys as well as a symmetric pre-shared keys (PSKs). While there are in general many ways in which asymmetric and symmetric long-term keys could be combined in a three-party protocol, in this paper we are going to limit ourselves to the configuration typically found in EAP. That is, we assume that only initiators and servers are in possession of a long-term private/public key-pair, while all responders and servers share a symmetric PSK. On the other hand, for two-party protocols we assume that the long-term keys are either strictly based on asymmetric keys or strictly based on PSKs. For every U party holding a public key pk_U, we assume that all other parties have an authenticated copy of it.

Syntax. A *protocol* is a tuple $\Pi = (\mathsf{KG}, \Sigma)$ of probabilistic polynomial-time algorithms, where KG specifies how long-term keys are generated for each party, and Σ specifies how (honest) parties behave. Each party $U \in \mathcal{P}$ can take part in multiple executions of the protocol, both concurrently and subsequently, called a *session*. We use an administrative label π_U^i to refer to the ith session at user U. This will sometimes be simplified to π. Associated to each session π_U^i, there is a collection of variables that embodies the (local) state of π_U^i during the run of the protocol.

- sk_U, pk_U – the (possibly empty) long-term private/public key of party U,
- peers – a list of the identities of the intended communication peers of π_U^i,
- $\mathsf{PK}[\cdot]$ – a map taking party identities to (possibly empty) public keys for each $V \in \pi_U^i.\mathsf{peers}$, i.e, $\mathsf{PK}[V] \mapsto pk_V/\perp$,
- $\mathsf{PSK}[\cdot]$ – a map taking party identities to (possibly empty) PSKs for each $V \in \pi_U^i.\mathsf{peers}$, i.e, $\mathsf{PSK}[V] \mapsto K_{UV}/\perp$,
- $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ – a vector of *accept states* $\alpha_i \in \{\mathsf{running}, \mathsf{accepted}, \mathsf{rejected}\}$,
- $k \in \{0,1\}^\lambda \cup \{\perp\}$ – the symmetric session-key derived by π_U^i.

Only initiators and responders accept sessions keys, i.e., if $S \in \mathcal{S}$, then we always have $\pi_S^i.k = \perp$. Note that this is pure formalism: we certainly expect many protocols in which the server might be in possession of the session key—in fact, the server might be the one that choses and distributes it—we simply do no not associate it with the variable k.

Remark 1. We use a *list* of acceptance states $\vec{\alpha}$ rather than a *single* acceptance state more commonly found in other formal protocol models. We do this in order to model protocols that are logically built out of sub-protocols. The individual acceptance states α_i provides a convenient way to signal to the adversary that a session has accepted in some intermediate sub-protocol Π_i of the full protocol Π. By convention, we will let α_n represent the acceptance state of the full protocol, and use $\alpha_F \stackrel{\text{def}}{=} \alpha_n$ to denote this state. A session is said to be *running*, *accepted* or *rejected*, based on the value of α_F. Thus, α_F has the same role as the single acceptance state variable α used in other protocol models.

We require the following semantics of the variables $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ and k:

$$\alpha_i = \mathsf{accepted} \implies \alpha_{i-1} = \mathsf{accepted}, \tag{1}$$

$$\alpha_i = \mathsf{rejected} \implies \alpha_{i+1} = \mathsf{rejected}, \tag{2}$$

$$\pi.\alpha_n = \mathsf{accepted} \implies \pi.k \neq \perp\ . \tag{3}$$

By convention, whenever we set $\alpha_i = \mathsf{rejected}$, we also automatically set $\alpha_j = \mathsf{rejected}$ for all $i < j$, in accordance with (2). Moreover, we assume that the session key $\pi.k$ is set only once.

$\underline{\mathbf{Exp}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda)}$:

Long-term key set-up:
  3P: For every $U \in \mathcal{I} \cup \mathcal{S}$ create $(sk_U, pk_U) \twoheadleftarrow \Pi.\mathsf{KG}(1^\lambda)$
    For every $(U, V) \in \mathcal{R} \times \mathcal{S}$ define $K_{UV} = K_{VU} \twoheadleftarrow \{0,1\}^\lambda$
    Define $\mathsf{pks} \leftarrow \{(U, pk_U) \mid U \in \mathcal{I} \cup \mathcal{S}\}$

  2P-Public-Key: for every $U \in \mathcal{I} \cup \mathcal{R}$ create $(sk_U, pk_U) \twoheadleftarrow \Pi.\mathsf{KG}(1^\lambda)$
    Define $\mathsf{pks} \leftarrow \{(U, pk_U) \mid U \in \mathcal{I} \cup \mathcal{R}\}$

  2P-PSK: For every $(U, V) \in \mathcal{I} \times \mathcal{R}$ define $K_{UV} = K_{VU} \twoheadleftarrow \{0,1\}^\lambda$
    Define $\mathsf{pks} \leftarrow \emptyset$

$out \twoheadleftarrow \mathcal{A}^{\mathcal{Q}}(1^\lambda, \mathsf{pks})$

Figure 2: Unified experiment used to simultaneously define AKE and ACCE security, including three-party and two-party settings as well as protocols using asymmetric and symmetric long-term keys.

Protocol correctness. It is required that an AKE protocol satisfies the following correctness requirement. In an honest execution of the protocol between an initiator π_A^i, a responder π_B^j and a server π_S^k (if in the three-party setting)—meaning that all messages are faithfully transmitted between them according the protocol description—then all sessions end up accepting with the correct intended peers, and π_A^i and π_B^j both hold the same session key $k \neq \perp$.

A unified security experiment. To define the security goals of both AKE and ACCE protocols we use the unified experiment shown in Fig. 2. Experiment $\mathbf{Exp}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda)$ is parameterized on the protocol Π, a *query set* $\mathcal{Q}$, and the adversary $\mathcal{A}$. While the query sets used to define AKE and ACCE security will be different, they will both contain the following "base" query set $\mathcal{Q}_{base}$:

- $\mathsf{NewSession}(U, [V, W])$: This query creates a new session π_U^i at party U, optionally specifying its intended communication peers V and W. It is required that U, V and W all have different roles.

  The variables k and $\vec{\alpha}$ are initialized to $\pi_U^i.k = \perp$ and $\pi_U^i.\vec{\alpha} = (\mathsf{running}, \ldots, \mathsf{running})$, respectively. Additionally, depending on the type of protocol (two-party/three-party, symmetric/asymmetric long-term keys), as well as the roles of U, V and/or W, the variables sk, pk, peers, PK and PSK are initialized accordingly.

  Finally, if $U \in \mathcal{I}$, then π_U^i also produces its first message m according to the specification of protocol Π. Both the administrative label π_U^i and m are returned to $\mathcal{A}$.

- $\mathsf{Send}(\pi_U^i, m)$: If $\pi_U^i.\alpha_F \neq \mathsf{running}$, return $\perp$. Otherwise, π_U^i creates a

response message m^* according to the specification of protocol Π. This depends on π_U^i's role and current internal state. Both m^* and $\pi_U^i.\vec{\alpha}$ are returned to $\mathcal{A}$.

- $\mathsf{Reveal}(\pi_U^i)$: If $\pi.\alpha_F \neq \mathsf{accepted}$ or $U \in \mathcal{S}$, return $\perp$. Else, return $\pi_U^i.k$. From this point on π_U^i is said to be *revealed*. Note that π_U^i is *not* considered revealed if the Reveal query was made before π_U^i accepted.

- $\mathsf{LongTermKeyReveal}(U, [V])$: Depending on the second input parameter, this query returns a certain long-term key of party U.

  - $\mathsf{LongTermKeyReveal}(U)$: If U has an associated private-public key-pair (sk_U, pk_U), return the private key sk_U.
  - $\mathsf{LongTermKeyReveal}(U, V)$: If U and V share a symmetric long-term key K_{UV}, return K_{UV}.

  After a long-term key has been leaked we say that it is *exposed* and the corresponding party *corrupted*.

Remark 2. We are working in the post-specified peer model [15], meaning that the identities of a session's peers might not necessarily be known by the session at the beginning of the protocol run, but are rather learned as the protocol progresses.

Note that experiment $\mathbf{Exp}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda)$ does not provide any output and does not define any "winning condition" for $\mathcal{A}$. Instead, it provides a single execution experiment on which we can define many different winning conditions. This is convenient since it allows an easy way of specifying the many different security models needed in this paper in a uniform manner. Common for all of the security models will be the notion of *freshness* which decides whether $\mathcal{A}$ has managed to satisfy a winning condition in a non-trivial way. Our definition(s) of freshness further depends on the notion of *partnering*, defined next. Partnering is used to formalize the intuition that for any session π that ends up holding a session key, there will (possibly) be some *other* session π' whose loss of session key will also compromise that of π.

Transcripts and partner functions. To define partnering in our security models we will use the concept of *partner functions* as introduced by Bellare and Rogaway [10]. However, to simplify our later analysis, we will limit ourselves only to *symmetric* and *monotonic* partner functions. Basically, a partner function is symmetric if it is its own inverse (up to $\perp$), and monotonic if it never "changes its mind", i.e., once two sessions become partners they remain so forever. Bellare and Rogaway did not demand these properties directly in their original definition, but instead claimed that they could be inferred from the definition (see [10, §6]). We find it easier to require these properties at the definitional level.

To formally define partner functions, we first the need the notion of a protocol *transcript*, which is essentially records the public communication of a protocol run. More precisely, consider a run of experiment $\mathbf{Exp}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda)$. Let T be the ordered transcript consisting of all the Send and NewSession queries made by $\mathcal{A}$, together with their responses. A transcript T is a *prefix* of another transcript T', written $T \subseteq T'$, if the first $|T|$ entries of T' are identical to T. Let $\mathcal{T}$ denote the set of all possible transcripts generated from running experiment $\mathbf{Exp}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda)$. We can now define partner functions.

Definition 1 (Partner functions)**.** A *symmetric and monotonic partner function* is a polynomial-time function $f\colon \mathcal{T} \times (\mathcal{P} \setminus \mathcal{S}) \times \mathbb{N} \to ((\mathcal{P} \setminus \mathcal{S}) \times \mathbb{N}) \cup \{\perp\}$, subject to the following requirements:

1. $f(T,U,i) = (V,j) \implies f(T,V,j) = (U,i)$, (symmetry)
2. $f(T,U,i) = (V,j) \implies f(T',U,i) = (V,j)$ for all $T \subseteq T'$. (monotonicity)

Instead of $f(T,U,i) = (V,j)$, we also write $f_T(\pi_U^i) = \pi_V^j$, or even just $f_T(\pi) = \pi'$ if the exact identities of the sessions are irrelevant.

Since all partner functions in this paper are required to be both symmetric and monotonic, we drop these qualifiers from now on and simply talk about "partner functions". Note that both requirements are straightforwardly met by partner functions based on SIDs.

Definition 2 (Partnering)**.** Let f be a partner function. A session π' is a *partner* to π if $f_T(\pi) = \pi'$.

Of course, by the symmetry requirement above, if π' is the partner to π, then π will also be a partner to π'. Hence, we can simply talk about π and π' being *partners*.

Remark 3. Partnering is only defined between initiators and responders. Servers are not considered partners to any session.

Remark 4. The use of partner functions to analyze key exchange protocols is rare in the literature. To the best of our knowledge, besides the original paper by Bellare and Rogaway [10], it has only been used in one other paper by Shoup and Rubin [43]. In a currently unpublished manuscript [12], we provide a more detailed treatment of partner functions in general.

Partnering soundness. For a security analysis based on partner functions to be meaningful, the partner function needs to satisfy certain soundness properties. Briefly, soundness demands that partners should: (1) end up with the same session key, (2) agree upon who they are talking to, (3) have compatible roles, and (4) be unique. However, since we are limiting our attention to symmetric partner functions in this paper, the last requirement follows directly so we omit it.

Definition 3 (Partner function soundness)**.** A partner function is *sound* if the following holds for all transcripts T. If sessions $f_{T'}(\pi_U^i) = \pi_V^j$ then:

1. $\pi_U^i.\alpha_F = \pi_V^j.\alpha_F = \mathsf{accepted} \implies \pi_U^i.k = \pi_V^j.k \neq \perp$,
2. $\pi_U^i.\mathsf{peers} = \{V, W\}$, $\pi_V^j.\mathsf{peers} = \{U, W\}$, and $W \in \mathcal{S}$,
3. $U \in \mathcal{I} \wedge V \in \mathcal{R}$ or $U \in \mathcal{R} \wedge V \in \mathcal{I}$,

Soundness is essentially the partner function equivalent of the Match-security notion introduced by Brzuska et al. [13], used for partnering based on SIDs. However, unlike Match-security, we demand that properties (1)–(3) hold unconditionally instead of only with overwhelming probability. We note that this requirement is not fundamental, and only used to simplify our later analysis.

2.3 2P-AKE and 3P-AKE

Syntax. A *2P/3P-AKE protocol* has the same syntax as the general protocol defined in Section 2.2. Note that is also no syntactical difference between a 2P-AKE protocol and a 3P-AKE protocol, apart from the fact that the former has no server session $S \in \mathcal{S}$. Consequently, in the two-party case the session variables peers, PK and PSK contain at most a single entry.

AKE security. A secure AKE protocol is supposed to provide secrecy of the distributed session keys. To capture this, the base query set $\mathcal{Q}_{base}$ is extended with the following query.

- $\mathsf{Test}(\pi_U^i)$: If $\pi_U^i.\alpha_F \neq \mathsf{accepted}$ or $U \in \mathcal{S}$, return $\perp$. Otherwise, draw a random bit b, and return π_U^i's session key if $b = 0$, or a random key if $b = 1$. We call π_U^i the *test-session* and the returned key the *test-key*. The Test query can only be made once.

Let $\mathcal{Q}_{\mathsf{AKE}} = \mathcal{Q}_{base} \cup \{\mathsf{Test}\}$. Experiment $\mathbf{Exp}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda)$ stops when $\mathcal{A}$ outputs a bit b'. The goal of the adversary is to correctly guess the secret bit b used to answer the Test query. However, $\mathcal{A}$ is only given "credit" if the chosen test-session was *fresh*. A session is fresh if the adversary did not learn its session key by trivial means, for example by revealing it or by impersonating its peers after having obtained their long-term keys etc. Formally, in Fig. 3, we specify three *freshness predicates* $\mathsf{Fresh}_{\mathsf{AKE}}$, $\mathsf{Fresh}_{\mathsf{AKE}^w}$, and $\mathsf{Fresh}_{\mathsf{AKE}^{\mathsf{static}}}$, of various permissiveness with respect to long-term key leakage. Each freshness predicate gives rise to a corresponding security model, denoted AKE, AKE^w and AKE$^{\mathsf{static}}$ respectively. We describe the three models in more detail below and summarize their main differences in Table 1.

AKE with forward secrecy: the AKE and AKE^w models. The AKE model is our "partner function analogue" of the standard eCK model (as defined in the updated version [34] of the original paper [35]), with the main difference being that we do not consider leakage of ephemeral values. In particular, the AKE model captures both key-compromise impersonation (KCI) attacks and forward secrecy. KCI attacks are captured since the test-session's own long-term private

$\mathsf{Fresh}_M(\pi_U^i)$:

1: $\mathsf{fresh} \leftarrow \mathtt{true}$
2: $\{V, W\} \leftarrow \pi_U^i.\mathsf{peers}$
3: $\mathsf{LTKeys} \leftarrow \{\pi_U^i.\mathsf{PK}[V], \pi_U^i.\mathsf{PK}[W], \pi_U^i.\mathsf{PSK}[V], \pi_U^i.\mathsf{PSK}[W], K_{VW}\}$
4:
5: $\mathsf{fresh} \leftarrow \mathsf{fresh} \wedge (\pi_U^i.\alpha_F = \mathsf{accepted})$
6: $\mathsf{fresh} \leftarrow \mathsf{fresh} \wedge (\pi_U^i$ not revealed$)$
7: $\mathsf{fresh} \leftarrow \mathsf{fresh} \wedge (f_T(\pi_U^i)$ not revealed$)$
8: $\mathsf{fresh} \leftarrow \mathsf{fresh} \wedge (\mathsf{corrupt}_M = \mathtt{false})$
9:
10: **return** fresh

- $\mathsf{corrupt}_{\mathsf{AKE/ACCE}} = \mathtt{true} \iff (f_T(\pi_U^i) = \perp) \wedge$ (a key in LTKeys was exposed before π_U^i accepted)
- $\mathsf{corrupt}_{\mathsf{AKE}^w} = \mathtt{true} \iff (f_T(\pi_U^i) = \perp) \wedge$ (a key in LTKeys is exposed)
- $\mathsf{corrupt}_{\mathsf{AKE}^{\mathsf{static}}} = \mathtt{true} \iff$ a key in LTKeys is exposed

Figure 3: Freshness predicate for security model $M \in \{\text{AKE}, \text{AKE}^w, \text{AKE}^{\mathsf{static}}, \text{ACCE}\}$. Some of the keys in LTKeys might be undefined, e.g., if $W \in \mathcal{S}$, then $\pi_U^i.\mathsf{PK}[W]$, $\pi_U^i.\mathsf{PK}[W]$ and K_{VW} are undefined in the two-party case, and $\pi_U^i.\mathsf{PK}[V]$ is undefined if V is a responder party in the three-party case. Undefined keys are ignored.

key can always be exposed by the adversary. Forward secrecy is captured since the adversary can additionally learn the long-term keys of the peers of the test-session after it accepted.

The forward secrecy guarantees provided by the AKE model are rather strong: if a session has a partner, then the adversary is allowed to expose *any* long-term key it wants, while if the session does not have a partner, then the adversary must wait until after the session accepted before it can expose the relevant keys. Note that partnering is used to model *passiveness* by the adversary in the test-session. Intuitively, even if the adversary knew all the long-term keys before the test-session started, if the test-session ends up with a partner, then the adversary cannot actually have exploited its knowledge of the keys.

Compared to the AKE model, the AKE^w model is more restrictive with respect to forward secrecy: if the test-session does not have partner, then the adversary is disallowed from exposing any of the relevant long-term keys. The AKE^w model is similar to the two-pass variant of the eCK model (see [34, Def. 3]). As mentioned in the introduction, standalone EAP does not achieve security in the AKE model, but we will show that it *is* secure in the AKE^w model.

AKE without forward secrecy: the $\text{AKE}^{\mathsf{static}}$ model. To accommodate protocols that does not provide forward secrecy we introduce the $\text{AKE}^{\mathsf{static}}$ model. Unlike the AKE and AKE^w models, the $\text{AKE}^{\mathsf{static}}$ model disallows the adversary

Table 1: Summary of the three AKE security models in terms of the amount of corruption allowable by the adversary (i.e., long-term key reveals). The table assumes π_A^i is the test-session having peers B and S (in the three-party case).

		Corrupt B or S	
Model	Corrupt A	if π_A^i has a partner	if π_A^i has no partner
AKE	allowed	allowed	allowed[1]
AKE^w	allowed	allowed	$\times$
AKE$^{\mathsf{static}}$	$\times$	$\times$	$\times$

[1] Only after π_A^i accepted.

from exposing the long-term keys altogether, no matter whether a session has a partner or not (of course, the adversary is allowed to expose long-term keys unrelated to the test-session and its peers).

On the other hand, for technical reasons (see the explanation following Lemma 9 in Section 3.2), we slightly strengthen the AKE$^{\mathsf{static}}$ model compared to the AKE and AKE^w-models along a different axis. Namely, we give the adversary the capability of *key registration*. That is, when creating a new session, the adversary is allowed to (optionally) specify the long-term key(s) that the session will use in its protocol run. Of course, any session for which the adversary supplies the long-term keys will be considered unfresh, so the key registration capability does not substantially strengthen the model.

Technically, key registration in the AKE$^{\mathsf{static}}$ model is handled by modifying the NewSession query. Furthermore, since we are only going to use the AKE$^{\mathsf{static}}$ model to analyze PSK-based two-party protocols in this paper, we specialize the definition to this specific case:

- NewSession$(U, [V], [\widehat{K}])$: This query works exactly like the NewSession query defined in Section 2.2, except that if the adversary supplies an optional long-term key $\widehat{K}$, then the newly created session π_U^i stores $\widehat{K}$ in $\pi_U^i.\mathsf{PSK}[V]$ rather than K_{UV}. In this case $\pi_U^i.\mathsf{PSK}[V]$ is considered *exposed*.

If the adversary makes a NewSesssion query where it provides a long-term key, then the key is nevertheless omitted from the NewSession query that gets added to the protocol transcript T. Thus, the protocol transcripts generated from the AKE$^{\mathsf{static}}$ model are syntactically the same as those generated from the AKE and AKE^w models, even if the latter does not include key registration.

Security definitions. Let $\mathcal{Q}_{\mathsf{AKE}}$ denote the query set either used in the AKE or AKE^w models (having NewSession queries without key registration), or in the AKE$^{\mathsf{static}}$ model (having NewSession queries with key registration).

Definition 4 (AKE winning event)**.** Suppose π was the test-session chosen by $\mathcal{A}$ in a run of experiment $\mathbf{Exp}_{\Pi,\mathcal{Q}_{\mathsf{AKE}},\mathcal{A}}(\lambda)$, b was the random bit used in answering the Test query, and suppose b' was the final output of $\mathcal{A}$. Fix a partner function f and let $\mathsf{AKE}^* \in \{\mathsf{AKE}, \mathsf{AKE}^w, \mathsf{AKE}^{\mathsf{static}}\}$ be the following random variable defined on experiment $\mathbf{Exp}_{\Pi,\mathcal{Q}_{\mathsf{AKE}},\mathcal{A}}(\lambda)$:

$$\mathsf{AKE}^* \stackrel{\text{def}}{=} \begin{cases} (b = b'), & \text{if } \mathsf{Fresh}^*_{\mathsf{AKE}}(\pi) = \mathtt{true} \\ \mathtt{true} \text{ with probability } 1/2, & \text{if } \mathsf{Fresh}^*_{\mathsf{AKE}}(\pi) = \mathtt{false} \end{cases} \tag{4}$$

Let $\mathbf{Exp}^{\mathsf{AKE}^*}_{\Pi,\mathcal{Q}_{\mathsf{AKE}},\mathcal{A}}(\lambda) \Rightarrow 1$ denote the event that $\mathsf{AKE}^* = \mathtt{true}$.

Definition 5 (AKE security)**.** For $\mathrm{AKE}^* \in \{\mathrm{AKE}, \mathrm{AKE}^w, \mathrm{AKE}^{\mathsf{static}}\}$ and some partner function f, define the *AKE*-advantage* of adversary $\mathcal{A}$ to be

$$\mathbf{Adv}^{\mathsf{AKE}^*}_{\Pi,\mathcal{A},f}(\lambda) \stackrel{\text{def}}{=} 2 \cdot \Pr[\mathbf{Exp}^{\mathsf{AKE}^*}_{\Pi,\mathcal{Q}_{\mathsf{AKE}},\mathcal{A}}(\lambda) \Rightarrow 1] - 1 \tag{5}$$

A protocol Π is *AKE*-secure*, if there exists a sound partner function f, such that for all PPT adversaries $\mathcal{A}$, its advantage $\mathbf{Adv}^{\mathsf{AKE}^*}_{\Pi,\mathcal{A},f}(\lambda)$ is negligible in security parameter λ.

If we want to emphasize that a protocol is two-party or three-party, we write $\mathbf{Adv}^{\mathsf{2P\text{-}AKE}^*}_{\Pi,\mathcal{A},f}(\lambda)$ or $\mathbf{Adv}^{\mathsf{3P\text{-}AKE}^*}_{\Pi,\mathcal{A},f}(\lambda)$, respectively.

Remark 5. Note that in our formulation of security we are quantifying over *all* PPT adversaries, not only those that satisfy the freshness predicate. Instead, if the adversary violates the freshness predicate, we "penalize" it in the winning condition (Def. 4) by having the challenger output a random bit on its behalf. This *penalty-style* of formulating security has previously been used in other papers like e.g., [7] and [20].

2.4 ACCE

In this section we define *authenticated and confidential channel establishment (ACCE)* protocols. Intuitively, an ACCE protocol combines an ordinary 2P-AKE protocol with a *stateful authenticated encryption (sAE) scheme*, where the session keys of the 2P-AKE protocol are used to key the sAE scheme.

Syntax. An ACCE protocol is a two-party-protocol as defined in Section 2.2, together with an associated sAE scheme $\mathsf{stE} = (\mathsf{stE.Init}, \mathsf{stE.Enc}, \mathsf{stE.Dec})$ (following [24][6]). The notion of a session is the same as before, but the (local) state is extended with two additional variables st_{E} and st_{D} in order to store the encryption/decryption state of the sAE scheme.

Correctness of the sAE scheme demands that if the *deterministic* algorithm $\mathsf{st.Init}$ produced initial states st^0_E, st^0_D; and the ACCE session key k was used to produce a sequence of ciphertext/state pairs $(C_i, st^{i+1}_E) \leftarrow \mathsf{stE.Enc}(k, m_i, st^i_E)$ such that $C_i \neq \perp$ for all $i \geq 0$; then one must have, for all $i \geq 0$, that $m'_i = m_i$ in the sequence of decryptions $(m'_i, st^{i+1}_D) \leftarrow \mathsf{stE.Dec}(k, C_i, st^i_D)$.

[6] For simplicity, we omit the properties of *length-hiding* and *associated data* in our treatment of ACCE. This omission is immaterial for the results established in this paper.

$\underline{\mathsf{Encrypt}(\pi, m_0, m_1)}$:

1: **if** $(\pi.\alpha_F \neq \mathsf{accepted}) \vee (|m_0| \neq |m_1|)$**:**
2: **return** $\perp$
3:
4: $\pi.u \leftarrow \pi.u + 1$
5: $(C^0, st_{\mathrm{E}}^0) \leftarrow \mathsf{stE.Enc}(\pi.k, m_0, \pi.st_{\mathrm{E}})$
6: $(C^1, st_{\mathrm{E}}^1) \leftarrow \mathsf{stE.Enc}(\pi.k, m_1, \pi.st_{\mathrm{E}})$
7:
8: $(\pi.\vec{C}[u], \pi.st_{\mathrm{E}}) \leftarrow (C^b, st_{\mathrm{E}}^b)$
9:
10: **return** $\pi.\vec{C}[u]$

$\underline{\mathsf{Decrypt}(\pi, C)}$:

1: **if** $(\pi.b = 0) \vee (\pi.\alpha_F \neq \mathsf{accepted})$**:**
2: **return** $\perp$
3:
4: $\pi.v \leftarrow \pi.v + 1$;
5: $(m, \pi.st_{\mathrm{D}}) \leftarrow \mathsf{stE.Dec}(\pi.k, C, \pi.st_{\mathrm{D}})$
6:
7: $\pi' \leftarrow f_T(\pi)$
8: **if** $(\pi' = \perp) \vee (\pi.v > \pi'.u) \vee (C \neq \pi'.\vec{C}[v])$**:**
9: $\pi.\texttt{in-sync} \leftarrow \texttt{false}$
10:
11: **if** $\pi.\texttt{in-sync} = \texttt{false}$**:**
12: **return** m
13: **return** $\perp$

Figure 4: The Encrypt and Decrypt queries for the ACCE security experiment.

ACCE security. The security of a (2P-)ACCE protocol Π is based on experiment $\mathbf{Exp}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda)$ where the base query set $\mathcal{Q}_{base}$ is extended with two additional queries, Encrypt and Decrypt. These two queries allow the adversary to interact with the channels established in the protocol. For bookkeeping purposes, the Encrypt and Decrypt queries—specified in Fig. 4—associate some additional variables to each session π_U^i. Namely:

- b – a random bit drawn randomly at the creation of the π_U^i,
- u, v – counters initialized to 0 and incremented for each call to $\mathsf{Encrypt}(\pi_U^i, \cdot, \cdot)$ and $\mathsf{Decrypt}(\pi_U^i, \cdot)$, respectively,
- $\vec{C}$ – a vector containing the ciphertexts returned from calls to $\mathsf{Encrypt}(\pi_U^i, \cdot, \cdot)$,
- `in-sync` – a flag used to detect trivial wins by the adversary.

By abuse of notation, we use $\pi_U^i.b$, $\pi_U^i.u$, etc., to refer to these variables even though they are not actually part of the local variables of π_U^i in terms of the ACCE protocol syntax.

Let $\mathcal{Q} = \mathcal{Q}_{base} \cup \{\mathsf{Encrypt}, \mathsf{Decrypt}\}$. Experiment $\mathbf{Exp}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda)$ stops when $\mathcal{A}$ outputs a pair (π, b') consisting of a session π and a bit b'. The goal of the adversary, formally captured in the following predicate, is to break either the confidentiality or integrity of one of the channels established by a fresh session.

Definition 6 (ACCE winning events)**.** Suppose (π, b') was the final output by $\mathcal{A}$ in a run of experiment $\mathbf{Exp}_{\Pi,\mathcal{Q}',\mathcal{A}}(\lambda)$.

$$\mathsf{ACCE} \stackrel{\text{def}}{=} \begin{cases} \pi.b = b', & \text{if } \mathsf{Fresh}_{\mathsf{ACCE}}(\pi) = \texttt{true} \\ \texttt{true} \text{ with probability } 1/2, & \text{if } \mathsf{Fresh}_{\mathsf{ACCE}}(\pi) = \texttt{false} \end{cases} \tag{6}$$

Let $\mathbf{Exp}^{\mathsf{ACCE}}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda) \Rightarrow 1$ denote the event that $\mathsf{ACCE} = \texttt{true}$.

Definition 7 (ACCE security)**.** Let f be a partner function. The *ACCE-advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{ACCE}}_{\Pi,\mathcal{A},f}(\lambda) \stackrel{\text{def}}{=} 2 \cdot \Pr[\mathbf{Exp}^{\mathsf{ACCE}}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda) \Rightarrow 1] - 1 \tag{7}$$

A protocol Π is *ACCE-secure*, if there exists a sound partnering function f, such that for all PPT adversaries $\mathcal{A}$, its ACCE-advantage $\mathbf{Adv}^{\mathsf{ACCE}}_{\Pi,\mathcal{A},f}(\lambda)$ is negligible in security parameter λ.

2.5 Explicit entity authentication

Explicit entity authentication, as opposed to *implicit* entity authentication, adds "aliveness" guarantees to a protocol in the sense that if a session at party A accepts with peer B, then A can be certain that there exists a corresponding session at B that contributed to this protocol run. While the need for AKE protocols to provide explicit entity authentication has been somewhat disputed in the literature (see e.g. [10, §1.6], [40, §6] or [27, §2.1]), our use of it in this paper mostly serve as an approximation of the (intuitively) simpler notion of *key confirmation* (see [19] for a detailed treatment of this property). On the other hand, explicit entity authentication has always been part of the security requirements of an ACCE protocol [24, 30, 26], and we are going to assume that in this paper too.

Since the definition of explicit entity authentication is formulated identically for both AKE and ACCE protocols, we give a merged definition here. Let $\mathcal{Q}_{\mathsf{AKE}}$ denote the query set of the AKE experiment (in any of the three security models), and let $\mathcal{Q}_{\mathsf{ACCE}}$ denote the query set of the ACCE experiment.

Definition 8 (Entity authentication predicate)**.** For $\mathsf{X} \in \{\mathsf{AKE}, \mathsf{ACCE}\}$, let T be the transcript of experiment $\mathbf{Exp}_{\Pi,\mathcal{A},\mathcal{Q}_{\mathsf{X}}}(\lambda)$. Predicate Auth is true if and only if the following holds for all fresh sessions π:

$$\pi.\alpha_F = \mathsf{accepted} \implies \exists \pi' \text{ such that } f_{T'}(\pi) = \pi'. \tag{8}$$

Let $\mathbf{Exp}^{\mathsf{X\text{-}Auth}}_{\Pi,\mathcal{Q}_{\mathsf{X}},\mathcal{A}}(\lambda) \Rightarrow 1$ denote the event that Auth is true. A fresh session that accepts without a partner is said to have *accepted maliciously*.

Definition 9 (Explicit entity authentication)**.** A protocol Π provides *explicit entity authentication* if there exists a sound partner function f, such that for all PPT adversaries $\mathcal{A}$, it holds that

1. Π is X-secure, and
2. $\mathbf{Adv}^{\mathsf{X\text{-}EA}}_{\Pi,\mathcal{A},f}(\lambda) \stackrel{\text{def}}{=} 1 - \Pr[\mathbf{Exp}^{\mathsf{X\text{-}Auth}}_{\Pi,\mathcal{Q}_{\mathsf{X}},\mathcal{A}}(\lambda) \Rightarrow 1]$ is negligible in security parameter λ,

where $\mathsf{X} \in \{\mathsf{AKE}, \mathsf{AKE}^{w}, \mathsf{AKE}^{\mathsf{static}}, \mathsf{ACCE}\}$.

Remark 6. Note that the explicit entity authentication of an AKE (resp. ACCE) protocol needs to hold with the *same* partner function as used to prove its AKE (resp. ACCE) security.

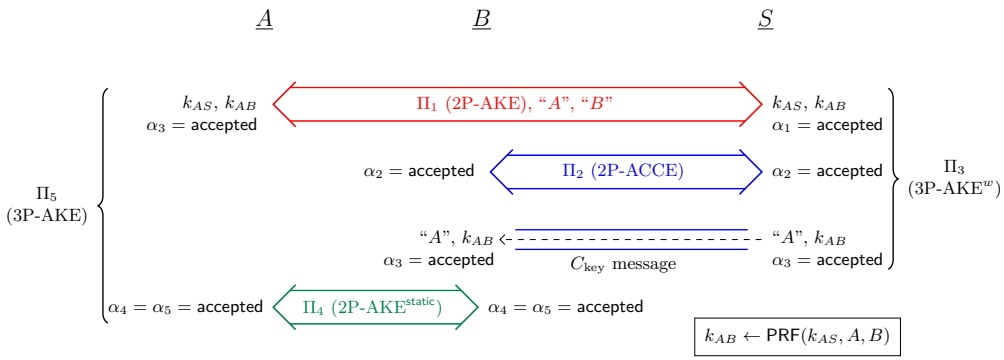

Figure 5: (Right) Construction of a 3P-AKE^w-secure protocol Π_3, using as building blocks a 2P-AKE-secure protocol Π_1, an ACCE-secure protocol Π_2, and a pseudorandom function PRF. (Left) Construction of a 3P-AKE secure protocol Π_5, using as building blocks a 3P-AKE^w-secure protocol Π_3 and a 2P-AKE$^{\text{static}}$-secure protocol Π_4.

3 Generic composition results

In this section we prove two composition theorems for two fairly generic constructions of 3P-AKE protocols. The first construction, shown as protocol Π_3 in Fig. 5, resembles the standalone EAP. It uses as building blocks any secure 2P-AKE protocol (in the strongest AKE model), any secure 2P-ACCE protocol, and a pseudorandom function for channel binding (see Appendix A for a formal definition).

The second construction, shown as protocol Π_5 in Fig. 5, resembles the EAP combined with a subsequent key-confirmation step, modeled here as a 2P-AKE protocol secure in the weakest AKE$^{\text{static}}$ model. We emphasize that the 3P-AKE protocol used as the underlying building block by protocol Π_5, does not have to be based on the Π_3 construction shown in Fig. 5, but can be derived from *any* 3P-AKE^w-secure protocol.

3.1 2P-AKE + 2P-ACCE + channel binding $\Longrightarrow$ 3P-AKE^w

Construction. From a 2P-AKE protocol Π_1 (based on public keys), a 2P-ACCE protocol Π_2 (based on pre-shared symmetric keys), and a pseudorandom function PRF, we construct the 3P-AKE protocol Π_3 shown in Fig. 5. Specifically, protocol Π_3 works as follows. First, sub-protocol Π_1 is run between the initiator A and the server S to derive an intermediate key k_{AS}. A also communicates the identities "A" and "B" to S, where B is the identity of the responder that A wants to talk to. These identities are sent independently of sub-protocol Π_1 and have no integrity protection.

Note that A knows the identities of both S and B at the beginning of the

protocol whereas S learns about B from the identities communicated by A. Technically, this means that a session at A needs to be initialized with the identities of S and B (setting the peers variable accordingly), while a session at S will update its peers variable to include B after receiving this identity from A.

From key k_{AS}, both A and S derive the key $k_{AB} \leftarrow \mathsf{PRF}(k_{AS}, A, B)$. This key will be the ultimate session key shared between A and B in protocol Π_3. In order for S to transfer k_{AB} to B, they first establish a secure channel using sub-protocol Π_2. Once established, S sends the session key k_{AB} together with the identity of A over the channel to B. For simplicity, we assume that the transfer of "A" and k_{AB} is done using a *single* channel message, which we call the C_{key} *message*. Note that unlike the identities sent over the A–S link, the identity on the S–B link enjoys integrity protection from the secure channel between S and B.

The initiator A accepts in protocol Π_3 when it has derived k_{AB}, while the responder B accepts once it has received—and properly decrypted—the C_{key} message, obtaining the session key k_{AB} as well as the identity "A" which it uses to update its peers variable.

Result. Our first composition result shows that protocol Π_3 is 3P-AKE^w-secure if sub-protocol Π_1 is 2P-AKE-secure, sub-protocol Π_2 is 2P-ACCE-secure, and PRF is a pseudorandom function. Note that since Π_3 does not provide explicit entity authentication—in fact, no initiator session A will have a partner at the time it accepts—it cannot achieve security in the strongest AKE model due to the attack mentioned for standalone EAP in the introduction.

Roughly, the proof of the first composition theorem works as follows. The 2P-AKE-security of sub-protocol Π_1 allows us to swap out the intermediate keys k_{AS} with random ones. The PRF-security of the key-derivation function PRF then allows us to replace the derived session keys k_{AB} with random ones. Finally, the ACCE-security of sub-protocol Π_2 ensures that the adversary cannot modify any C_{key} messages nor can it learn anything about the session keys transferred inside them. Thus, at this point the adversary has zero advantage in winning in its 3P-AKE experiment.

Theorem 1. *Let Π_3 be the protocol described in Section 3.1. If protocol Π_1 is 2P-AKE-secure, Π_2 is ACCE-secure using a symmetric partner function, and PRF is a secure PRF, then there exists a sound partner function f_3, such that protocol Π_3 is 3P-AKE^w-secure.*

Concretely, if Π_1 is AKE-secure with the partner function f_1, and Π_2 is ACCE-secure with the symmetric partner function f_2, then we can create a partner function f_3, and adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$ and $\mathcal{D}$, such that

$$\begin{aligned}\mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq\ & \mathbf{Adv}^{\mathsf{ACCE\text{-}EA}}_{\Pi_2,\mathcal{B}_1,f_2}(\lambda) + 2n^2 \cdot \mathbf{Adv}^{\mathsf{2P\text{-}AKE}}_{\Pi_1,\mathcal{B}_2,f_1}(\lambda) \\ & + 2n^2 \cdot \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{D}}(\lambda) + 4n^2 \cdot \mathbf{Adv}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{B}_3,f_2}(\lambda),\end{aligned} \tag{9}$$

where $n = (n_\pi + 1) \cdot |\mathcal{I} \cup \mathcal{R}|$; n_π being an upper bound on the number of sessions at each party.

Proof. We begin by defining the partner function f_3 using the partner functions for sub-protocols Π_1 and Π_2.

Defining the partner function for Π_3. Intuitively, f_3 is constructed by "composing" the two partner functions f_1 and f_2 assumed to exist for sub-protocols Π_1 and Π_2. For example, if π_A^i is an initiator session, then $f_3(\pi_A^i) = \pi_B^j$ if there exists a server session π_S^k, such that $f_1(\pi_A^i) = \pi_S^k$ and $f_2(\pi_S^k) = \pi_B^j$, That is, π_B^j is π_A^i's f_3-partner if there exists a server session π_S^k that acts as the connection between them in the two sub-protocols Π_1 and Π_2.[7]

In more detail, when π_A^i is an initiator session having intended peers B (responder) and S (server), then:

- $f_{3,T_3}(\pi_A^i) = \pi_B^j$ if
  1. $f_{1,T_1}(\pi_A^i) = \pi_S^k$ and $f_{2,T_2}(\pi_S^k) = \pi_B^j$,
  2. $\pi_B^j.\mathsf{peers} = \{A, S\}$,
  3. $\pi_S^k.\mathsf{peers} = \{A, B\}$ (in particular, this means that π_S^k received the same identities that π_A^i sent on the A–S link Fig. 5),
  4. the C_{key} message received by π_B^j was identical to the one produced by π_S^k.
- $f_{3,T_3}(\pi_A^i) = \perp$, otherwise.

When π_B^j is a responder session having intended peers A and S, then f_3 is defined similarly by "reversing" the order of f_1 and f_2:

- $f_{3,T_3}(\pi_B^j) = \pi_A^i$ if
  1. $f_{2,T_2}(\pi_B^j) = \pi_S^k$ and $f_{1,T_1}(\pi_S^k) = \pi_A^i$;
  2. $\pi_A^i.\mathsf{peers} = \{B, S\}$,
  3. $\pi_S^k.\mathsf{peers} = \{A, B\}$,
  4. the C_{key} message received by π_B^j was identical to the one produced by π_S^k.
- $f_{3,T_3}(\pi_B^j) = \perp$, otherwise.

The soundness of f_3 follows from the soundness of f_1 and f_2, together with the fact that the C_{key} message must have been delivered correctly in the protocol run involving the relevant session.

[7] Technically, to make this formally precise, one needs to extract from the 3P-AKE transcript T two transcripts T_1 and T_2, containing the queries pertaining to the two-party sub-protocols Π_1 and Π_2, so that running f_1 and f_2 on them is well-defined. See Appendix C, Table 2 for details.

AKE^w-security. The proof of AKE^w-security of protocol Π_3 is structured as a sequence of games (see [42]). In the following, when we say that a certain game *aborts*, we mean that the challenger stops the execution of the experiment and outputs a random bit on $\mathcal{A}$'s behalf.

Game 0: This is the real 3P-AKE^w security game, hence

$$\mathbf{Adv}^{\mathsf{G}_0}_{\Pi_3,\mathcal{A},f_3}(\lambda) = \mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{A},f_3}(\lambda) \ .$$

Game 1: This game proceeds as the previous one, but aborts if a fresh responder or server session *accepts maliciously* in sub-protocol Π_2, meaning that it accepted without a partner in Π_2 according to f_2.

Lemma 1. $\mathbf{Adv}^{\mathsf{G}_0}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_1}_{\Pi_3,\mathcal{A},f_3}(\lambda) + \mathbf{Adv}^{\mathsf{ACCE\text{-}EA}}_{\Pi_2,\mathcal{B}_1,f_2}(\lambda)$.

Proof (sketch). Reduction $\mathcal{B}_1$ begins by creating all the long-term keys for sub-protocol Π_1 and selecting a random bit b. Essentially, $\mathcal{B}_1$ will simulate the Π_1 part of Π_3 itself, while forwarding all messages pertaining to Π_2 to its 2P-ACCE challenger. In particular, $\mathcal{B}_1$ creates all the intermediate keys k_{AS} itself, and from them derive the session keys k_{AB}. In order to create the C_{key} message of some server session π, $\mathcal{B}_1$ issues an $\mathsf{Encrypt}(\pi, k_{AB}, k_{AB})$ query to its own ACCE experiment. Moreover, when $\mathcal{A}$ issues a Test query, then depending on bit b, $\mathcal{B}_1$ returns the real session key or a random key. When $\mathcal{A}$ terminates, then $\mathcal{B}_1$ terminates too (in this case no malicious accept has occurred).

To analyze $\mathcal{B}_1$'s winning probability, we only have to observe that $\mathcal{B}_1$ provides a perfect simulation of Π_3 for $\mathcal{A}$. This means that if a malicious accept occurs in sub-protocol Π_2, then a malicious accept also occurs in $\mathcal{B}_1$'s ACCE experiment. □

Remark 7. The abort condition in Game 1 does not mean that every session in protocol Π_3 will have a partner (according to f_3). In fact, no initiator session will have a partner at the time when it accepts because at that point sub-protocol Π_2 hasn't even started yet.

Game 2: This game implements a *selective* AKE security game [31, §3.3], rather than the normal adaptive one. That is, at the beginning of the game, the adversary is required to "commit" to its choice of test-session and its partner (if any).

Technically, at the beginning of the game the adversary must output two pairs (U, i) and (V, j), with $i \in [1, n_\pi]$ and $j \in [0, n_\pi]$, where n_π is an upper bound on the number of sessions at each party, and a choice of $j = 0$ means that π^i_U is not intended to get a partner. The game then proceeds as in Game 1, except that if either of the following events occur, then the challenger "penalizes" the adversary by outputting a random bit at the end.

(i) π^i_U was not selected as the test-session by $\mathcal{A}$.

(ii) π_U^i gets a different partner than π_V^j (including the case that it *gets* a partner if $j = 0$).

Lemma 2.

$$\mathbf{Adv}^{\mathsf{G}_1}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq (n_\pi + 1)^2 \cdot |\mathcal{I} \cup \mathcal{R}|^2 \cdot \mathbf{Adv}^{\mathsf{G}_2}_{\Pi_3,\mathcal{A}',f_3}(\lambda). \tag{10}$$

Proof. From an adversary $\mathcal{A}$ that wins against the adaptive game in Game 1, we create an adversary $\mathcal{A}'$ that wins against the selective game in Game 2 as follows. First, $\mathcal{A}'$ randomly selects a pair $(U, i) \twoheadleftarrow (\mathcal{I} \cup \mathcal{R}) \times [1, n_\pi]$ and a pair (V, j), which, depending on the role of U, is either selected as $(V, j) \twoheadleftarrow \mathcal{I} \times [0, n_\pi]$ or $(V, j) \twoheadleftarrow \mathcal{R} \times [0, n_\pi]$. It outputs (U, i) and (V, j) as its choice to the selective security game it is playing. $\mathcal{A}'$ then runs $\mathcal{A}$ and answers all of its queries by forwarding them to its own selective security game. When $\mathcal{A}$ stops with output b', then $\mathcal{A}'$ stops and outputs the same bit as well.

Algorithm $\mathcal{A}'$ perfectly simulates Game 1 for $\mathcal{A}$, so $\mathcal{A}'$'s choice of selective security targets matches those of $\mathcal{A}$ with probability at least $1/\left((n_\pi + 1) \cdot |\mathcal{I} \cup \mathcal{R}|\right)^2$. When $\mathcal{A}'$'s guess is correct it wins with the same probability as $\mathcal{A}$, while when it is wrong, $\mathcal{A}'$ gets penalized in its selective security game, hence wins with probability 1/2. □

In the remaining games, let π_U^i and π_V^j denote the targets that the adversary commits to in its selective security game; π_U^i being the test-session, and π_V^j its (potentially empty) partner. Define the *co-partner* of π_U^i to be the server session being involved in the protocol run between π_U^i and π_V^j. Specifically, if π_U^i has the initiator role, then its co-partner is defined to be $f_{1,T_1}(\pi_U^i)$; while if π_U^i has the responder role, then its co-partner is defined to be $f_{2,T_2}(\pi_U^i)$.

Note that if π_U^i has the initiator role, then its co-partner does not necessarily exists, while if π_U^i has the responder role, then its co-partner is guaranteed to exist by Game 1.

Game 3: This game proceeds as the previous one, except that it swaps out the intermediate key k_{AS} derived in sub-protocol Π_1 with a random key in the protocol run involving the test-session. That is, for the session out of π_U^i and π_V^j that has the initiator role in protocol Π_3, the challenger replaces its intermediate key k_{AS} in sub-protocol Π_1 with a random key. Moreover, the intermediate key derived by its partner in sub-protocol Π_1 (if any) is also replaced with the same random key.

Lemma 3. $\mathbf{Adv}^{\mathsf{G}_2}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_3}_{\Pi_3,\mathcal{A},f_3}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{2P\text{-}AKE}}_{\Pi_1,\mathcal{B}_2,f_1}(\lambda)$.

Proof (sketch). Reduction $\mathcal{B}_2$ begins by drawing a random bit b_{sim} and creates all the long-term PSKs for sub-protocol Π_2. $\mathcal{B}_2$ then runs $\mathcal{A}$ and forwards all its queries that pertain to sub-protocol Π_1 to its own AKE experiment. All queries that pertain to sub-protocol Π_2, $\mathcal{B}_2$ answers itself using the PSKs it created. It also implements all the abort conditions of the previous games. To answer $\mathcal{A}$'s $\mathsf{Test}(\pi_U^i)$ query, $\mathcal{B}_2$ proceeds as follows.

If $b_{sim} = 1$ then $\mathcal{B}_2$ responds as normal by drawing a random key and returning this to $\mathcal{A}$. If $b_{sim} = 0$ and π_U^i is an initiator session, then $\mathcal{B}_2$ makes a $\mathsf{Test}(\pi_U^i)$ query to its own 2P-AKE game to obtain π_U^i's intermediate key k_{AS} in sub-protocol Π_1. From k_{AB} (which is either π_U^i's real key in sub-protocol Π_1 or a random one, depending on the secret bit in $\mathcal{B}_2$'s 2P-AKE experiment) and the pseudorandom function PRF, $\mathcal{B}_2$ derives the key k_{AB} which it finally returns to $\mathcal{A}$. If $b_{sim} = 0$ and π_U^i is a responder session, then π_U^i must have a co-partner π_S^k by Game 1. To obtain the intermediate key k_{AS} needed to derive the session key k_{AB}, $\mathcal{B}_2$ first makes a $\mathsf{Test}(\pi_S^k)$ query to its own 2P-AKE experiment. From the returned key k_{AB} and , the pseudorandom function PRF, $\mathcal{B}_2$ derives the key k_{AB} which it returns to $\mathcal{A}$.

When $\mathcal{A}$ outputs its guess b', then $\mathcal{B}_2$ stops and outputs 0 to its 2P-AKE experiment if $b' = b_{sim}$, and 1 otherwise.

Note that if the Test query made by $\mathcal{B}_2$ in its own AKE experiment returns a real key k_{AS}, then $\mathcal{B}_2$ perfectly simulates Game 2, while if the Test query returns a random key then $\mathcal{B}_2$ simulates Game 3. Thus, the lemma follows if we can show that test-session chosen by $\mathcal{B}_2$ in its own 2P-AKE experiment is fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}}$ whenever the test-session π_U^i chosen by $\mathcal{A}$ is fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^w}$.

If π_U^i is an initiator session, then $\mathcal{B}_2$ uses the same session π_U^i as the test-session in its own 2P-AKE experiment. The AKE-freshness of π_U^i then follows immediately since predicate $\mathsf{Fresh}_{\mathsf{AKE}^w}$ is more restrictive than predicate $\mathsf{Fresh}_{\mathsf{AKE}}$. If π_U^i is a responder session, then the test-session chosen by $\mathcal{B}_2$ is π_U^i's co-partner π_S^k, so we need to argue that π_S^k is fresh in $\mathcal{B}_2$'s 2P-AKE experiment whenever π_U^i is AKE^w-fresh. There are two cases to consider: either π_U^i has an f_3-partner or it does not. If π_U^i has a partner (which by Game 2 must be π_V^j), then $\mathcal{A}$ cannot have made a $\mathsf{Reveal}(\pi_V^j)$ query since this would violate the AKE^w-freshness of π_U^i. Moreover, since f_3 is constructed from f_1 and f_2, π_V^j must be π_S^k's f_1-partner. Thus, $\mathcal{B}_2$ is also allowed to forward any $\mathsf{LongTermKeyReveal}$ query to either A or S without violating the freshness of π_S^k according to predicate $\mathsf{Fresh}_{\mathsf{AKE}}$.

If π_U^i does not have an f_3-partner, then $\mathcal{A}$ cannot have made any $\mathsf{LongTermKeyReveal}$ query to A or S (since this would violate AKE^w-freshness), and thus neither has $\mathcal{B}_2$. Furthermore, if π_U^i doesn't have an f_3-partner then this implies that its co-partner π_S^k cannot have an f_1-partner either. Thus, $\mathcal{B}_2$ can safely forward all of $\mathcal{A}$'s Reveal queries without violating the AKE-freshness of π_S^k. □

Game 4: This game proceeds as the previous one, except that when deriving the session key k_{AB} in the protocol run involving the test-session π_U^i, the challenger uses a random function $\$(\cdot,\cdot)$ rather than the function $\mathsf{PRF}(k_{AS},\cdot,\cdot)$.

More specifically, if π_U^i has the initiator role then its session key k_{AB} is derived using the random function $\$(\cdot,\cdot)$ instead of the function $\mathsf{PRF}(k_{AS},\cdot,\cdot)$. Additionally, if π_U^i has a co-partner π_S^k, then π_S^k uses the same random function to derive the key k_{AB} that it will forward in its C_{key} message.

If π_U^i has the responder role, then it must have a co-partner π_S^k by Game 1. When deriving the key k_{AB} that π_S^k will use for its C_{key} message, the challenger uses the random function $\$(\cdot,\cdot)$ instead of the function $\mathsf{PRF}(k_{AS},\cdot,\cdot)$.

Lemma 4. $\mathbf{Adv}^{\mathsf{G}_3}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_4}_{\Pi_3,\mathcal{A},f_3}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF}}(\mathcal{D})$.

Proof. Algorithm $\mathcal{D}$ has access to an oracle $\mathcal{O}$ which either implements the function $\mathsf{PRF}(\widetilde{k},\cdot,\cdot)$ with an independent and uniformly distributed key $\widetilde{k}$, or a random function $\$(\cdot,\cdot)$. $\mathcal{D}$ begins by drawing a random bit b_{sim} and creates all the long-term keys for sub-protocols Π_1 and Π_2. Next, it runs $\mathcal{A}$ and answers all its queries according to Game 3 by using the keys it created, except that it answers $\mathcal{A}$'s $\mathsf{Test}(\pi_U^i)$ query as follows.

If $b_{sim} = 1$, then $\mathcal{D}$ returns a random key as normal. If $b_{sim} = 0$ and π_U^i is an initiator session, then $\mathcal{D}$ answers with $\mathcal{O}(U,V)$ (recall that $\pi_U^i = \pi_U^i$ and $\pi_V^j = \pi_V^j$). If $b_{sim} = 0$ and π_U^i is a responder session, then $\mathcal{D}$ answers with $\mathcal{O}(V',U')$, where V' and U' were the identities that the co-partner of π_U^i received over the A–S link in Fig. 5 (recall that if π_U^i is a responder session it is guaranteed to have a co-partner by Game 1).

When $\mathcal{A}$ outputs its guess b', then $\mathcal{D}$ stops and outputs 0 to its PRF-game if $b' = b_{sim}$, and 1 otherwise.

When $\mathcal{D}$'s oracle $\mathcal{O}$ implements PRF, then $\mathcal{D}$ perfectly simulates Game 3, while if $\mathcal{O}$ implements a random function $\$(\cdot,\cdot)$, then $\mathcal{D}$ perfectly simulates Game 4. Thus, the advantage difference of $\mathcal{A}$ winning in Game 3 and Game 4 corresponds exactly to the probability difference that $\mathcal{D}$ outputs 1 when interacting with PRF or a random function $\$(\cdot,\cdot)$ as its oracle $\mathcal{O}$. □

At this point one might expect that the adversary should be unable to distinguish the test-key from random since the session key of π_U^i is now derived using a random function rather than the pseudorandom function PRF. Unfortunately, we cannot (currently) rule out that $\mathcal{A}$ might be able to learn something about the session key through the C_{key} message delivered from the server to the responder. Furthermore, $\mathcal{A}$ could potentially also modify the C_{key} message in such a way that it still decrypts to the same session key. In this case π_U^i and π_V^j would end up with the same key while at the same time not being partners according to the definition of the partner function f_3. Hence, $\mathcal{A}$ could safely reveal π_V^j and trivially win.

In the following two games we show that neither of these scenarios are possible due to the ACCE-security of sub-protocol Π_2. In the first game we show that $\mathcal{A}$ is unable to successfully forge the C_{key} message in the protocol run involving π_U^i, and in the second game we show that $\mathcal{A}$ is unable learn anything about the session key from observing the C_{key} message.

Game 5: Suppose π_U^i has a co-partner π_S^k and that the ciphertext C was the C_{key} message produced by π_S^k (if any). Let π^* be the f_2-partner of π_S^k in sub-protocol Π_2 required to exist by Game 1. Game 5 proceeds as Game 4, but if

π^* receives a C_{key} message different from C then it is simply assumed to have decrypted to $\perp$.

Remark 8. Note that if π_U^i has the responder role, then π^* is π_U^i itself, while if π_U^i has the initiator role then π^* (if it exists) is some responder session. We write "if it exists" because if π_U^i has the initiator role then it might not actually have a co-partner at all. However, in that case there is no difference between Game 4 and Game 5 since is no relevant C_{key} message is being created and thus also no relevant π^* session.

Lemma 5. $\mathbf{Adv}^{\mathsf{G}_4}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_5}_{\Pi_3,\mathcal{A},f_3}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{B}'_3,f_2}(\lambda)$.

Proof (sketch). Assume π_U^i has a co-partner π_S^k, and that π_S^k produced the ciphertext C as its C_{key} message. With the definition of π^* being the same as in the game description above, let F denote the event that $\mathcal{A}$ successfully forges the C_{key} message being delivered to π^*, that is:

- *Event* F: $\mathcal{A}$ sends to π^* a C_{key} message $C' \neq C$, and C' decrypts to something other than $\perp$.

As long as event F does not occur then Game 4 and Game 5 are identical, hence, by the Difference Lemma [42], we have

$$\mathbf{Adv}^{\mathsf{G}_4}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_5}_{\Pi_3,\mathcal{A},f_3}(\lambda) + \Pr[F]. \tag{11}$$

To bound $\Pr[F]$ we create an adversary $\mathcal{B}'_3$ that capitalizes on the event F in order to break the ACCE-security of sub-protocol Π_2. Algorithm $\mathcal{B}'_3$ begins by drawing a random bit b_{sim} and creates all the (asymmetric) long-term keys for sub-protocol Π_1. All of $\mathcal{A}$'s Send queries that pertain to sub-protocol Π_1, $\mathcal{B}'_3$ answers itself using the long-term keys it created. Particularly, $\mathcal{B}'_3$ can answer all $\mathsf{LongTermKeyReveal}$ queries targeting the asymmetric long-term keys of initiators and servers itself. Moreover, it can also answer all of $\mathcal{A}$'s Reveal queries that targets initiator sessions, as well as its Test query if π_U^i if $U \in \mathcal{I}$ (using the bit b_{sim}).

Send queries that pertain to sub-protocol Π_2, as well as $\mathsf{LongTermKeyReveal}$ queries that target the PSKs shared between servers and responders, are forwarded to $\mathcal{B}'_3$'s own ACCE experiment. Whenever a server session π accepts in sub-protocol Π_2, then $\mathcal{B}'_3$ creates its C_{key} message by querying $\mathsf{Encrypt}(\pi, A\|k_{AB}, A\|k_{AB})$[8] to its ACCE experiment, where "A" is the identity of the initiator that π received on the A–S link in Fig. 5, and k_{AB} is the session key that $\mathcal{B}'_3$ derived from π's intermediate key k_{AS} in sub-protocol Π_1.

Whenever $\mathcal{A}$ forwards a C_{key} message to a responder session different from π^*, then $\mathcal{B}'_3$ first makes a Reveal query to that session in its ACCE experiment in order to obtain its channel-key for sub-protocol Π_2. Using this channel-key, $\mathcal{B}'_3$ decrypts the received C_{key} message and simulates the responder

[8] Here we are abusing notation and using "π" to denote both the server session that $\mathcal{B}'_3$ simulates for $\mathcal{A}$ in protocol Π_3, as well as the corresponding "proxy" server session that $\mathcal{B}'_3$ creates in its own ACCE experiment in order to answer the queries to the former.

session accordingly (i.e., rejecting if the C_{key} message didn't decrypt properly, while accepting and setting the session-key and peers variables based on the decrypted C_{key} message if not). Consequently, $\mathcal{B}_3'$ can also answer all of $\mathcal{A}$'s Reveal queries targeting responder sessions different from π^*.

If $\mathcal{A}$ at any point stops during $\mathcal{B}_3'$'s simulation, not having sent a C_{key} message to π^*, then $\mathcal{B}_3'$ stops too and outputs (π, b') to its ACCE experiment where π is an arbitrary session and b' is a random bit, i.e., $b' \twoheadleftarrow \{0,1\}$.

Finally, if $\mathcal{A}$ *does* forward a C_{key} message C' to π^*, then $\mathcal{B}_3'$ stops its simulation and outputs (π^*, b') to its ACCE game, where the bit b' is determined as follows. If $C' = C$, where C is the C_{key} message produced by π_U^i's co-partner π_S^k, then $b' \twoheadleftarrow \{0,1\}$. If $C' \neq C$, meaning that C' is a potential C_{key} message forgery, then $\mathcal{B}_3'$ first makes the query $m' \leftarrow \mathsf{Decrypt}(\pi^*, C')$ to its ACCE experiment. If $m' \neq \perp$ then $b' = 1$, otherwise $b' \twoheadleftarrow \{0,1\}$. That is, if $\mathcal{B}_3'$'s $\mathsf{Decrypt}$ query returns something other than $\perp$, then it outputs 1, while in all other cases it outputs a random bit.

We now analyze $\mathcal{B}_3'$. If $\mathcal{A}$ does not send a C_{key} message to π^* during $\mathcal{B}_3'$'s simulation then $\mathcal{B}_3'$ outputs (π, b') to its ACCE experiment, where π is an arbitrary session and b' a random. In this case $\mathcal{B}_3'$ clearly wins with probability $1/2$.

If $\mathcal{A}$ *does* send a C_{key} message to π^*, then $\mathcal{B}_3'$ picks π^* as its ACCE target. We begin by arguing that π^* is fresh according to predicate $\mathsf{Fresh}_{\mathsf{ACCE}}$, provided $\mathcal{A}$'s test-target π_U^i is fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^w}$. Since $\mathcal{B}_3'$'s simulation stops immediately once π^* accepts, we only have to consider the effects of $\mathsf{LongTermKeyReveal}$ queries against the PSK shared between π^* and its server peer.

If π_U^i has the responder role then $\pi_U^i = \pi^*$. By Game 1, π^* must have a f_2-partner in sub-protocol Π_2 and its ACCE-freshness follows immediately since then any long-term key can legally be exposed; in particular, this includes the PSK shared between π^* and its server peer.

Now suppose π_U^i has the initiator role. If π_U^i does not have a co-partner or this co-partner never reached the accept state (hence not producing a C_{key} message), then there is nothing to prove since then there is also no π^* session. On the other hand, if π_U^i has co-partner π_S^k which created a C_{key} message C, then by Game 1 there must be some session π^* being the f_2-partner of π_S^k. If $\mathcal{A}$ forwards C unmodified to π^*, then π_U^i and π^* would be f_3-partners and so the ACCE-freshness of π^* would again follow immediately.[9] Conversely, if $\mathcal{A}$ does not forward C unmodified to π^*, then π_U^i and π^* would not be f_3-partners. Hence, if π_U^i is to be fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^w}$, then the long-term keys of its peers cannot have been exposed. In particular, this means that the PSK of π^* cannot have been exposed. It follows that π^* is fresh according to predicate $\mathsf{Fresh}_{\mathsf{ACCE}}$.

It remains to calculate $\mathcal{B}_3'$'s winning probability when $\mathcal{A}$ forwards a C_{key} message to π^*—meaning that $\mathcal{B}_3'$ picked π^* as its ACCE target. If the C_{key}

[9] Although in this case it doesn't even matter that π^* is ACCE-fresh since $\mathcal{B}_3'$ would nevertheless output a random bit b' if π^* and π_U^i were partners (since event F wouldn't have happened in this case).

message that π^* received was forwarded unmodified from its f_2-partner π_S^k, then $\mathcal{B}_3'$ outputs a random bit and thus wins with probability 1/2. On the other hand, if the C_{key} message that π^* received was different from the one that π_S^k sent out, then there is a potential for event F to occur. Note that $\mathcal{B}_3'$ perfectly simulates Game 4 until $\mathcal{A}$ sends a C_{key} message to π^*, so the probability that F occurs in $\mathcal{B}_3'$'s simulation is the same as the probability that F occurs in Game 4.

Let C' be the C_{key} message that π^* received. Recall that $\mathcal{B}_3'$ outputs 1 if the following $\mathsf{Decrypt}(\pi^*, C')$ query returned something other than $\perp$, and a random bit otherwise. Hence, $\mathcal{B}_3'$ only capitalizes on event F if π^*'s secret bit in the ACCE experiment is 1 (since if $\pi^*.b = 0$ then the $\mathsf{Decrypt}$ query always returns $\perp$; see Fig. 4). In particular, we have $\Pr[\mathbf{Exp}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{Q},\mathcal{B}_3'}(\lambda) \Rightarrow 1 \mid F \wedge \pi^*.b = 1] = 1$ and $\Pr[\mathbf{Exp}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{Q},\mathcal{B}_3'}(\lambda) \Rightarrow 1 \mid F \wedge \pi^*.b = 0] = 1/2$.

Finally, note that the value of π^*'s secret bit b in $\mathcal{B}_3'$'s ACCE experiment is independent of event F. This is because there is nothing in $\mathcal{B}_3'$'s simulation that depends on $\pi^*.b$ before π^* receives the C_{key} message, and $\mathcal{B}_3'$'s simulation stops immediately once this happens. Thus $\Pr[\pi^*.b = b \mid F] = 1/2$. Conditioned on event F occurring $\mathcal{B}_3'$ winning probability is then

$$\begin{aligned}\Pr[\mathbf{Exp}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{Q},\mathcal{B}_3'}(\lambda) \Rightarrow 1 \mid F] &= \Pr[\mathbf{Exp}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{Q},\mathcal{B}_3'}(\lambda) \Rightarrow 1 \mid F \wedge \pi^*.b = 0] \cdot \frac{1}{2} \\ &\quad + \Pr[\mathbf{Exp}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{Q},\mathcal{B}_3'}(\lambda) \Rightarrow 1 \mid F \wedge \pi^*.b = 1] \cdot \frac{1}{2} \qquad (12) \\ &= \frac{1}{2} \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{3}{4}. \qquad (13)\end{aligned}$$

Combing the above probability with the case when F does not occur yields Lemma 5. □

The previous game established that $\mathcal{A}$ cannot modify the C_{key} message in the protocol run involving the test-session π_U^i. The next and final game shows that $\mathcal{A}$ also cannot learn anything about π_U^i's session key by merely observing the C_{key} message.

Game 6: This game proceeds as the previous one, but when creating the C_{key} message of the co-partner of π_U^i, the challenger encrypts the all-zero string 0^λ instead of the session key k_{AB}. If this C_{key} message is eventually delivered to the intended responder session (either π_U^i or π_V^j), then its session key is still set to be k_{AB} however.

Lemma 6. $\mathbf{Adv}^{\mathsf{G}_5}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_6}_{\Pi_3,\mathcal{A},f_3}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{B}_3'',f_2}(\lambda)$.

Proof (sketch). $\mathcal{B}_3''$ is identical to algorithm $\mathcal{B}_3'$ in the proof of Lemma 5, except for the following differences:

- When creating the C_{key} message of π_U^i's co-partner (if it exists), say π_S^k, $\mathcal{B}_3''$ makes the query $\mathsf{Encrypt}(\pi_S^k, A\|k_{AB}, A\|0^\lambda)$ instead of $\mathsf{Encrypt}(\pi_S^k, A\|k_{AB}, A\|k_{AB})$.

- If $\mathcal{A}$ sends a C_{key} message C' to π^* (with the same definition of π^* as in the description of Game 5), then $\mathcal{B}_3''$ does *not* stop its simulation. Instead $\mathcal{B}_3''$ continues the simulation as follows.

  If C' is equal to the C_{key} message that was previously output by the co-partner of π_U^i using the $\mathsf{Encrypt}(\pi_S^k, A\|k_{AB}, A\|0^\lambda)$ query described above, then π^*'s peer and session key variables are set based on the "left-input" to the $\mathsf{Encrypt}$ query.

  If C' is not equal to that C_{key} message, then C' is assumed to have decrypted to $\perp$ and π^*'s running state α_F is set to $\mathsf{rejected}$.

- Finally, when $\mathcal{A}$ outputs its guess b', then $\mathcal{B}_3''$ outputs the following to its ACCE experiment. If the test-session π_U^i has a co-partner π_S^k, then $\mathcal{B}_3''$ outputs $(\pi_S^k, 0)$ if $b' = b_{sim}$ and $(\pi_S^k, 1)$ otherwise. If the test-session does not have a co-partner, then $\mathcal{B}_3''$ simply outputs an arbitrary session together with a random bit.

Note that if the test-session does not have a co-partner then there is no difference between Game 5 and Game 6, and $\mathcal{B}_3''$ perfectly simulates it. If the test-session has a co-partner π_S^k, and $\pi_S^k.b = 0$ in $\mathcal{B}_3''$'s ACCE experiment, then $\mathcal{B}_3''$ perfectly simulates Game 5 (since the C_{key} message of π_S^k is an encryption of the actual session key k_{AB}). On the other hand, if $\pi_S^k.b = 1$ then $\mathcal{B}_3''$ perfectly simulates Game 6 (since the C_{key} message of π_S^k is an encryption of 0^λ).

It only remains to argue that whenever $\mathcal{B}_3''$ uses π_S^k as its ACCE target-session, then it is fresh according to predicate $\mathsf{Fresh}_{\mathsf{ACCE}}$ as long as π_U^i is fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^w}$. However, this follows by the same arguments that was used to show that π^* was ACCE-fresh in the proof of Lemma 5, so we omit it. □

Concluding the proof of Theorem 1. We argue that $\mathbf{Adv}^{\mathsf{G}_6}_{\Pi_3,\mathcal{A},f_3}(\lambda) = 0$. By the change in Game 4, the session key of the test-session π_U^i is derived using a random function $\$(A, B)$, where "$A$" and "$B$" are the identities of the initiator and responder that π_U^i believes took part in this protocol run. We claim that the only other session that holds a session key derived from $\$(\cdot,\cdot)$ using the same identities "A" and "B", is π_U^i's partner π_V^j (if it exists).

First, note that the random function is evaluated at no more than two sessions: one initiator session and one server session. Second, the session key derived by the server session is delivered to at most one responder session. Finally, the identities used to evaluate $\$(\cdot,\cdot)$ at the initiator and server could be different since $\mathcal{A}$ can modify the identities communicated at the A–S link in Fig. 5.

However, if $\mathcal{A}$ modifies these identities, then the initiator and server derive independent keys, which means that the initiator and responder will ultimately have independent keys too. Moreover, the initiator and responder sessions will not be partners since the communicated identities at the S–B link in Fig. 5 will be different too (recall that f_3-partnering includes the sessions' recorded peers, and by Game 5 the adversary is unable to change the C_{key} message). On the other hand, if the identities were the same, then the initiator and responder

session would necessarily be f_3-partners. This follows because the initiator has the server session as its f_1-partner in sub-protocol Π_1 and the server session's C_{key} message, if delivered at all, must be delivered honestly to its f_2-partner in sub-protocol Π_2. Combined with their agreement on their peers, this means the initiator and responder session would be partners by the definition of f_3.

Altogether, since the session key of the test-session is derived using an independent random function, and since the corresponding C_{key} message leaks nothing about the session key by Game 6, the adversary has zero advantage in Game 6 as claimed.

Combining the bounds from Lemma 1 to Lemma 6 we get

$$\begin{aligned}\mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{A},f_3}(\lambda) \leq\ & \mathbf{Adv}^{\mathsf{ACCE\text{-}EA}}_{\Pi_2,\mathcal{B}_1,f_2}(\lambda) + 2n^2 \cdot \mathbf{Adv}^{\mathsf{2P\text{-}AKE}}_{\Pi_1,\mathcal{B}_2,f_1}(\lambda) + 2n^2 \cdot \mathbf{Adv}^{\mathsf{PRF}}_{\mathsf{PRF},\mathcal{D}}(\lambda) \\ & + 2n^2 \cdot \mathbf{Adv}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{B}'_3,f_2}(\lambda) + 2n^2 \cdot \mathbf{Adv}^{\mathsf{ACCE}}_{\Pi_2,\mathcal{B}''_3,f_2}(\lambda),\end{aligned}$$

where $n = (n_\pi + 1) \cdot |\mathcal{I} \cup \mathcal{R}|$.

Letting $\mathcal{B}_3$ be the ACCE adversary that with probability 1/2 either implements algorithm $\mathcal{B}'_3$ or algorithm $\mathcal{B}''_3$, the concrete bound (9) in the statement of Theorem 1 follows. □

Remark 9. Note that the conclusion above only holds if protocol Π_3 employs channel binding. If the identities of A and B did not go into to the evaluation of the pseudorandom function PRF, then Π_3 would be vulnerable to a simple UKS attack: just change the responder identity "B" being sent over the (unauthenticated) A–S link from "B" to "B'". Without channel binding, A and B' would obtain the same session key but disagree on their intended peers.

3.2 3P-AKE^w + 2P-AKE$^{\mathsf{static}}$ $\Longrightarrow$ 3P-AKE

Construction. From a 3P-AKE protocol Π_3 and a PSK-based 2P-AKE protocol Π_4, we construct the 3P-AKE protocol Π_5 shown in Fig. 5. Specifically, protocol Π_5 works as follows. First, sub-protocol Π_3 is run between A, B and S in order to establish an intermediate "session key" K_{Π_3}. Then, sub-protocol Π_4 is run between A and B using K_{Π_3} as the their "long-term key" PSK. The session key derived in Π_4 becomes A and B's final session key in Π_5.

Result. Our second composition result shows that protocol Π_5 is 3P-AKE-secure if sub-protocol Π_3 is 3P-AKE^w-secure and sub-protocol Π_4 is 2P-AKE$^{\mathsf{static}}$-secure with explicit entity authentication. We remark that the last requirement is necessary in order for our proof to go through. In fact, Π_5 inherits the property of explicit entity authentication from sub-protocol Π_4. Moreover, while Π_4 does not achieve any forward secrecy on its own, protocol Π_5 does. The reason is that within Π_5, sub-protocol Π_4 is merely used to upgrade the security of Π_3 from weak forward secrecy to full forward secrecy.

Theorem 2. *Let Π_5 be the protocol described in Section 3.2. If protocol Π_3 is 3P-AKE^w-secure and protocol Π_4 is 2P-AKE$^{\mathsf{static}}$-secure with explicit entity authentication, then there exists a sound partner function f_5 such that protocol Π_5 is 3P-AKE-secure.*

Concretely, for partner functions f_3 and f_4, we can create a partner function f_5, and adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, such that

$$\mathbf{Adv}^{\mathsf{3P\text{-}AKE}}_{\Pi_5,\mathcal{A},f_5}(\lambda) \leq 4n^2 \cdot \mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{B}_1,f_3}(\lambda) + 2n^2 \cdot \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}}_{\Pi_4,\mathcal{B}_2,f_4}(\lambda) \tag{14}$$

where $n = (n_\pi + 1) \cdot |\mathcal{I} \cup \mathcal{R}|$; n_π being an upper bound on the number of sessions at each party.

Proof of Theorem 2. First we need to define a partner function for protocol Π_5. Similar to the definition of the partner function used in the proof of Theorem 1, we construct f_5 from the partner functions f_3 and f_4 assumed to exist for subprotocols Π_3 and Π_4. However, the definition of f_5 is much simpler since both f_3 and f_4 have the same "domain" and "range" in terms of the sessions they partner. That is, both f_3 and f_4 partner sessions belonging to the sets $\mathcal{I}$ and $\mathcal{R}$. This is different from f_1 and f_2 in the proof of Theorem 1 where f_1 partnered sessions belonging to the sets $\mathcal{I}$ and $\mathcal{S}$, while f_2 partnered sessions belonging to the sets $\mathcal{I}$ and $\mathcal{R}$. Consequently, f_5 can simply be defined as follows:

$$f_{5,T_5}(\pi) = \pi' \iff (f_{3,T_3}(\pi) = \pi') \wedge (f_{4,T_4}(\pi) = \pi') \tag{15}$$

The soundness of f_5 follows directly from the soundness of f_3 and f_4.

AKE-security.

Game 0: This is the real 3P-AKE security game, hence

$$\mathbf{Adv}^{\mathsf{G}_0}_{\Pi_5,\mathcal{A},f_5}(\lambda) = \mathbf{Adv}^{\mathsf{3P\text{-}AKE}}_{\Pi_5,\mathcal{A},f_5}(\lambda) \ .$$

Game 1: In this game, the challenger aborts if a session accepts maliciously in protocol Π_5. Let M be denote this event. Since Game 0 and Game 1 are identical until M occurs, it follows by the Difference Lemma [42] that

$$\mathbf{Adv}^{\mathsf{G}_0}_{\Pi_5,\mathcal{A},f_5}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_1}_{\Pi_5,\mathcal{A},f_5}(\lambda) + \Pr[M]. \tag{16}$$

The following lemma bounds $\Pr[M]$. The proof is given in Appendix B.

Lemma 7.

$$\Pr[M] \leq (n_\pi + 1)^2 \cdot |\mathcal{I} \cup \mathcal{R}|^2 \cdot \left(2 \cdot \mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{B}'_1,f_3}(\lambda) + \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}\text{-}\mathsf{EA}}_{\Pi_4,\mathcal{B}'_2,f_4}(\lambda)\right)$$

Game 2: This game implements a selective AKE security game similar to Game 2 in the proof of Theorem 1. However, this time the adversary is not required to commit to a single test-session, but can chose one out of two.

Specifically, at the beginning of the game the adversary must output two pairs (U, i) and (V, j). The game then proceeds as in Game 1, except that if either of the following events occur, then the challenger "penalizes" the adversary by outputting a random bit at the end.

(i) Neither π_U^i nor π_V^j were selected as the test-session by $\mathcal{A}$.

(ii) π_U^i and π_V^j did not get partnered to each other.

Lemma 8. $\mathbf{Adv}^{\mathsf{G}_1}_{\Pi_5,\mathcal{A},f_5}(\lambda) \leq ({n_\pi}^2 \cdot |\mathcal{I}| \cdot |\mathcal{R}|)/2 \cdot \mathbf{Adv}^{\mathsf{G}_2}_{\Pi_5,\mathcal{A}',f_5}(\lambda)$.

Proof. Similar to the proof of Game 2 in Theorem 1 (Lemma 2). □

In the remaining games, let π_U^i and π_V^j denote the two sessions the adversary commits to in the selective security game.

Game 3: This game proceeds as the previous one, but when the first session out of π_U^i and π_V^j accepts in sub-protocol Π_3, then the challenger replaces its intermediate key K_{Π_3} with a random key $\widetilde{K}$. When the other session accepts in Π_3, the challenger replaces its intermediate key with the same random key $\widetilde{K}$.

Lemma 9. $\mathbf{Adv}^{\mathsf{G}_2}_{\Pi_5,\mathcal{A},f_5}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_3}_{\Pi_5,\mathcal{A},f_5}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{B}_1'',f_3}(\lambda)$.

Proof (sketch). From $\mathcal{A}$ we build an adversary $\mathcal{B}_1''$ against the AKE^w-security of the 3P-AKE protocol Π_3 as follows. $\mathcal{B}_1''$ begins by choosing a random bit b_{sim}. It then runs $\mathcal{A}$ and implements all the abort conditions introduced so far. All of $\mathcal{A}$'s Send queries that pertain to sub-protocol Π_3, $\mathcal{B}_1''$ forwards to its own 3P-AKE experiment.

When the first session out of π_U^i and π_V^j accepts in sub-protocol Π_3, $\mathcal{B}_1''$ makes a Test query to its own 3P-AKE experiment in order to obtain its session key k^* (which is either its actual key or a random key). $\mathcal{B}_1''$ uses k^* as that session's intermediate key K_{Π_3} in protocol Π_5. When the second session out of π_U^i and π_V^j accepts in sub-protocol Π_3, then $\mathcal{B}_1''$ again uses the key k^* as its intermediate key K_{Π_3}. For all sessions different from π_U^i and π_V^j $\mathcal{B}_1''$ obtains their intermediate keys by making a corresponding Reveal query to its own 3P-AKE experiment.

All of $\mathcal{A}$'s Send queries that pertain to sub-protocol Π_4, $\mathcal{B}_1''$ answers itself using the intermediate keys it obtained from sub-protocol Π_3 as the "long-term keys" for sub-protocol Π_4. In particular, $\mathcal{B}_1''$ derives the session keys of every session in sub-protocol Π_4 (and hence in protocol Π_5) itself. Using the session keys so-obtained, together with the bit b_{sim} it drew in the beginning of the simulation, $\mathcal{B}_1''$ can answer $\mathcal{A}$'s Test-query.

Finally, let b' be the output of $\mathcal{A}$. If $b' = b_{sim}$, then $\mathcal{B}_1''$ outputs 0. Otherwise, $\mathcal{B}_1''$ outputs 1.

We now analyze $\mathcal{B}_1''$. Recall that $\mathcal{B}_1''$ picks its test-session based on which of π_U^i and π_V^j accepted first in sub-protocol Π_3. We argue that if π_U^i is fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}}$, then the test-session chosen by $\mathcal{B}_1''$ is fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^w}$. This follows from the fact that π_U^i and π_V^j are f_5-partners (by Game 2), which implies that they are also f_3-partners. Thus, no matter the test-session chosen, $\mathcal{B}_1''$ will not make a Reveal query to neither the test-session nor its partner, since it will use the Test query instead. Moreover, since the selected test-session has a partner, it remains AKE^w-fresh even if its peers were corrupted.

Finally, we calculate $\mathcal{B}_1''$'s advantage. Let b denote the challenge-bit used in $\mathcal{B}_1''$'s own 3P-AKE experiment, and for $i \in \{2,3\}$, let "$G_i^{\mathcal{A}} \Rightarrow 1$" denote that $\mathcal{A}$ wins in Game 2 or Game 3, respectively.

If $b = 0$, then $\mathcal{B}_1''$'s Test query is answered with a real key. In this case $\mathcal{B}_1''$ simulates Game 2 perfectly for $\mathcal{A}$, thus

$$\Pr[\mathbf{Exp}^{\mathsf{3P\text{-}AKE}^w}_{\Pi,\mathcal{Q},\mathcal{B}_1''}(\lambda) \Rightarrow 1 \mid b = 0] = \Pr[G_2^{\mathcal{A}} \Rightarrow 1]. \tag{17}$$

On the other hand, if $b = 1$, meaning that $\mathcal{B}_1$'s Test query is answered with a random key, then $\mathcal{B}_1$ simulates Game 3 perfectly for $\mathcal{A}$, thus

$$\Pr[\mathbf{Exp}^{\mathsf{3P\text{-}AKE}^w}_{\Pi,\mathcal{Q},\mathcal{B}_1''}(\lambda) \Rightarrow 1 \mid b = 1] = 1 - \Pr[G_3^{\mathcal{A}} \Rightarrow 1]. \tag{18}$$

It follows that $\mathcal{B}_1''$'s advantage is

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{B}_1'',f_3}(\lambda) &= \Pr[\mathbf{Exp}^{\mathsf{3P\text{-}AKE}^w}_{\Pi,\mathcal{Q},\mathcal{B}_1''}(\lambda) \Rightarrow 1 \mid b = 0] \\
&\quad - \Pr[\mathbf{Exp}^{\mathsf{3P\text{-}AKE}^w}_{\Pi,\mathcal{Q},\mathcal{B}_1''}(\lambda) \Rightarrow 0 \mid b = 1] && (19)\\
&= \Pr[G_2^{\mathcal{A}} \Rightarrow 1] - \Pr[G_3^{\mathcal{A}} \Rightarrow 1] && (20)\\
&= \frac{1}{2} \cdot \mathbf{Adv}^{\mathsf{G}_2}_{\Pi_5,\mathcal{A},f_5}(\lambda) - \frac{1}{2} \cdot \mathbf{Adv}^{\mathsf{G}_3}_{\Pi_5,\mathcal{A},f_5}(\lambda), && (21)
\end{aligned}$$

which proves the lemma. □

Finally, we show that any successful attack on protocol Π_3 in Game 3 can be transformed into a successful attack on sub-protocol Π_4.However, before giving the details we explain why the most straightforward reduction idea does not work, and how the fact that the $\mathrm{AKE}^{\mathsf{static}}$ experiment allows for key-registration of long-term PSKs helps us overcome the problem. Suppose $\mathcal{A}$ is a successful adversary against protocol Π_5 in Game 3 and let $\mathcal{B}$ be the following reduction to sub-protocol Π_4. $\mathcal{B}$ begins by creating all the long-term keys for sub-protocol Π_3 and simulates every part of Π_3 itself. Furthermore, for all sessions except the test-session π_U^i and its partner π_V^j, $\mathcal{B}$ also simulates every part of sub-protocol Π_4 itself. In particular, since $\mathcal{B}$ simulated sub-protocol Π_3 itself, $\mathcal{B}$ knows all the intermediate keys K_{Π_3} and can thus use these as the long-term keys in sub-protocol Π_4. On the other hand, in order to simulate sub-protocol Π_4 for π_U^i and π_V^j, $\mathcal{B}$ instead forwards the corresponding queries to its own 2P-AKE

experiment. Once $\mathcal{A}$ stops and outputs a guess b', $\mathcal{B}$ stops and outputs the same bit b' to its 2P-AKE experiment.

It is easy to see that $\mathcal{B}$ defined in this way provides a perfect simulation of Game 3 for adversary $\mathcal{A}$. Consequently, $\mathcal{B}$ should intuitively win in its 2P-AKE experiment against sub-protocol Π_4 whenever $\mathcal{A}$ wins in Game 3. Unfortunately, this implication cannot be formally justified due to a technicality of our partner function framework. Namely, in the transcript generated from $\mathcal{B}$'s simulation of Game 3 for adversary $\mathcal{A}$, there are records of *many* sessions executing sub-protocol Π_4. However, in the corresponding transcript generated in $\mathcal{B}$'s own 2P-AKE experiment, there are only records from *two* sessions: the test-session π_U^i and its partner π_V^j. This is because for all other sessions $\mathcal{B}$ simulates sub-protocol Π_4 itself and does not forward these queries to its own 2P-AKE experiment. However, this means that when evaluating the partner function f_5 on the transcript from Game 3—which in particular includes running f_4 on the sub-transcript that pertains to sub-protocol Π_4—then this will involve *many* sessions, whereas when f_4 is evaluated on the transcript from $\mathcal{B}$'s 2P-AKE experiment, it involves only *two* sessions. Alas, without additional assumptions, we cannot conclude that these two evaluations of f_4 will necessarily give the same answers with respect to the test-session π_U^i.

The obvious solution to this is for $\mathcal{B}$ to forward *all* queries pertaining to sub-protocol Π_4 to its own 2P-AKE experiment; not only those targeting π_U^i and π_V^i. However, this faces another problem: the long-term PSKs used in $\mathcal{B}$'s 2P-AKE experiment are independent and uniformly distributed, but the intermediate keys K_{Π_3} derived in sub-protocol Π_3 might not be. Thus, if $\mathcal{B}$ forwards all queries pertaining to sub-protocol Π_4 to its own 2P-AKE experiment, then its simulation of Game 3 will be wrong. Note that forwarding the queries pertaining to π_U^i and π_V^i *is* actually justified since their intermediate key from sub-protocol Π_3 is replaced by an independent and uniformly distributed key $\widetilde{K}$ by the change in Game 3. But for all the other sessions this is not so.

This where we use the fact that the (2P-)AKE$^{\mathsf{static}}$ experiment allows for key-registration of long-term PSKs: for sessions different from π_U^i and π_V^i reduction $\mathcal{B}$ can register the intermediate keys from sub-protocol Π_3 as their long-term PSK in its 2P-AKE experiment. Since these sessions are not required to be fresh (according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^{\mathsf{static}}}$), it is not a problem that their PSKs are exposed. This change makes $\mathcal{B}$'s simulation of Game 3 perfect again. Additionally, it ensures that there is now a one-to-one correspondence between the sub-transcript corresponding to sub-protocol Π_4 in $\mathcal{B}$'s simulation of Game 3 and the transcript of sub-protocol Π_4 in $\mathcal{B}$'s 2P-AKE experiment. Thus, the previous issue of evaluating the partner function f_4 on incomparable transcripts is no longer present. We summarize the above in the following lemma.

Lemma 10. $\mathbf{Adv}^{\mathsf{G}_3}_{\Pi_5,\mathcal{A},f_5}(\lambda) \leq \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}}_{\Pi_4,\mathcal{B}_2'',f_4}(\lambda)$.

Proof (sketch). Reduction $\mathcal{B}_2''$ begins by creating all the long-term keys for sub-protocol Π_3, then runs $\mathcal{A}$. For all of $\mathcal{A}$'s Send queries that pertain to sub-protocol Π_3, $\mathcal{B}_2''$ answers itself using the keys it created. Once an initiator or responder

session π accepts in sub-protocol Π_3, then $\mathcal{B}_2''$ forwards all of $\mathcal{A}$'s remaining Send queries towards π to its own 2P-AKE$^{\mathsf{static}}$ experiment.

Specifically, once a session π accepts in in sub-protocol Π_3 then $\mathcal{B}_2''$ creates a corresponding "proxy" session in its own 2P-AKE$^{\mathsf{static}}$ experiment with a $\mathsf{NewSession}$ query. If π is one of the two selective security targets π_U^i and π_V^j, then $\mathcal{B}_2''$ uses a $\mathsf{NewSession}$ query *without* key registration. On the other hand, if π is different π_U^i and π_V^j, then $\mathcal{B}_2''$ uses a $\mathsf{NewSession}$ query *with* key registration, where the key that $\mathcal{B}_2''$ registers is the session key it derived for π in sub-protocol Π_3.

To answer $\mathcal{A}$'s $\mathsf{LongTermKeyReveal}$ queries, $\mathcal{B}_2''$ uses the long-term keys it created for sub-protocol Π_3. $\mathcal{A}$'s Reveal queries, as well as Test query, $\mathcal{B}_2''$ forwards to its own 2P-AKE$^{\mathsf{static}}$ experiment.

Finally, when $\mathcal{A}$ stops and outputs its guess b', then $\mathcal{B}_2''$ stops and outputs the same bit b' to its own 2P-AKE$^{\mathsf{static}}$ experiment.

Note that $\mathcal{B}_2''$ provides a perfect simulation of Game 3. Specifically, it provides a correct simulation of sub-protocol Π_3 since it created all the keys itself. Moreover, the simulation of sub-protocol Π_4 is also perfect, since the long-term keys used by the proxy sessions in $\mathcal{B}_2''$'s own 2P-AKE$^{\mathsf{static}}$ experiment are identically distributed to the "intermediate" keys in protocol Π_5. In particular, for sessions different from π_U^i and π_V^j the use of key registration ensures that the PSK used by the corresponding proxy sessions is exactly the same as intermediate keys derived in protocol Π_5. On the other hand, when $\mathcal{B}_2''$ creates the proxy sessions corresponding to π_U^i and π_V^j, it does so *without* key registration. By definition, this means that in $\mathcal{B}_2''$'s 2P-AKE$^{\mathsf{static}}$ experiment these proxy sessions will use an an independent, uniformly distributed PSK. However, by the change in Game 3, π_U^i and π_V^j's intermediate keys in protocol Π_5 are replaced with an independent, randomly distributed key $\widetilde{K}$, hence the distribution is the same.

It remains to argue that if π_U^i is a fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}}$ in Game 3, then it must also be fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^{\mathsf{static}}}$ in $\mathcal{B}_2''$'s 2P-AKE$^{\mathsf{static}}$ experiment as well. First of all, $\mathcal{B}_2''$ makes no $\mathsf{LongTermKeyReveal}$ queries to its own 2P-AKE$^{\mathsf{static}}$ experiment, since it answers $\mathcal{A}$'s $\mathsf{LongTermKeyReveal}$ queries with the long-term keys it created itself for sub-protocol Π_3. Secondly, if π_U^i is to be fresh in Game 3, then $\mathcal{A}$ cannot make a Reveal query to neither π_U^i nor π_V^j (since they are partners). Thus, if this is the case, then $\mathcal{B}_2''$ makes no such query to its $\mathcal{B}_2''$'s 2P-AKE$^{\mathsf{static}}$ experiment either. □

Concluding the proof of Theorem 2. Combining Lemma 7 to Lemma 10 we get

$$\begin{aligned}\mathbf{Adv}^{\mathsf{3P\text{-}AKE}}_{\Pi_5,\mathcal{A},f_5}(\lambda) \leq\ & 2n^2 \cdot \mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{B}_1',f_3}(\lambda) + n^2 \cdot \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}}_{\Pi_4,\mathcal{B}_2',f_4}(\lambda) \\ & + 2n^2 \cdot \mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{B}_1'',f_3}(\lambda) + n^2 \cdot \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}}_{\Pi_4,\mathcal{B}_2'',f_4}(\lambda)\end{aligned} \tag{22}$$

where $n = (n_\pi + 1) \cdot |\mathcal{I} \cup \mathcal{R}|$.

Letting $\mathcal{B}_1$ be the 3P-AKE adversary that with probability 1/2 either implements algorithm $\mathcal{B}'_1$ or algorithm $\mathcal{B}''_1$, and letting $\mathcal{B}_2$ be the 2P-AKE$^{\mathsf{static}}$ adversary that either implements algorithm $\mathcal{B}'_2$ or algorithm $\mathcal{B}''_2$, the concrete bound (14) in the statement of Theorem 2 follows. □

4 Security of EAP

In this section we explore the security of EAP. As mentioned in the introduction, there is no single definitive version of EAP which we can use for this purpose since the specification itself (RFC 3748 [4]) leaves many of its components undefined. Consequently, any analysis of EAP will have to make assumptions on these components.

In this paper we model the "cryptographic core" of EAP as essentially being the generic construction given in Section 3.1. That is, we identify the standalone EAP without any subsequent key-confirmation step with the protocol construction Π_3 shown in Fig. 5. Thus, by suitably instantiating the different building-blocks that make up this construction (see Section 4.1 for details), Theorem 1 immediately applies to standalone EAP—assuming it uses channel binding. Unfortunately, channel binding is not mandated in the EAP standard, and many defined EAP methods do not require it. Section 4.3 describes some of the problems that this omission might lead to.

4.1 EAP with channel binding

In Theorem 1, let us identify sub-protocol Π_1 with the EAP method run between the client and the server. Let sub-protocol Π_2 be the key-transport protocol run between the server and the authenticator. Finally, suppose that EAP employs the channel binding mechanism defined in [38]. Then we immediately get the following result for standalone EAP.

Theorem 3 (3P-AKE^w security of EAP)**.** *If the chosen EAP method used within EAP is 2P-AKE-secure, the key-transport protocol is ACCE-secure, and the employed key-derivation function is a secure PRF that provides channel binding on the client's and authenticator's identities, then EAP is 3P-AKE^w-secure.*

To be even more concrete, we can also instantiate sub-protocols Π_1 and Π_2 with some actual real-world protocols. For example, Brzuska et al. [14] recently showed that the EAP-TLS method constitutes a secure 2P-AKE protocol, thus satisfying the requirements on sub-protocol Π_1. For sub-protocol Π_2 we can pick any AAA protocol that is run over TLS, like RADIUS-over-TLS [45] or Diameter [18], which then reduces the problem to showing that TLS is an ACCE-secure protocol. Fortunately, multiple papers [24, 30, 26, 36, 11] have already done this so taking sub-protocol Π_2 to be either RADIUS-over-TLS or Diameter satisfies the requirement of Theorem 3.

4.2 Channel-binding scope

In Theorem 1, and Theorem 3, we assumed that the channel binding mechanism included the identity of the client and the authenticator in order to bind their identities cryptographically to the session key. Implicitly, this also assumes that all identities are globally unique and belong to the same namespace. While this is a standard assumption when doing cryptographic modeling, in reality the various links in EAP take place over different types of communication media with different types of identities and addressing schemes. For instance, when EAP is used in combination with IEEE 802.11, the communication between the client and the access point is based on link-layer addresses, the communication between the client and the server is typically based on usernames (client) and domain names (server), while the communication between the server and the access point might be based solely on IP addresses. As pointed out in [23], mapping between these identifiers is not always straightforward. In fact, some of the identifiers might not even be available to all the protocol participants. Specifically, since the communication between the client and the access point happens at the link-layer, the IP addresses used by the access point towards the server might not be available to the client unless the access point broadcasts it. In practice, most link-layer protocols have facilities to provide this kind of information to the client[10], but there is no guarantee that the authenticator will actually provide it.

Moreover, in some settings this information may not even be relevant. For example, in a WLAN supported by many access points, the client might not actually care about *which* specific access point it connects to, as long as it connects to a legitimate access point of that WLAN. Thus, in this case the granularity of the channel-binding doesn't need to be at the individual access point level, but rather at the WLAN level, defined by all the access points broadcasting the same network identifier (SSID). Of course, by doing so the security guarantees provided by the channel-binding will be weaker. Specifically, when channel-binding occurs at the individual level, then the corruption of a single access point will not influence clients connecting to access points having a *different* identity. On the other hand, when channel-binding occurs at the network level, then a single corrupted access point will affect *all* connections within that WLAN. In this case, the channel binding only protects connections occurring in networks having a different SSID.

More generally, the information included in the channel-binding defines the scope of the protection it provides, and can include more than just identities. For instance, physical media types, data rates, cost-information, channel frequencies, etc., can all be used as input to the channel-binding (see [16] for a discussion of these possibilities). The specifications for channel-binding within EAP [38, 21] leaves open exactly the type of information that should go into the binding, because the amount of information that will be available to both the client and the server can vary.

[10] For instance, the `Identity` type field in EAP `Request` messages is often "piggybacked" by link-layer protocols to include this type of information; see e.g., EAPOL/802.1X [1].

4.3 EAP without channel binding

Since the "cryptographic core" of EAP is modeled by protocol Π_3 in Section 3.1, if EAP does not employ channel binding it is vulnerable to exactly the same UKS attack that was described for protocol Π_3 without channel binding in Remark 9. Namely, since the identities being communicated from the client to the server are without any integrity protection (see the A–S link in Fig. 5), an adversary can modify them so that the server will distribute the derived session key to the wrong authenticator. Less abstractly, this attack is an instance of the *lying authenticator problem*: since the communication between the client and server is normally routed through the authenticator, this allows the authenticator to easily modify the information presented to the two sides. Thus, without channel binding it suffices to compromise a single authenticator in order to compromise an entire network. Moreover, since authenticators are typically low-protected devices, such as wireless access points, the lying authenticator problem is a substantial attack vector on enterprise networks. As explained in the previous section, even if the channel binding only included the network name, it would clearly be an upgrade over EAP without channel binding, and comes at essentially no cost.

Interestingly, a situation very similar to that of EAP without channel binding can be found in the UMTS and LTE mobile networks. In particular, UMTS and LTE employ a key exchange protocol called AKA which is structured almost identically to the EAP protocol[11]: a mobile client that wants to connect to a base station first has to authenticate to its home operator. The home operator then transmits so-called *authentication vectors* (which in particular includes a session key) to the base station in much the same way as the server forwards the session key to the authenticator in EAP. Moreover, similar to many EAP methods, the AKA protocol also lacks channel binding for its authentication vectors. In their recent analysis of the AKA protocol, Alt et al. [5, §5] noted this lack of channel-binding, and suggested a fix which is essentially identical to the key-derivation approach analyzed in this paper.

5 Security of IEEE 802.11

5.1 Description of the IEEE 802.11 protocol

IEEE 802.11 [2] is the most widely used standard for creating WLANs. It supports three modes of operation depending on the network topology: infrastructure mode, ad-hoc mode, and mesh network mode. In ad-hoc mode and mesh-networking mode there is no central infrastructure, and the wireless clients talk directly to each other. On the other hand, in infrastructure mode the clients only communicate through an access point, which usually also provides connectivity to a larger WAN. In this paper we only cover IEEE 802.11 in infrastructure mode which is by far the most common mode.

[11] In fact, EAP is widely used within mobile networks.

The IEEE 802.11 protocol is a link-layer protocol, aiming to secure the wireless link between the client and the access point. It defines two main security protocols: the *4-Way-Handshake (4WHS)*, used to authenticate and establish session keys between the client and the access point; and the *Counter Mode CBC-MAC protocol (CCMP)*, used to secure the actual application data. We will only cover the 4WHS in this paper.

The 4WHS is based on a symmetric *Pairwise Master Key (PMK)* shared between the client and the access point. The PMK can either be pre-configured at the client and access point or distributed through some other means, like for instance EAP. The first alternative is most typically found in wireless home networks where a static PMK is manually configured at the access point and at every connecting device.[12] This variant is also commonly referred to as WPA2-PSK. The second alternative, often referred to as WPA2-Enterprise, is normally used in large organization like universities and big companies where there are many users and access points. In this setting it is infeasible for every user and access point to share the same PMK. Instead, a central authentication server is used to manage authentication as well as distributing new PMKs for every established session. Usually the protocol used to access the authentication server is EAP.

In Section 5.2 we analyze the pre-shared key variant of the 4WHS, while in Section 5.3 we analyze it when combined with EAP.

5.2 Analyzing the 4-Way-Handshake

The 4WHS is shown in Figure 6. It depends on a pseudorandom function PRF and a MAC scheme $\Sigma = (\mathsf{MAC}, \mathsf{Vrfy})$; see Appendix A for their formal definitions. We use the notation $[x]_k \stackrel{\text{def}}{=} x \| \sigma$ to denote a message x together with its MAC tag $\sigma \leftarrow \mathsf{MAC}(k, x)$. Identities in the 4WHS are based on the parties' 48-bit link-layer addresses which makes it possible to compare them based on their corresponding numerical values. Particularly, the functions $\max\{A, B\}$ and $\min\{A, B\}$ returns, respectively, the largest and the smallest of two link-layer addresses A and B when interpreted as 48-bit integers.

In our modeling we will mostly ignore the exact encoding of the IEEE 802.11 packets as used by the 4WHS. For our purposes it sufficient to model them as consisting of a nonce plus a fixed constant p_i that uniquely determines each handshake message m_i. If a received message does not match the expected format, including the value of the constant p_i, it is silently discarded. The 4WHS proceeds as follows:

1. The 4WHS begins with the access point AP sending the message $m_1 = \eta_{AP} \| p_1$ to the client C, where η_{AP} is a nonce and p_1 a constant.

2. On receiving m_1, C generates its own nonce η_C and derives a so-called *pairwise transient key (PTK)* using the pseudorandom function PRF and

[12] Usually the PMK is not configured directly, but instead derived from a password using a password-based KDF. We ignore this distinction here.

the long-term key it shares with AP. Specifically, $\text{PTK} \stackrel{\text{def}}{=} k_\mu \| k_\alpha \leftarrow \mathsf{PRF}_K(P\|\eta)$, where $P\|\eta = \min\{AP, C\}\|\max\{AP, C\}\|\min\{\eta_{AP}, \eta_C\}\|\max\{\eta_{AP}, \eta_C\}$. The sub-key k_α will be the session key eventually output by the client, while the sub-key k_μ will be used in the MAC Σ to protect the handshake messages. After deriving PTK, C creates and sends the next protocol message $m_2 = [\eta_C \| p_2]_{k_\mu}$.

3. On receiving $m_2 = [\eta_C \| p_2]_{k_\mu}$, AP uses the containing nonce η_C to derive $k_\mu \| k_\alpha \leftarrow \mathsf{PRF}_K(P\|\eta)$. With the sub-key k_μ, it verifies the MAC tag of m_2. If the verification succeeds, then AP stores $\text{PTK} \leftarrow k_\mu \| k_\alpha$ as its PTK and sends out the third protocol message $m_3 = [\eta_{AP} \| p_3]_{k_\mu}$. If the verification fails, then AP silently discards m_2, as well as the derived keys k_μ, k_α, and continues running.

4. On receiving $m_3 = [\eta'_{AP} \| p_3]_{k_\mu}$, C first verifies its MAC tag with key k_μ, and checks that η'_{AP} equals the nonce η_{AP} that C previously received in message m_1. If the verification succeeds then C sends out the final handshake message $m_4 = [p_4]_{k_\mu}$. Additionally, it sets its own acceptance state to $\alpha = \mathsf{accepted}$. If the verification fails, C silently discards m_3 and continues running.

5. On receiving m_4, AP verifies its MAC tag using the key k_μ. If the verification succeeds, it sets its own acceptance state to $\alpha = \mathsf{accepted}$. If the verification fails, AP silently discards m_4 and continues running.

Remark 10. An adversary can freely modify message m_1 since it has no integrity protection. However, since every recipient of an m_1 message will check that it matches the excepted format of "$x\|p_1$", the adversary is in reality limited to only modifying the value of the nonce. Of course, this is a simplification compared to the real IEEE 802.11 header, where there are actually multiple different bit fields which the adversary could manipulate—in principle. Still, the fact is that except for the nonce η_{AP}, all bit fields in the IEEE 802.11 header of the first m_1 message have pre-determined values. Thus, the attacker does not have more opportunities to manipulate the real IEEE 802.11 m_1 message as opposed to in our simplified modeling of it.

On the other hand, in the IEEE 802.11 header of messages m_2, m_3 and m_4, there *are* bit fields that the adversary could potentially influence. But since these messages are protected by a MAC, the adversary will be unable to modify them. Whether we model p_2, p_3 and p_4 as constants or as arbitrary distinct values makes no difference for our analysis.

Remark 11. The fourth handshake message m_4 serves no cryptographic purpose and could safely have been omitted. However, to stay true to the actual 4WHS, we leave it in.

AKE$^{\text{static}}$-security. We begin by proving that the 4WHS constitutes a secure 2P-AKE in the AKE$^{\text{static}}$ model. Following that, we show that it also achieves

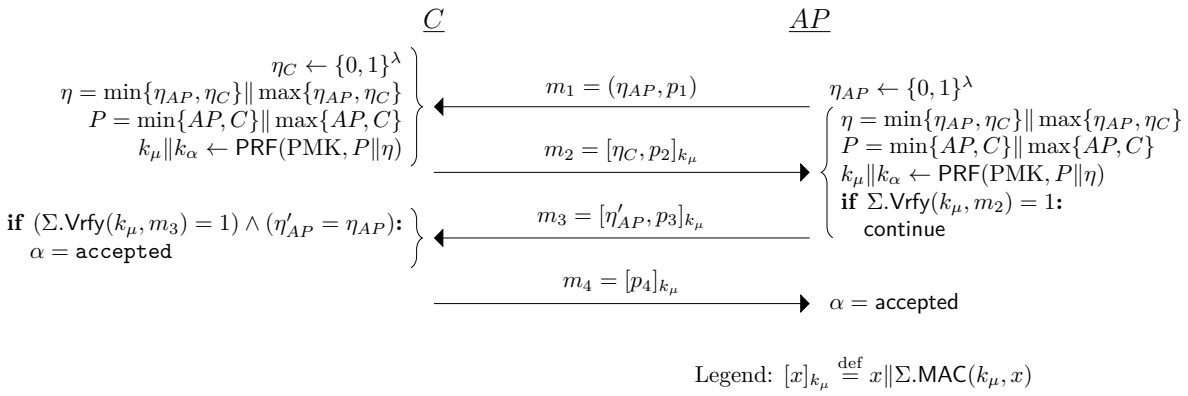

Figure 6: The IEEE 802.11 4-Way-Handshake protocol. The client C and the access point AP share a long-term symmetric key PMK.

explicit entity authentication. In the following, let $\mathcal{P}_{AP} = \mathcal{I}$ and $\mathcal{P}_C = \mathcal{R}$, i.e., in the 4WHS protocol the access point has the initiator role while the client has the responder role. According to the IEEE 802.11 standard, each client and access point is allowed to share multiple long-term PMKs with each other. In the following analysis we make the simplifying assumption that every client–access point pair only shares a single PMK.

Theorem 4. *The 4WHS protocol is* $\mathrm{AKE}^{\mathsf{static}}$*-secure. In particular, for any PPT adversary* $\mathcal{A}$*, there exists a partner function* f *and an algorithm* $\mathcal{D}$*, such that*

$$\mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \leq 2 \cdot |\mathcal{P}_C| \cdot |\mathcal{P}_{AP}| \cdot \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF}}(\mathcal{D}) + \frac{(n_P n_\pi)^2}{2^{\lambda+1}}, \tag{23}$$

where n_π *is the number of sessions at each party, and* $n_P = |\mathcal{P}_C| + |\mathcal{P}_{AP}|$*.*

Proof. Recall that in the $\mathrm{AKE}^{\mathsf{static}}$ model the adversary is allowed to register the PMK a session will use when creating it via the $\mathsf{NewSession}$ query. Of course, in this case the session's PMK will be considered exposed and the session will thus not be fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^{\mathsf{static}}}$.

Defining the partner function f. For the analysis of the 4HWS it would be natural to use SIDs as the partnering mechanism. Namely, the SID of a session π would be the string $P \| \eta$ that π input to its PRF in order to create its session key (see Fig. 6).[13] However, because our paper is phrased in terms of partnering functions, we "synthetically" encode the SID as a partnering function by saying that π's partner session is the *first* session—different from π—that sets the same SID as π. Taking the first one is important because a partner function is a function and not a relation.

[13]For an access point the SID would only be set if the verification of the received m_2 message succeeded.

In more detail, suppose π_{AP}^i is an access point session having $C \in \mathcal{P}_C$ as its intended peer. If π_{AP}^i itself created the nonce η_{AP} for message m_1, and later successfully verified an incoming m_2 message containing the nonce η_C, then $f_T(\pi_{AP}^i) = \pi_C^j$ if and only if (1) π_C^j has AP as its indented peer and (2) π_C^j was the first session at C that used the nonces η_C and η_{AP} to derive its PTK.

Similarly, suppose π_C^i is a client session having $AP \in \mathcal{P}_{AP}$ as its intended peer. If π_C^i used the nonces η_C and η_{AP} to derive its PTK after receiving message $m_1 = (\eta_{AP} \| p_1)$, then $f_T(\pi_C^i) = \pi_{AP}^j$ if and only if (1) π_{AP}^j has C as its intended peer, (2) π_{AP}^j created the nonce η_{AP} and (3) π_{AP}^j was the first session at AP that successfully verified an m_2 message containing the nonce η_C.

The soundness of f is immediate from its definition and PRF being a deterministic function.

Game 0: This is the real 2P-AKE security game, hence

$$\mathbf{Adv}^{\mathsf{G}_0}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) = \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \ .$$

Game 1: This game proceeds as the previous one, but aborts if not all the nonces in the game are distinct, hence

$$\mathbf{Adv}^{\mathsf{G}_0}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_1}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) + \frac{(n_P n_\pi)^2}{2^{\lambda+1}} \ . \tag{24}$$

Game 2: This game implements a selective AKE security game where at the beginning of the game the adversary has to "commit" to the pre-shared PMK that will be used by the test-session.

Specifically, at the beginning the of the game, the adversary has to output two party identities $C \in \mathcal{P}_C$ and $AP \in \mathcal{P}_{AP}$. The game then proceeds as in Game 1, except that it aborts if the test-session selected by the adversary did not use the PMK shared between C and AP.

Lemma 11. $\mathbf{Adv}^{\mathsf{G}_1}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \leq |\mathcal{P}_{AP}| \cdot |\mathcal{P}_C| \cdot \mathbf{Adv}^{\mathsf{G}_2}_{\mathsf{4WHS},\mathcal{A}',f}(\lambda)$.

Proof. From an adversary $\mathcal{A}$ that wins against the adaptive game in Game 1, we create an adversary $\mathcal{A}'$ that wins against the selective game in Game 2 as follows. First, $\mathcal{A}'$ randomly selects two party identities $C \in \mathcal{P}_C$ and $AP \in \mathcal{P}_{AP}$. It outputs C and AP as its choice to the selective security game it is playing. $\mathcal{A}'$ then runs $\mathcal{A}$ and answers all of its queries by forwarding them to its own selective security game. When $\mathcal{A}$ stops with output b', then $\mathcal{A}'$ stops and outputs the same bit as well.

Algorithm $\mathcal{A}'$ perfectly simulates Game 1 for $\mathcal{A}$, so $\mathcal{A}'$'s choice of selective security targets matches those of $\mathcal{A}$ with probability at least $1/(|\mathcal{P}_{AP}| \cdot |\mathcal{P}_C|)$. When $\mathcal{A}'$'s guess is correct it wins with the same probability as $\mathcal{A}$, while when it is wrong, $\mathcal{A}'$ gets penalized in its selective security game, hence wins with probability $1/2$. □

In the remainder of the proof, let C and AP denote the parties that the adversary commits to in Game 2, and let PMK^* denote the PMK shared between them. Note that by the requirements of the $\mathsf{Fresh}_{\mathsf{AKE}^{\mathsf{static}}}$ predicate (Fig. 3), PMK^* cannot be exposed if the test-session is to be fresh. In particular, this means that the adversary cannot make a $\mathsf{LongTermKeyReveal}(C, AP)$ query, nor create the test-session via a $\mathsf{NewSession}$ query where it registers PMK^* as its long-term key.

Game 3: In this game the challenger replaces the pseudorandom function PRF with a random function $\$(\cdot)$ in all evaluations involving the pre-shared key PMK^*. That is, calls of the form $\mathsf{PRF}(\text{PMK}^*, \cdot)$ are instead answered by $\$(\cdot)$.

Lemma 12. $\mathbf{Adv}^{\mathsf{G}_2}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_3}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{D}}(\lambda)$.

Proof. Algorithm $\mathcal{D}$ has access to an oracle $\mathcal{O}$, which either implements the function $\mathsf{PRF}(\widetilde{\text{PMK}}, \cdot)$ for some independently and uniformly distributed key $\widetilde{\text{PMK}}$, or it implements a truly random function $\$(\cdot)$. $\mathcal{D}$ begins by choosing a random bit b_{sim} and creating all the PMKs for all client-access points pairs different from the selective security targets C and AP. It then runs $\mathcal{A}$.

For all of $\mathcal{A}$'s queries that does not involve computations with the PMK of C and AP, $\mathcal{D}$ answers itself using the keys it created. On the other hand, for queries that would normally involve computations with the PMK of C and AP, algorithm $\mathcal{D}$ uses its oracle $\mathcal{O}$ to do these computations, and the answers the queries accordingly. Finally, when $\mathcal{A}$ stops with some output b', then $\mathcal{D}$ stops and outputs 0 to its PRF experiment if $b' = b_{sim}$, and 1 otherwise.

When $\mathcal{O} = \mathsf{PRF}(\widetilde{\text{PMK}}, \cdot)$, then $\mathcal{D}$ perfectly simulates Game 2 since the PMKs are chosen independently and uniformly at random; while when $\mathcal{O} = \$(\cdot)$, then $\mathcal{D}$ perfectly simulates Game 3. The lemma follows. □

Concluding the proof of Theorem 4. Suppose the test-session in Game 3 accepted with the "SID" $P\|\eta$. By Game 1 we know that the only sessions that evaluated the pseudorandom function on this SID was the test-session and possibly its partner. However, by Game 3 the PRF is now a truly random function which is unavailable to the adversary provided the test-session is to remain $\text{AKE}^{\mathsf{static}}$-fresh. In particular, this means that the PTK derived by the test-session (and possibly its partner) is a truly random string $\widetilde{\text{PTK}} = \widetilde{k_\mu}\|\widetilde{k_\alpha} \leftarrow \{0,1\}^{2\lambda}$, and where $\widetilde{k_\alpha}$ is independent of all other values. Thus, $\mathbf{Adv}^{\mathsf{G}_3}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) = 0$, and Theorem 4 follows. □

Explicit entity authentication. We now prove that the 4WHS also provides explicit entity authentication. The proof of this fact follows the same outline as for the key-indistinguishability part of the proof, using essentially the same game hops. However, instead of bounding the key-indistinguishability advantage of the adversary in the final game, we instead bound the probability that

a session will accept maliciously. We do this by reducing to the SUF-CMA security of the MAC Σ (see Appendix A for a formal definition of the SUF-CMA notion). We point out that we are using a variant of SUF-CMA that provides the adversary with multiple verification attempts, rather than just a single one at the end. While single-verification and multi-verification are not equivalent in the more standard UF-CMA setting, they *are* equivalent in the stronger SUF-CMA setting; see [6]. Moreover, for message authentication *codes*—as opposed to *message authentication* schemes in general—UF-CMA security implies SUF-CMA security for MAC schemes. Since the IEEE 802.11 4WHS only uses MACs and not general message authentication schemes, using the SUF-CMA assumption is justified. More specifically, the MACs standardized for IEEE 802.11 are HMAC [29] and CMAC [17].

Theorem 5. *The 4WHS provides explicit entity authentication. In particular, for any PPT adversary $\mathcal{A}$, there exists algorithms $\mathcal{D}$ and $\mathcal{F}$, such that*

$$\begin{aligned}\mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}\text{-}\mathsf{EA}}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \leq 2 \cdot |\mathcal{P}_C| \cdot |\mathcal{P}_{AP}| \cdot \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{D}}(\lambda) + \frac{(n_P n_\pi)^2}{2^{\lambda+1}} \\ + 2n_\pi \cdot (q+1) \cdot |\mathcal{P}_C| \cdot |\mathcal{P}_{AP}| \cdot \mathbf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\Sigma,\mathcal{F}}(\lambda),\end{aligned} \tag{25}$$

where f, n_π, and n_P are the same as in Theorem 4, and q is an upper bound on the number of m_2 message $\mathcal{A}$ sends to an access point session.

Proof. The initial part of the proof proceeds through essentially the same three game hops as in the proof of Theorem 4.

Game 0: This is the real security experiment, hence

$$\mathbf{Adv}^{\mathsf{G}_0}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) = \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}\text{-}\mathsf{EA}}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \ .$$

Game 1: This game proceeds as the previous one, but aborts if not all the nonces in the game are distinct, hence

$$\mathbf{Adv}^{\mathsf{G}_0}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_1}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) + \frac{(n_P n_\pi)^2}{2^{\lambda+1}} \ . \tag{26}$$

Game 2: This game implements a selective security game similar to the one in Game 2 of Theorem 4. However, rather than committing to its test-session target, this time the adversary is required to commit to the PMK used by the first session that accepts maliciously.

Specifically, at the beginning the of the game, the adversary has to output two party identities $C \in \mathcal{P}_C$ and $AP \in \mathcal{P}_{AP}$. The game then proceeds as in Game 1, except that it aborts if the first session that accepts maliciously (if any) did not use the PMK shared between C and AP. By essentially the same arguments as for Game 2 of Theorem 4 (Lemma 11), we have

$$\mathbf{Adv}^{\mathsf{G}_1}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \leq |\mathcal{P}_{AP}| \cdot |\mathcal{P}_C| \cdot \mathbf{Adv}^{\mathsf{G}_2}_{\mathsf{4WHS},\mathcal{A}',f}(\lambda) \ . \tag{27}$$

In the remainder of the proof, let C and AP denote the parties that the adversary commits to in the selective stage of Game 2, and let PMK^* denote the corresponding PMK shared between them.

Game 3: In this game the challenger replaces the pseudorandom function PRF with a random function $\$(\cdot)$ in all evaluations involving the pre-shared key PMK^*. That is, calls of the form $\mathsf{PRF}(\text{PMK}^*, \cdot)$ are instead answered by $\$(\cdot)$. By the same arguments as for Game 3 of Theorem 4 (Lemma 12), we have

$$\mathbf{Adv}^{\mathsf{G}_2}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) \leq \mathbf{Adv}^{\mathsf{G}_3}_{\mathsf{4WHS},\mathcal{A},f}(\lambda) + 2 \cdot \mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{D}}(\lambda). \tag{28}$$

Finally, we conclude the proof of Theorem 5 by showing that if the adversary can get a session to accept maliciously in Game 3, then we can create an adversary $\mathcal{F}$ that can create forgeries against the MAC Σ.

Lemma 13. $\mathbf{Adv}^{\mathsf{G}_3\text{-}\mathsf{EA}}_{\mathsf{4WHS},f,\mathcal{A}}(\lambda) \leq 2n_\pi(q+1) \cdot \mathbf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\Sigma,\mathcal{F}}(\lambda)$.

Proof. The forger $\mathcal{F}$ has access to two oracles, $\mathcal{O}^{\mathsf{MAC}}(\cdot)$ and $\mathcal{O}^{\mathsf{Vrfy}}(\cdot,\cdot)$, which implements the functions $\Sigma.\mathsf{MAC}(\widetilde{k},\cdot)$ and $\Sigma.\mathsf{Vrfy}(\widetilde{k},\cdot,\cdot)$ for some independent uniformly distributed key $\widetilde{k}$.

Algorithm $\mathcal{F}$ begins by drawing a random bit b_{sim} and waits for $\mathcal{A}$ to output a pair (C, AP) as its selective security target. Based on $\mathcal{A}$'s selected target, $\mathcal{F}$ randomly guesses a pair $(U^*, i) \twoheadleftarrow \{C, AP\} \times [1, n_\pi]$. For every client–access point pair except (C, AP), $\mathcal{F}$ creates all the corresponding long-term PMKs itself. Finally, $\mathcal{F}$ runs $\mathcal{A}$ in its second stage, simulating Game 3 as follows.

For sessions not between C and AP, $\mathcal{F}$ simulates everything itself using the PMKs it created. For sessions between C and AP, $\mathcal{F}$ also simulates everything itself, but this time by implementing a random function $\$(\cdot)$ rather than the function $\mathsf{PRF}(\text{PMK}^*,\cdot)$. Additionally, for session $\pi^i_{U^*}$, $\mathcal{F}$ will embed its MAC oracles $\mathcal{O}^{\mathsf{MAC}}$ and $\mathcal{O}^{\mathsf{Vrfy}}$ into some of the computations that would normally involve using the MAC Σ with the PTK that was derived by $\pi^i_{U^*}$ in the handshake. Specifically, depending on whether $\pi^i_{U^*}$ belongs to the client or the access point, $\mathcal{F}$ simulates $\pi^i_{U^*}$ using the procedures labeled SimC or SimAP in Fig. 7, respectively. Below we provide some more intuition and explanation for these procedures.

In both SimC and SimAP some of $\pi^i_{U^*}$'s MAC computations are "outsourced" to $\mathcal{F}$'s MAC oracles $\mathcal{O}^{\mathsf{MAC}}$ and $\mathcal{O}^{\mathsf{Vrfy}}$. However, in order to maintain a consistent simulation, $\mathcal{F}$ needs to handle the situation where some session at the intended peer of $\pi^i_{U^*}$ derives the same PTK as $\pi^i_{U^*}$. That is, if $\mathcal{F}$ embeds its MAC oracles into a computation at $\pi^i_{U^*}$ that would normally involve the key PTK, then $\mathcal{F}$ must also embed the MAC oracles into any session at $\pi^i_{U^*}$'s intended peer that derives the same PTK. Since $\pi^i_{U^*}$ and the other session need to input the same nonces to the random function $\$(\cdot)$ in order to derive the same PTK, we call this situation *nonce-forwarding*.

Particularly, if $\pi^i_{U^*}$ belongs to the access point AP, then nonce-forwarding means that $\mathcal{A}$ forwarded the nonce η_{AP} contained in $\pi^i_{U^*}$'s m_1 message to some

session at the client C, say π_C^j, and then forwarded the nonce η_C contained in π_C^j's responding m_2 message back to $\pi_{U^*}^i$ (see Fig. 6). In this case $\pi_{U^*}^i$ and π_{AP}^j would derive the same PTK since they use the same nonces as input to the random function $\$(\cdot)$. Furthermore, note that when $\pi_{U^*}^i$ belongs to the access point AP, then the adversary can potentially send many m_2 messages to $\pi_{U^*}^i$ since a message that fails verification will simply be discarded while $\pi_{U^*}^i$ continues running. Thus, $\mathcal{A}$ can potentially be nonce-forwarding between $\pi_{U^*}^i$ and multiple sessions at the client C, since $\mathcal{A}$ can forward $\pi_{U^*}^i$'s initial m_1 message to many sessions at C. This is handled by the list Fwd in the SimAP code.

If $\pi_{U^*}^i$ belongs to the client C, then nonce-forwarding means that $\pi_{U^*}^i$ received a nonce η_{AP} in message m_1 that was previously produced by some session π_{AP}^j at the access point AP, and $\mathcal{A}$ subsequently forwards the nonce η_C contained in $\pi_{U^*}^i$'s m_2 response back to π_{AP}^j. Unlike in the SimAP case, when $\pi_{U^*}^i$ belongs to the client C then $\mathcal{A}$ can be nonce-forwarding to at most one session at the access point AP. This follows because we have assumed that all nonces are unique by Game 1. This (possibly non-existent) session is recorded in the variable π_{AP}^* in the SimC code.

Given the definition of nonce-forwarding above, the procedures SimC and SimAP can be explained as follows.

- SimC: When $\pi_{U^*}^i$ belongs to the client C, then $\mathcal{F}$ waits until it receives the first protocol message $m_1 = (\eta_{AP}, p_1)$.

  Line 200–207: On receiving m_1, $\mathcal{F}$ first generates a random nonce η_C^*, and creates $\pi_{U^*}^i$'s response message m_2 by querying its MAC oracle on $\eta_C^* \| p_2$, i.e., $m_2 = \eta_C^* \| p_2 \| \tau$, where $\tau \leftarrow \mathcal{O}^{\mathsf{MAC}}(\eta_C^* \| p_2)$.

  After sending out m_2, $\mathcal{F}$ waits until $\pi_{U^*}^i$ receives a protocol message m_3. Recall that a message which doesn't pass verification is silently dropped while the session continues running. Thus, there could potentially be many such m_3 "attempts" made by $\mathcal{A}$.

  Line 300–311: When receiving an m_3 attempt, $\mathcal{F}$ first checks that it contains the same nonce η_{AP}^* that $\pi_{U^*}^i$ previously received in message m_1. If so, then $\mathcal{F}$ forwards m_3 to its MAC verification oracle. If $\mathcal{O}^{\mathsf{Vrfy}}(m_3) = 1$, then $\mathcal{F}$ stops in its SUF-CMA experiment. Otherwise, it silently discards m_3 and continues the simulation for $\mathcal{A}$.

  Line 400–407: Finally, if $\mathcal{A}$ was nonce-forwarding between $\pi_{U^*}^i$ and some session π_{AP}^j at the access point AP. Then $\mathcal{F}$ uses the MAC verification oracle to verify the m_2 attempt.

- SimAP: As explained above, when $\pi_{U^*}^i$ belongs to the access point AP, the adversary can potentially make many m_2 message "attempts" to $\pi_{U^*}^i$. Furthermore, these m_2 "attempts" might also contain different nonces. Let q be an upper bound on the number of *distinct* nonces contained in these m_2 attempts. The total number of m_2 attempts can of course be higher than q, since $\mathcal{A}$ might repeat a nonce across multiple attempts.

SimC:

100: Initialization:
101: $\eta^*_C, \eta^*_{AP} \leftarrow \bot$
102: $\pi^*_{AP} \leftarrow \bot$

200: On receiving m_1:
201: parse m_1 as (η_{AP}, p_1)
202: $\eta^*_{AP} \leftarrow \eta_{AP}$
203: **if** some session π^j_{AP} created η_{AP}**:**
204: $\pi^*_{AP} \leftarrow \pi^j_{AP}$
205: $\eta^*_C \twoheadleftarrow \{0,1\}^\lambda$
206: $m_2 \leftarrow \mathcal{O}^{\mathsf{MAC}}(\eta^*_C \| p_2)$
207: send m_2

300: On receiving an m_3 attempt:
301: parse m_3 as $\eta_{AP} \| p_3 \| \tau$
302: **if** $\eta_{AP} \neq \eta^*_{AP}$**:**
303: discard m_3
304: continue simulation
305: **else**
306: $d \leftarrow \mathcal{O}^{\mathsf{Vrfy}}(\eta_{AP} \| p_3, \tau)$
307: **if** $d = 1$**:**
308: stop simulation
309: **else**
310: discard m_3
311: continue simulation

400: On nonce-forwarding η^*_C to π^*_{AP}:
401: parse m_2 attempt as $\eta^*_C \| p_2 \| \tau$
402: $d \leftarrow \mathcal{O}^{\mathsf{Vrfy}}(\eta^*_C \| p_2, \tau)$
403: **if** $d = 1$**:**
404: stop simulation
405: **else**
406: discard m_2
407: continue simulation

SimAP:

100: Initialization:
101: $\eta^*_{AP} \leftarrow \bot$
102: $\vec{\mathsf{N}} \leftarrow \emptyset$
103: $\mathsf{Fwd} \leftarrow \emptyset$
104: $\mathsf{distinct} \leftarrow 0$
105: $q^* \twoheadleftarrow [1, q]$

200: Create m_1:
201: $\eta^*_{AP} \twoheadleftarrow \{0,1\}^\lambda$
202: $m_1 \leftarrow \eta^*_{AP} \| p_1$
203: send m_1

300: On nonce-forwarding η^*_{AP} to any π^j_C:
301: create π^j_C's response message
302: $m_2 \leftarrow \eta_C \| p_2 \| \tau$ as normal
303: $\mathsf{Fwd} \leftarrow \mathsf{Fwd} \cup \{\eta_C\}$

400: On receiving an m_2 attempt:
401: parse m_2 attempt as $\eta_C \| p_2 \| \tau$
402:
403: **if** $\eta_C \notin \vec{\mathsf{N}}$**:**
404: $\mathsf{distinct} \leftarrow \mathsf{distinct} + 1$
405: $\vec{\mathsf{N}}[\mathsf{distinct}] \leftarrow \eta_C$
406:
407: $d \leftarrow 0$
408: *// if η_C equals the q^*-th distinct nonce*
409: *// and η_C was not nonce-forwarded:*
410: *// embed the $\mathcal{O}^{\mathsf{Vrfy}}$ oracle*
411: **if** $(\eta_C = \vec{\mathsf{N}}[q^*]) \wedge (\eta_C \notin \mathsf{Fwd})$**:**
412: $d \leftarrow \mathcal{O}^{\mathsf{Vrfy}}(\eta_C \| p_2, \tau)$
413: **else**
414: $k_\mu \| k_\alpha \leftarrow \$(AP, C, \eta^*_{AP}, \eta_C)$
415: $d \leftarrow \Sigma.\mathsf{Vrfy}(k_\mu, \eta_C \| p_2, \tau)$
416:
417: **if** $d = 1$**:**
418: stop simulation
419: **else**
420: discard m_2
421: continue simulation

Figure 7: Description of $\mathcal{F}$'s simulation of the guessed session $\pi^i_{U^*}$, depending on whether $U^* = C$ (shown in SimC) or $U^* = AP$ (shown in SimAP). The random function that $\mathcal{F}$ implements for runs between AP and C is denoted by $\$(\cdot)$. In both SimC and SimAP it is assumed that if the parsing of a received message fails, then the message is silently dropped and the simulation continues; for clarity, this is omitted from the code.

Line 100–105: $\mathcal{F}$ begins by drawing a random $q^* \twoheadleftarrow [1, q]$, guessing that the q^*-th distinct nonce that $\pi^i_{U^*}$ receives in an m_2 attempt will lead it to accept maliciously. We emphasize that this does not mean that $\pi^i_{U^*}$ must necessarily maliciously after receiving the q^*-th m_2 attempt (since some of the nonces in these q^* attempts might have been repeated); nor does it mean that $\pi^i_{U^*}$ must necessarily maliciously after receiving the q^*-th distinct nonce for the *first* time (since $\mathcal{A}$ can make many m_2 attempts with this same nonce).

Line 200–203: After guessing q^*, $\mathcal{F}$ creates $\pi^i_{U^*}$'s first protocol message $m_1 = (\eta_{AP}, p_1)$ as normal by generating a random nonce η^*_{AP}.

Line 300–303: If $\mathcal{A}$ forwards $\pi^i_{U^*}$'s nonce η^*_{AP} to some session π^j_C at the client C, then π^j_C's nonce η_C could potentially be used in a nonce-forwarding at some later point. Hence, η_C is stored in Fwd for future reference.

Line 400–421: When $\pi^i_{U^*}$ receives an m_2 message attempt $m_2 = \eta_C \| p_2 \| \tau$, then $\mathcal{F}$ first checks whether η_C equals the q^* distinct nonce that $\pi^i_{U^*}$ received in an m_2 attempt, and that η_C does not constitute a nonce-forwarding between $\pi^i_{U^*}$ and some session at C. If so, then $\mathcal{F}$ uses its MAC verification oracle $\mathcal{O}^{\mathsf{Vrfy}}$ to verify m_2. Otherwise, $\mathcal{F}$ uses the random function $\$(\cdot)$ to derive a MAC key k_μ, and uses the MAC Σ with the key k_μ to verify m_2 itself. In both cases, $\mathcal{F}$ either stops the simulation if the verification succeeds, or continues if it did not (which includes discarding m_2).

In addition to the abort conditions included in the SimC and SimAP procedures, $\mathcal{F}$ also stops its simulation if its guess of $\pi^i_{U^*}$ was wrong in any other way, i.e., if $\pi^i_{U^*}$ was not the first session to accept maliciously.

We claim that $\mathcal{F}$ provides a perfect simulation of Game 3 for adversary $\mathcal{A}$, and makes a valid forgery in its SUF-CMA experiment with probability at least $1/(n_\pi \cdot q)$. Following its description, we break the analysis of $\mathcal{F}$ down into two cases based on whether $\pi^i_{U^*}$ belongs to the client C or the access point AP.

Case $U^* = C$. First assume that $\pi^i_{U^*}$ belongs to the client C, meaning that $\mathcal{F}$ uses the procedure SimC to simulate it. Notice that in this case it's unnecessary for $\mathcal{F}$ to derive a PTK for $\pi^i_{U^*}$ since it forwards all MAC computations to its two oracles $\mathcal{O}^{\mathsf{MAC}}$ and $\mathcal{O}^{\mathsf{Vrfy}}$, and because $\mathcal{F}$'s simulation always stops before $\pi^i_{U^*}$ accepts. Nevertheless, let PTK^* denote the hypothetical key that $\mathcal{F}$ could have derived for $\pi^i_{U^*}$ had it actually done so after $\pi^i_{U^*}$ received message m_1. Specifically, PTK^* would have been derived using the random function $\$(\cdot)$ on the nonces η^*_C and η^*_{AP}.

By Game 1 it is guaranteed that the nonce η^*_C created by $\pi^i_{U^*}$ is unique. Thus, if $\mathcal{A}$ is not nonce-forwarding, then PTK^* would be independent from all other PTKs derived in the simulation. Consequently, only embedding the MAC oracles into $\pi^i_{U^*}$'s computations—in particular, using the oracle $\mathcal{O}^{\mathsf{MAC}}$

to produce $\pi^i_{U^*}$'s first protocol message m_2, and the oracle $\mathcal{O}^{\mathsf{Vrfy}}$ to verify m_3 attempts—yields a correct simulation. Moreover, if the verification of an m_3 message succeeds, then this constitutes a valid forgery in $\mathcal{F}$'s SUF-CMA game since $\mathcal{F}$ never queried its $\mathcal{O}^{\mathsf{MAC}}$ oracle in order to create this m_3 (since the constants p_2 and p_3 in these messages must be different).

On the other hand, if $\mathcal{A}$ *is* nonce-forwarding between $\pi^i_{U^*}$ and some session at the access point AP, say π^j_{AP}, then π^j_{AP} would derive the same PTK* as $\pi^i_{U^*}$. In order to keep the simulation consistent in this case, it is necessary to also use the verification oracle $\mathcal{O}^{\mathsf{Vrfy}}$ at session π^j_{AP} when verifying the m_2 attempts sent to it. If the verification fails, meaning that the simulation continues, then this does not preclude $\mathcal{F}$ from later making a valid forgery to its SUF-CMA game (recall that the SUF-CMA experiment allows multiple verification queries; see Appendix A). On the other hand, if the verification succeeds, meaning that the simulation stops, then π^j_A and $\pi^i_{U^*}$ would be partners by the definition of partner function f. This contradicts the fact that $\pi^i_{U^*}$ was supposed to accept maliciously, so the guess of $\pi^i_{U^*}$ must have been wrong.[14]

Case $U^* = AP$. Now suppose $\pi^i_{U^*}$ belongs to the access point AP, meaning that $\mathcal{F}$ uses the procedure SimAP to simulate it. $\mathcal{F}$'s simulation of message m_1 is of course perfect since it follows the specification of Game 3 precisely. We claim that $\mathcal{F}$ also simulates the verification of any m_2 message attempt perfectly as well.

First, note that $\mathcal{F}$'s verification simulation is *consistent*. That is, for all m_2 attempts that contain the same nonce η_C, $\mathcal{F}$ uses the same MAC key to verify it every time. Of course, in those instances when $\mathcal{F}$ uses its oracle $\mathcal{O}^{\mathsf{Vrfy}}$ to verify m_2 (i.e., line 412 in the SimAP code), then $\mathcal{F}$ does not actually known the corresponding MAC key being used. Consistency is required since the pseudorandom function used to derive the PTK is deterministic, so m_2 attempts containing the same nonce should be verified using the same MAC key every time.

Second, because $\pi^i_{U^*}$'s own nonce η^*_{AP} is unique by Game 1, and because its PTK's are derived using a random function $\$(\cdot)$, it follows that for each *distinct* client nonce η_C that $\pi^i_{U^*}$ receives in m_2 attempts, there corresponds an independent uniformly distributed PTK. Consequently, embedding the verification oracle $\mathcal{O}^{\mathsf{Vrfy}}$ into the computations where η_C is equal to the q^*-th distinct nonce η^*_C and has not been nonce-forwarded, leads to verification answers which are identically distributed to those that $\mathcal{F}$ would get by deriving the PTK itself and using Σ "locally".

Thus, $\mathcal{F}$ perfectly simulates the verification of incoming m_2 messages at session $\pi^i_{U^*}$. It remains to argue that $\mathcal{F}$ makes a valid forgery in its SUF-CMA experiment whenever $\pi^i_{U^*}$ is the first session to accept maliciously and the guess

[14] Of course, if the m_2 attempt that $\mathcal{A}$ sent towards π^j_A was not *identical* to the m_2 message that $\pi^i_{U^*}$ output previously, then this would actually be a valid forgery in $\mathcal{F}$'s SUF-CMA game since the m_2 attempt was not created by a previous call to $\mathcal{O}^{\mathsf{MAC}}$. However, this extra winning chance for $\mathcal{F}$ is not needed in order to bound $\mathcal{A}$'s advantage in Game 3.

of q^* was correct.

Note that there is one instance in which $\mathcal{F}$ does *not* embed its MAC verification oracle into $\pi^i_{U^*}$ at all: if $\mathcal{A}$ was nonce-forwarding between $\pi^i_{U^*}$ and some client session π^j_C when delivering the q^*-th distinct nonce to $\pi^i_{U^*}$. Of course, in this situation $\mathcal{F}$ will certainly not make a valid forgery in its SUF-CMA experiment. However, in this case it would also be impossible for $\pi^i_{U^*}$ to accept maliciously because if $\pi^i_{U^*}$ accepted after receiving an m_2 message containing the nonce that π^j_C sent out, then π^j_C and π^j_C would be partners by the definition of partner function f. In particular, if $\pi^i_{U^*}$ is to accept maliciously, then it cannot be on an m_2 attempt for which $\mathcal{A}$ was nonce-forwarding. Consequently, if $\mathcal{F}$ observes that $\mathcal{A}$ is nonce-forwarding on the q^*-th distinct nonce being delivered to $\pi^i_{U^*}$, then it knows that its guess of either $\pi^i_{U^*}$ or q^* must have been wrong.

Conversely, if the guess of both $\pi^i_{U^*}$ *and* q^* was correct, then $\mathcal{F}$ successfully forges in its SUF-CMA game. Thus, since $\mathcal{F}$'s guess of $\pi^i_{U^*}$ and q^* was done independently of $\mathcal{A}$, and since its simulation of $\pi^i_{U^*}$ is perfect, the probability that $\mathcal{F}$ wins in its SUF-CMA experiment is at least $1/(n_\pi \cdot q)$ times $\mathcal{A}$'s winning probability in Game 3.

Up to a factor of q^{-1}, we see that $\mathcal{F}$ successfully forges whenever $\pi^i_{U^*}$ accepts maliciously in Game 3, regardless of whether $U^* = C$ or $U^* = AP$. This proves the lemma. □

Combining all the bounds from Game 0 to Game 3, together with Lemma 13, yields Theorem 5. □

5.3 Security of IEEE 802.11 with upper-layer authentication

In enterprise and university networks it is inconvenient for every user to share a common PMK when accessing the WLAN (as well as being less secure). User authentication in these environments are instead handled by a central authentication server, typically accessed via EAP. While the IEEE 802.11 standard technically allows for different types of upper-level authentication mechanisms being used, the de-facto standard is EAP. Since we have already proved that certain variants of EAP satisfies the 3P-AKE^w notion (Theorem 3), and that the 4WHS is a secure 2P-AKE protocol providing explicit entity authentication in the AKE$^{\mathsf{static}}$ security model (Theorem 4 and 5); the security of IEEE 802.11 with upper-level authentication now follows directly by applying our second composition theorem (Theorem 2) by setting $\Pi_3 = \text{EAP}$ and $\Pi_4 = \text{4WHS}$.

Theorem 6 (3P-AKE security of IEEE 802.11 w/upper-layer authentication)**.** *If the* PMK *for the 4WHS is derived using a variant of EAP that is 3P-AKE^w-secure, then the IEEE 802.11 protocol with upper-layer authentication is 3P-AKE-secure.*

Acknowledgments

We would like to thank Colin Boyd, Britta Hale and Cas Cremers for helpful comments and discussions. Chris Brzuska is grateful to NXP for supporting his chair for IT Security Analysis.

A Additional definitions

A.1 Pseudorandom functions

A *pseudorandom function (PRF)* is a family of polynomial-time functions $F\colon \{0,1\}^\lambda \times \{0,1\}^\ell \to \{0,1\}^L$, having key-length key-length λ, input length ℓ and output length L. Let $\mathsf{Func}(\ell, L)$ denote the family of all functions from $\{0,1\}^\ell$ to $\{0,1\}^L$. The security of a PRF is defined by the experiments shown in Fig. 8.

$\mathbf{Exp}^{\mathsf{PRF\text{-}0}}_{\mathsf{PRF},\mathcal{A}}(\lambda)$:	$\mathbf{Exp}^{\mathsf{PRF\text{-}1}}_{\mathsf{PRF},\mathcal{A}}(\lambda)$:
1: $K \twoheadleftarrow \{0,1\}^\lambda$	1: $f \twoheadleftarrow \mathsf{Func}(\ell, L)$
2: $b \leftarrow \mathcal{A}^{\mathsf{PRF}(K,\cdot)}(1^\lambda)$	2: $b \leftarrow \mathcal{A}^{f(\cdot)}(1^\lambda)$
3: **return** b	3: **return** b

Figure 8: Experiments defining PRF security.

Definition 10 (PRF)**.** Let PRF be a PRF. The *PRF-advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{A}}(\lambda) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{PRF\text{-}0}}_{\mathsf{PRF},\mathcal{A}}(\lambda) \Rightarrow 1] - \Pr[\mathbf{Exp}^{\mathsf{PRF\text{-}1}}_{\mathsf{PRF},\mathcal{A}}(\lambda) \Rightarrow 1] \qquad (29)$$

A PRF is *(PRF-)secure* if $\mathbf{Adv}^{\mathsf{prf}}_{\mathsf{PRF},\mathcal{A}}(\lambda)$ is negligible in security parameter λ for all PPT adversaries $\mathcal{A}$.

A.2 Message Authentication Codes

A *message authentication code (MAC)* is a pair of polynomial-time algorithms $\Sigma = (\mathsf{MAC}, \mathsf{Vrfy})$, where

- $\mathsf{MAC}\colon \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^*$ is a deterministic *tag-generation* algorithm that takes in a key $K \in \{0,1\}^\lambda$, a *message* $m \in \{0,1\}^*$ and returns a *tag* $\tau \in \{0,1\}^*$.
- $\mathsf{Vrfy}\colon \{0,1\}^\lambda \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ is a deterministic *verification-algorithm* that takes in a key $K \in \{0,1\}^\lambda$, a message $m \in \{0,1\}^*$ and a candidate tag $\tau \in \{0,1\}^*$; and produces a *decision* $d \in \{0,1\}$. Algorithm $\mathsf{Vrfy}(K, \cdot, \cdot)$ works as follows on inputs m and τ: if $\tau = \mathsf{MAC}(K, m)$ then return 1 (ACCEPT) else return 0 (REJECT).

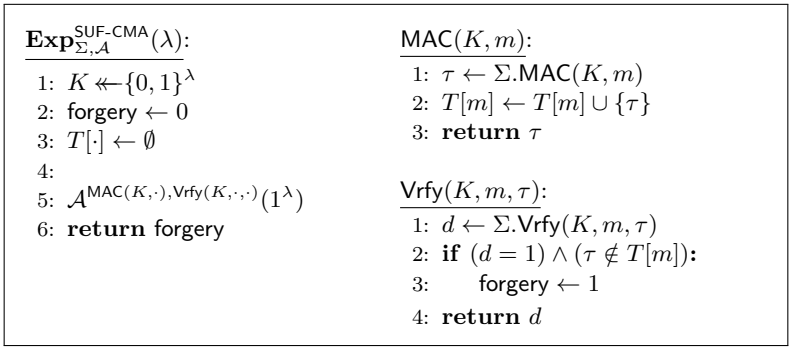

Figure 9: Experiment defining SUF-CMA security for a MAC $\Sigma = (\mathsf{MAC}, \mathsf{Vrfy})$.

The security of a MAC is defined by the experiment shown in Fig. 9.

Definition 11 (SUF-CMA security)**.** Let $\Sigma = (\mathsf{MAC}, \mathsf{Vrfy})$ be a MAC. The *SUF-CMA-advantage* of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda) \stackrel{\text{def}}{=} \Pr[\mathbf{Exp}^{\mathsf{SUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda) \Rightarrow 1]. \tag{30}$$

We say that Σ is *strongly-unforgeable against chosen-message attacks (SUF-CMA)*, or simply *SUF-CMA-secure*, if $\mathbf{Adv}^{\mathsf{SUF\text{-}CMA}}_{\Sigma,\mathcal{A}}(\lambda)$ is negligible in security parameter λ for any PPT adversary $\mathcal{A}$.

B Proof of Lemma 7

Lemma 7.

$$\Pr[M] \leq (n_\pi + 1)^2 \cdot |\mathcal{I} \cup \mathcal{R}|^2 \cdot \left(2 \cdot \mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{B}'_1,f_3}(\lambda) + \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}\text{-}\mathsf{EA}}_{\Pi_4,\mathcal{B}'_2,f_4}(\lambda)\right)$$

Proof. In the following games, let M_i denote the event that a session accepts maliciously in sub-protocol Π_2 in Game i.

Game 0: This is original 3P-AKE security experiment, hence

$$\Pr[M_0] = \Pr[M]. \tag{31}$$

Game 1: This game implements a selective security game similar to Game 2 in the proof of Theorem 1. However, this time the adversary is required to commit to the session that will accept maliciously first.

Specifically, at the beginning of the game the adversary must first choose a pair (U, i), with $i \in [1, n_\pi]$. The game then proceeds as in Game 0, except that if some session accepts maliciously before π^i_U, the challenger aborts the game and outputs 0 (i.e., $\mathcal{A}$ loses). In particular, this includes the possibility

that $\mathcal{A}$ makes a query that renders π_U^i unfresh (which would preclude π_U^i from accepting maliciously).

Lemma 14. $\Pr[M_0] \leq n_\pi \cdot |\mathcal{I} \cup \mathcal{R}| \cdot \Pr[M_1]$.

Proof. The proof is essentially the same as for Game 2 in Theorem 1 (Lemma 2), only that this time the selective security adversary guesses one session rather than two. □

In the remaining games, let π_U^i denote the session that the adversary commits to in Game 1. Note that π_U^i is not necessarily the same as the test-session eventually chosen by the adversary.

Game 2: This game extends the selective security requirements of Game 1 by demanding that the adversary also commits to the partner of π_U^i in sub-protocol Π_3 (if any).

Specifically, at the beginning of the game the adversary must pick a pair (U, i) as in Game 1, and also pick a pair (V, j), with $j \in [0, n_\pi]$. The game then proceeds as in Game 1, but it additionally aborts if π_U^i gets a different f_3-partner than π_V^j in sub-protocol Π_3 (including the case that π_U^i *gets* an f_3-partner when $j = 0$).

Remark 12. Note that there is no contradiction between π_U^i accepting maliciously in protocol Π_5 according to partner function f_5, while simultaneously having an f_3-partner in sub-protocol Π_3.

Lemma 15. $\Pr[M_1] \leq (n_\pi + 1) \cdot \max\{|\mathcal{I}|, |\mathcal{R}|\} \cdot \Pr[M_2]$,

Proof. Again, from an adversary $\mathcal{A}$ that plays against the *single* selective security requirement of Game 1, we can create an adversary $\mathcal{A}'$ against the *two* selective security requirements of Game 2: after $\mathcal{A}$ outputs its commitment to a pair (U, i), then $\mathcal{A}'$ guesses another pair (V, j) (conditioned on the role of U), and outputs (U, i) and (V, j) as its own commitments to Game 2. □

In the remaining games, let π_V^j denote the (possibly empty) f_3-partner of π_U^i that the adversary commits to in Game 2 (in addition to π_U^i).

Game 3: This game proceeds as the previous one, but when the first session out of π_U^i and π_V^j accepts in sub-protocol Π_3, then the challenger replaces its intermediate key K_{Π_3} with a random key $\widetilde{K}$. When the other session accepts in Π_3, the challenger replaces its intermediate key with the same random key $\widetilde{K}$.

Lemma 16. $\Pr[M_2] \leq \Pr[M_3] + 2 \cdot \mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3, \mathcal{B}'_1, f_3}(\lambda)$.

Proof (sketch). Reduction $\mathcal{B}'_1$ begins by choosing a random bit b_{sim}. It then runs $\mathcal{A}$ and implements all the abort conditions introduced so far. All of $\mathcal{A}$'s Send queries that pertain to the 3P-AKE sub-protocol Π_3, $\mathcal{B}'_1$ forwards to its own 3P-AKE experiment.

When the first session out of π_U^i and π_V^j accepts in sub-protocol Π_3, $\mathcal{B}_1'$ makes a Test query to its own 3P-AKE experiment in order to obtain its session key k^* (which is either its actual key or a random key). $\mathcal{B}_1'$ uses k^* as that session's intermediate key K_{Π_3} in protocol Π_5. When the second session out of π_U^i and π_V^j accepts in sub-protocol Π_3, then $\mathcal{B}_1'$ again uses the key k^* as its intermediate key K_{Π_3}. For all sessions different from π_U^i and π_V^j $\mathcal{B}_1'$ obtains their intermediate keys by making a corresponding Reveal query to its own 3P-AKE experiment.

All of $\mathcal{A}$'s Send queries that pertain to sub-protocol Π_4, $\mathcal{B}_1'$ answers itself using the intermediate keys it obtained from sub-protocol Π_3 as the "long-term keys" for sub-protocol Π_4. In particular, $\mathcal{B}_1'$ derives the session keys of every session in sub-protocol Π_4 (and hence in protocol Π_5) itself. Using the session keys so-obtained, together with the bit b_{sim} it drew in the beginning of the simulation, $\mathcal{B}_1'$ can answer $\mathcal{A}$'s Test-query.

Finally, when π_U^i accepts in protocol Π_5, then $\mathcal{B}_1$ stops it simulation and outputs a bit b_{out} to its own 3P-AKE experiment defined as follows. If π_U^i accepted maliciously in protocol Π_5, then $b_{out} = 0$, otherwise, b_{out} is defined to be a random bit.

Recall that $\mathcal{B}_1'$ picks its test-session based on which of π_U^i and π_V^j accepted first in sub-protocol Π_3. We argue that if π_U^i accepts maliciously in $\mathcal{B}_1'$'s simulation, then both π_U^i and π_V^j are valid test-targets in $\mathcal{B}_1'$'s own 3P-AKE experiment (i.e., fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^w}$). We consider three cases:

- *π_U^i chosen as test-session and $j \neq 0$:* In this case $\mathcal{B}_1'$ makes no Reveal query towards π_U^i or its f_3-partner π_V^j in its own 3P-AKE experiment because it obtains their intermediate keys from sub-protocol Π_3 through the Test query (to π_U^i). Moreover, since π_U^i has an f_3-partner, it remains AKE^w-fresh even if its peers are corrupted.

- *π_U^i chosen as test-session and $j = 0$:* Again, $\mathcal{B}_1'$ makes no Reveal query towards π_U^i in its own 3P-AKE experiment, and since π_U^i doesn't have an f_3-partner, $\mathcal{B}_1'$ of course makes no Reveal query to that either. We claim that $\mathcal{B}_1'$ also never issued a $\mathsf{LongTermKeyReveal}$ query to π_U^i's peers. To see this, note that if π_U^i is to accept maliciously in protocol Π_5, then it must be fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}}$. In particular, this means that $\mathcal{A}$ cannot issue any $\mathsf{LongTermKeyReveal}$ queries to π_U^i's peers *before* π_U^i accepted.[15] But $\mathcal{B}_1'$ stops its simulation immediately once π_U^i accepts, so no $\mathsf{LongTermKeyReveal}$ query will actually be forwarded towards π_U^i's peers in $\mathcal{B}_1'$'s 3P-AKE experiment in this case.

- *π_V^j chosen as test-session:* Note that by symmetry of the f_3 partner function, π_V^j has π_U^i as its f_3-partner. Thus, $\mathcal{B}_1'$ makes no Reveal query towards

[15] Recall that predicate $\mathsf{Fresh}_{\mathsf{AKE}}$ forbids any $\mathsf{LongTermKeyReveal}$ query to a session's peers if (1) it does not have a partner, and (2) it has not accepted yet. This corresponds exactly to the setting we are in when a session accepts maliciously.

π_V^j or its f_3-partner (π_U^i), since this is answered with the Test query instead' Moreover, since π_V^j has an f_3-partner, it remains AKE^w-fresh even if its peers are corrupted.

Taken together, the above cases show that no-matter which one of π_U^i and π_V^i was selected as the test-session by $\mathcal{B}_1'$, it will be fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^w}$ in $\mathcal{B}_1'$'s 3P-AKE experiment if π_U^i accepted maliciously.

Finally, we analyze $\mathcal{B}_1'$'s advantage. Let b denote the challenge-bit used in $\mathcal{B}_1'$'s own 3P-AKE experiment and let $\mathsf{MA}^{\mathcal{B}_1'}$ denote the event that π_U^i accepted maliciously in $\mathcal{B}_1'$'s simulation.

If $b = 0$, then $\mathcal{B}_1'$'s Test query is answered with a real key. In this case $\mathcal{B}_1'$ simulates Game 2 perfectly for $\mathcal{A}$ up until the point when π_U^i accepts (in protocol Π_5), thus:

$$\Pr[\mathsf{MA}^{\mathcal{B}_1'} \mid b = 0] = \Pr[M_2]. \tag{32}$$

On the other hand, if $b = 1$, meaning that $\mathcal{B}_1'$'s Test query is answered with a random key, then $\mathcal{B}_1'$ simulates Game 3 perfectly for $\mathcal{A}$, thus:

$$\Pr[\mathsf{MA}^{\mathcal{B}_1'} \mid b = 1] = \Pr[M_3]. \tag{33}$$

Furthermore, the probability that $\mathcal{B}_1'$ outputs $b_{out} = 0$ when the event $\mathsf{MA}^{\mathcal{B}_1'}$ occurs is 1, and and when it don't occurs the probability is 1/2 (independently of the value of b). Thus, $\mathcal{B}_1'$'s advantage is:

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{3P\text{-}AKE}^w}_{\Pi_3,\mathcal{B}_1',f_3}(\lambda) &= \Pr[\mathbf{Exp}^{\mathsf{AKE}^w}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda) \Rightarrow 1 \mid b = 0] \\
&\quad - \Pr[\mathbf{Exp}^{\mathsf{AKE}^w}_{\Pi,\mathcal{Q},\mathcal{A}}(\lambda) \Rightarrow 0 \mid b = 1] && (34) \\
&= 1 \cdot \Pr[\mathsf{MA}^{\mathcal{B}_1'} \mid b = 0] + \frac{1}{2} \cdot \Big(1 - \Pr[\mathsf{MA}^{\mathcal{B}_1'} \mid b = 0]\Big) \\
&\quad - 1 \cdot \Pr[\mathsf{MA}^{\mathcal{B}_1'} \mid b = 1] - \frac{1}{2} \cdot \Big(1 - \Pr[\neg\mathsf{MA}^{\mathcal{B}_1'} \mid b = 1]\Big) && (35) \\
&\overset{(32)+(33)}{=} \frac{1}{2} \cdot \Pr[M_2] - \frac{1}{2} \cdot \Pr[M_3], && (36)
\end{aligned}$$

which proves the lemma. □

Finally, we show that if π_U^i accepts maliciously in Game 3, then it must also have accepted maliciously in sub-protocol Π_4. Here we use that the AKE$^{\mathsf{static}}$ experiment allows key-registration of long-term PSKs.

Lemma 17. $\Pr[M_3] \leq \mathbf{Adv}^{\mathsf{2P\text{-}AKE}^{\mathsf{static}}\text{-}\mathsf{EA}}_{\Pi_4,\mathcal{B}_2',f_4}(\lambda)$.

Proof (sketch). Reduction $\mathcal{B}_2'$ begins by creating all the long-term keys for sub-protocol Π_3, then runs $\mathcal{A}$. For all of $\mathcal{A}$'s Send queries that pertain to sub-protocol Π_3, $\mathcal{B}_2'$ answers itself using the keys it created. Once an initiator or responder session π accepts in sub-protocol Π_3, then $\mathcal{B}_2'$ forwards all of $\mathcal{A}$'s remaining Send queries towards π to its own 2P-AKE$^{\mathsf{static}}$ experiment.

Specifically, once a session π accepts in in sub-protocol Π_3 then $\mathcal{B}'_2$ creates a corresponding "proxy" session in its own 2P-AKE$^{\mathsf{static}}$ experiment with a $\mathsf{NewSession}$ query. If π is one of the two selective security targets π^i_U and π^j_V, then $\mathcal{B}'_2$ uses a $\mathsf{NewSession}$ query *without* key registration. On the other hand, if π is different π^i_U and π^j_V, then $\mathcal{B}'_2$ uses a $\mathsf{NewSession}$ query *with* key registration, where the key that $\mathcal{B}'_2$ registers is the session key it derived for π in sub-protocol Π_3.

To answer $\mathcal{A}$'s $\mathsf{LongTermKeyReveal}$ queries, $\mathcal{B}'_2$ uses the long-term keys it created for sub-protocol Π_3. $\mathcal{A}$'s Reveal queries, as well as Test query, $\mathcal{B}'_2$ forwards to its own 2P-AKE$^{\mathsf{static}}$ experiment.

Finally, when π^i_U accepts in protocol Π_5, then $\mathcal{B}'_2$ stops it simulation and outputs a random bit to its 2P-AKE$^{\mathsf{static}}$ experiment.[16]

Note that $\mathcal{B}'_2$ provides a perfect simulation of Game 3. Specifically, it provides a correct simulation of sub-protocol Π_3 since it created all the keys itself. Moreover, the simulation of sub-protocol Π_4 is also perfect, since the long-term keys used by the proxy sessions in $\mathcal{B}'_2$'s own 2P-AKE$^{\mathsf{static}}$ experiment are identically distributed to the "intermediate" keys in protocol Π_5. In particular, for sessions different from π^i_U and π^j_V the use of key registration ensures that the PSK used by the corresponding proxy sessions is exactly the same as intermediate keys derived in protocol Π_5. On the other hand, when $\mathcal{B}'_2$ creates the proxy sessions corresponding to π^i_U and π^j_V, it does so *without* key registration. By definition, this means that in $\mathcal{B}'_2$'s 2P-AKE$^{\mathsf{static}}$ experiment these proxy sessions will use an an independent, uniformly distributed PSK. However, by the change in Game 3, π^i_U and π^j_V's intermediate keys in protocol Π_5 are replaced with an independent, randomly distributed key $\widetilde{K}$, hence the distribution is the same.

It remains to argue that if π^i_U accepts maliciously in Game 3, then it must also have accepted maliciously in sub-protocol Π_4—which by extension implies that the corresponding proxy session in $\mathcal{B}'_2$'s 2P-AKE$^{\mathsf{static}}$ experiment accepted maliciously too.

First we claim that session π^j_V cannot be π^i_U's partner in sub-protocol Π_4. To see this, note that if $j = 0$, then $\mathcal{B}'_2$ never creates a corresponding proxy session in its 2P-AKE$^{\mathsf{static}}$ experiment, hence π^i_U cannot possibly have a partner in sub-protocol Π_4. On the other hand, if $j \neq 0$, then by the definition of π^j_V (ref. Game 2) it must be π^i_U's f_3-partner in sub-protocol Π_3. But if π^j_V was *also* the f_4-partner of π^i_U in sub-protocol Π_4, then it would necessarily be π^i_U's f_5-partner as well—contradicting the fact that π^i_U was supposed to accept maliciously.

Second, we claim that π^i_U is fresh according to predicate $\mathsf{Fresh}_{\mathsf{AKE}^{\mathsf{static}}}$. This is true because $\mathcal{B}'_2$ makes no $\mathsf{LongTermKeyReveal}$ query at all in its 2P-AKE$^{\mathsf{static}}$ experiment, and also makes no Reveal query to the proxy session of π^i_U.[17]

[16] We do not care about the value of $\mathcal{B}'_2$'s output since we are only interested in its ability to get a session to accept maliciously in sub-protocol Π_4.

[17] As argued by the previous claim, π^j_V is not π^i_U's f_4-partner in sub-protocol Π_4, so a Reveal query to it does not affect the AKE$^{\mathsf{static}}$-freshness of π^i_U.

Together, these two claims show that the proxy session corresponding to π_U^i accepted maliciously in $\mathcal{B}_2'$'s 2P-AKE$^{\mathsf{static}}$ experiment. □

Combining Lemma 14 to Lemma 17 yields Lemma 7. □

C Partner function parsing rules

Let T_3 be the transcript generated by running $\mathbf{Exp}_{\Pi_3,\mathcal{Q}_3,\mathcal{A}}(\lambda)$. Table 2 explains how to extract from T_3 two other transcripts, T_1 and T_2, corresponding to runs of $\mathbf{Exp}_{\Pi_1,\mathcal{Q}_1,\mathcal{A}'}(\lambda)$ and $\mathbf{Exp}_{\Pi_2,\mathcal{Q}_2,\mathcal{A}''}(\lambda)$, respectively. Essentially, the parsing rules work as follows.

- For each initiator session in protocol Π_3, create a corresponding initiator session in protocl Π_1.
- For each responder session in protocol Π_3, create a corresponding responder session in protocol Π_2.
- For each trusted-third party session in protocol Π_3, create *two* sessions: one *responder* session in protocol Π_1; and one *initiator* session in protocol Π_2. However, the latter is only created "on-demand" after the trusted third-party session accepts in sub-protocol Π_1.

References

[1] IEEE Standard for Local and metropolitan area networks - Port-Based Network Access Control. *IEEE Std 802.1X-2010 (Revision of IEEE Std 802.1X-2004)*, pages C1–205, Feb 2010. `https://doi.org/10.1109/IEEESTD.2010.5409813`. (Cited on page 38.)

[2] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2012*, pages 1–2793, March 2012. `https://dx.doi.org/10.1109/IEEESTD.2012.6178212`. (Cited on pages 3, 5, and 39.)

[3] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005: 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 65–84, Les Diablerets, Switzerland, January 23–26, 2005. Springer, Heidelberg, Germany. (Cited on page 3.)

Table 2: Parsing rules for extracting transcripts T_1 and T_2 from a transcript T_3, assuming that $A \in \mathcal{I}$, $B \in \mathcal{R}$ and $S \in \mathcal{S}$ in protocol Π_3. Parsing is done as follows. For each entry in T_3, look up the row in the column marked "T_3" that matches this query. From this row, use the corresponding queries in the columns marked "T_1" and "T_2" to create the entries on T_1 and T_2 respectively ("$-$" means that no query should be created).

T_3	T_1	T_2
$(\mathsf{NewSession}(A, B, S), \pi_A^i, m)$	$(\mathsf{NewSession}(A, S), \pi_A^i, m)$	$-$
$(\mathsf{NewSession}(B, A, S), \pi_B^j, \perp)$	$-$	$(\mathsf{NewSession}(B, S), \pi_B^j, \perp)$
$(\mathsf{NewSession}(S, A, B), \pi_S^k, \perp)$	$(\mathsf{NewSession}(S, A), \pi_S^k, \perp)$	$-$
$(\mathsf{Send}(\pi_A^i, m), m^*, (\mathsf{running}, \mathsf{running}, \mathsf{running}))$	$(\mathsf{Send}(\pi_A^i, m), m^*, (\mathsf{running}))$	$-$
$(\mathsf{Send}(\pi_A^i, m), m^*, (\mathsf{accepted}, \mathsf{accepted}, \mathsf{accepted}))$	$(\mathsf{Send}(\pi_A^i, m), m^*, (\mathsf{accepted}))$	$-$
$(\mathsf{Send}(\pi_A^i, m), \perp, (\mathsf{rejected}, \mathsf{rejected}, \mathsf{rejected}))$	$(\mathsf{Send}(\pi_A^i, m), \perp, (\mathsf{rejected}))$	$-$
$(\mathsf{Send}(\pi_B^j, m), m^*, (\mathsf{accepted}, \mathsf{running}, \mathsf{running}))$	$-$	$(\mathsf{Send}(\pi_B^j, m), m^*, (\mathsf{running}))$
$(\mathsf{Send}(\pi_B^j, m), m^*, (\mathsf{accepted}, \mathsf{accepted}, \mathsf{running}))$	$-$	$(\mathsf{Send}(\pi_B^j, m), m^*, (\mathsf{accepted}))$
$(\mathsf{Send}(\pi_B^j, m), \perp, (\mathsf{accepted}, \mathsf{rejected}, \mathsf{rejected}))$	$-$	$(\mathsf{Send}(\pi_B^j, m), \perp, (\mathsf{rejected}))$
$(\mathsf{Send}(\pi_B^j, C_{\mathrm{key}}), \perp, (\mathsf{accepted}, \mathsf{accepted}, \mathsf{accepted}))$	$-$	$-$
$(\mathsf{Send}(\pi_B^j, C'_{\mathrm{key}}), \perp, (\mathsf{accepted}, \mathsf{accepted}, \mathsf{rejected}))$	$-$	$-$
$(\mathsf{Send}(\pi_S^k, m), m^*, (\mathsf{running}, \mathsf{running}, \mathsf{running}))$	$(\mathsf{Send}(\pi_S^k, m), m^*, (\mathsf{running}))$	$-$
$(\mathsf{Send}(\pi_S^k, m), \perp, (\mathsf{rejected}, \mathsf{rejected}, \mathsf{rejected}))$	$(\mathsf{Send}(\pi_S^k, m), \perp, (\mathsf{rejected}))$	$-$
$(\mathsf{Send}(\pi_S^k, m), m^*, (\mathsf{accepted}, \mathsf{running}, \mathsf{running}))^\dagger$	$(\mathsf{Send}(\pi_S^k, m), \perp, (\mathsf{accepted}))$	$(\mathsf{NewSession}(S, B), \pi_S^k, m^*)$
$(\mathsf{Send}(\pi_S^k, m), m^*, (\mathsf{accepted}, \mathsf{running}, \mathsf{running}))$	$-$	$(\mathsf{Send}(\pi_S^k, m), m^*, (\mathsf{running}))$
$(\mathsf{Send}(\pi_S^k, m), C_{\mathrm{key}}, (\mathsf{accepted}, \mathsf{accepted}, \mathsf{accepted}))$	$-$	$(\mathsf{Send}(\pi_S^k, m), \perp, (\mathsf{accepted}))$
$(\mathsf{Send}(\pi_S^k, m), \perp, (\mathsf{accepted}, \mathsf{rejected}, \mathsf{rejected}))$	$-$	$(\mathsf{Send}(\pi_S^k, m), \perp, (\mathsf{rejected}))$

†This rule only applies if $\pi_S^k.\vec{\alpha} = (\mathsf{running}, \mathsf{running}, \mathsf{running})$ for all prior Send queries to π_S^k, i.e., if receiving message m caused session π_S^k to accept in sub-protocol Π_1.

[4] Bernard Aboba, Larry J. Blunk, John R. Vollbrecht, James Carlson, and Henrik Levkowetz. Extensible Authentication Protocol. RFC 3748, RFC Editor, June 2004. `https://tools.ietf.org/html/rfc3748`. (Cited on pages 3, 5, and 37.)

[5] Stéphanie Alt, Pierre-Alain Fouque, Gilles Macario-Rat, Cristina Onete, and Benjamin Richard. A cryptographic analysis of UMTS/LTE AKA. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16: 14th International Conference on Applied Cryptography and Network Security*, volume 9696 of *Lecture Notes in Computer Science*, pages 18–35, Guildford, UK, June 19–22, 2016. Springer, Heidelberg, Germany. (Cited on pages 3, 7, and 39.)

[6] Mihir Bellare, Oded Goldreich, and Anton Mityagin. The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309, 2004. `http://eprint.iacr.org/2004/309`. (Cited on page 45.)

[7] Mihir Bellare, Dennis Hofheinz, and Eike Kiltz. Subtleties in the definition of IND-CCA: When and how should challenge decryption be disallowed? *Journal of Cryptology*, 28(1):29–48, January 2015. (Cited on page 17.)

[8] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany. (Cited on page 9.)

[9] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993. (Cited on page 9.)

[10] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: The three party case. In *27th Annual ACM Symposium on Theory of Computing*, pages 57–66, Las Vegas, NV, USA, May 29 – June 1, 1995. ACM Press. (Cited on pages 3, 7, 8, 8, 12, 12, 13, and 19.)

[11] Karthikeyan Bhargavan, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Santiago Zanella Béguelin. Proving the TLS handshake secure (as it is). In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 235–255, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on page 37.)

[12] Chris Brzuska, Cas Cremers, Håkon Jacobsen, and Bogdan Warinschi. Partner mechanisms in key exchange protocols. Unpublished manuscript, 2017. (Cited on page 13.)

[13] Christina Brzuska, Marc Fischlin, Bogdan Warinschi, and Stephen C. Williams. Composability of Bellare-Rogaway key exchange protocols. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 51–62, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. (Cited on pages 9 and 14.)

[14] Christina Brzuska, Håkon Jacobsen, and Douglas Stebila. Safely exporting keys from secure channels: On the security of EAP-TLS and TLS key exporters. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 670–698, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany. (Cited on page 37.)

[15] 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, Santa Barbara, CA, USA, August 18–22, 2002. Springer,

Heidelberg, Germany. http://eprint.iacr.org/2002/120/. (Cited on page 12.)

[16] T. Charles Clancy and Katrin Hoeper. Making the case for EAP channel bindings. In *2009 IEEE Sarnoff Symposium, Princeton, NJ, March 30-31 & April 1*, pages 1–5. IEEEXplore, March 2009. https://doi.org/10.1109/SARNOF.2009.4850319. (Cited on page 38.)

[17] Morris J. Dworkin. SP 800-38B. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, October 2016. https://dx.doi.org/10.6028/NIST.SP.800-38B. (Cited on page 45.)

[18] Victor Fajardo, Jari Arkko, John Loughney, and Glen Zorn. Diameter Base Protocol. RFC 6733, IETF RFC Editor, October 2012. https://tools.ietf.org/html/rfc6733. (Cited on pages 5, 6, and 37.)

[19] Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *2016 IEEE Symposium on Security and Privacy*, pages 452–469, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press. (Cited on pages 8 and 19.)

[20] Wesley George and Charles Rackoff. Rethinking definitions of security for session key agreement. Cryptology ePrint Archive, Report 2013/139, 2013. http://eprint.iacr.org/2013/139. (Cited on page 17.)

[21] Sam Hartman, T. Charles Clancy, and Katrin Hoeper. Channel-Binding Support for Extensible Authentication Protocol (EAP) Methods. RFC 6677, RFC Editor, July 2012. https://tools.ietf.org/html/rfc6677. (Cited on pages 6, 6, 6, and 38.)

[22] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05: 12th Conference on Computer and Communications Security*, pages 2–15, Alexandria, Virginia, USA, November 7–11, 2005. ACM Press. (Cited on page 7.)

[23] Katrin Hoeper and Lidong Chen. Where EAP security claims fail. In Victor Leung and Sastri Kota, editors, *4th International ICST Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness, QSHINE 2007, Vancouver, Canada, August 14-17, 2007*, page 46. ACM, 2007. (Cited on pages 3 and 38.)

[24] 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, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany. (Cited on pages 17, 19, and 37.)

[25] Kazukuni Kobara, SeongHan Shin, and Mario Strefler. Partnership in key exchange protocols. In Wanqing Li, Willy Susilo, Udaya Kiran Tupakula, Reihaneh Safavi-Naini, and Vijay Varadharajan, editors, *ASIACCS 09: 4th ACM Symposium on Information, Computer and Communications Security*, pages 161–170, Sydney, Australia, March 10–12, 2009. ACM Press. (Cited on page 9.)

[26] Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367, 2013. `http://eprint.iacr.org/2013/367`. (Cited on pages 19 and 37.)

[27] Hugo Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. (Cited on page 19.)

[28] 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, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany. (Cited on page 8.)

[29] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Available at `http://www.ietf.org/rfc/rfc2104.txt`. (Cited on page 45.)

[30] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. (Cited on pages 19 and 37.)

[31] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. Cryptology ePrint Archive, Report 2013/339, 2013. `http://eprint.iacr.org/2013/339`. (Cited on page 23.)

[32] Ralf Küsters and Max Tuengerthal. Composition theorems without pre-established session identifiers. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 11: 18th Conference on Computer and Communications Security*, pages 41–50, Chicago, Illinois, USA, October 17–21, 2011. ACM Press. (Cited on page 7.)

[33] Ralf Küsters and Max Tuengerthal. Ideal key derivation and encryption in simulation-based security. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 161–179, San Francisco, CA, USA, February 14–18, 2011. Springer, Heidelberg, Germany. (Cited on page 7.)

[34] Brian LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. Cryptology ePrint Archive, Report 2006/073, 2006. `http://eprint.iacr.org/2006/073`. (Cited on pages 7, 14, and 15.)

[35] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007: 1st International Conference on Provable Security*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16, Wollongong, Australia, November 1–2, 2007. Springer, Heidelberg, Germany. (Cited on page 14.)

[36] Yong Li, Sven Schäge, Zheng Yang, Florian Kohlar, and Jörg Schwenk. On the security of the pre-shared key ciphersuites of TLS. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 669–684, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany. (Cited on page 37.)

[37] Junghyun Nam, Kim-Kwang Raymond Choo, Juryon Paik, and Dongho Won. Two-round password-only authenticated key exchange in the three-party setting. Cryptology ePrint Archive, Report 2014/017, 2014. `http://eprint.iacr.org/2014/017`. (Cited on page 3.)

[38] Yoshihiro Ohba, Mohan Parthasarathy, and Mayumi Yanagiya. Channel Binding Mechanism based on Parameter Binding in Key Derivation. RFC (Informational), IETF RFC Editor, December 2006. `https://tools.ietf.org/html/draft-ohba-eap-channel-binding-02`. (Cited on pages 6, 37, and 38.)

[39] Carl Rigney, Allan Rubens, William Allen Simpson, and Steve Willens. Remote Authentication Dial In User Service (RADIUS). RFC 2865, IETF RFC Editor, June 2000. `https://tools.ietf.org/html/rfc2865`. (Cited on page 5.)

[40] Phillip Rogaway. On the of role of definitions in and beyond cryptography. In *ASIAN*, volume 3321 of *Lecture Notes in Computer Science*, pages 13–32. Springer, 2004. (Cited on pages 9 and 19.)

[41] Jörg Schwenk. Nonce-based kerberos is a secure delegated AKE protocol. Cryptology ePrint Archive, Report 2016/219, 2016. `http://eprint.iacr.org/2016/219`. (Cited on pages 3 and 3.)

[42] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. http://eprint.iacr.org/2004/332. (Cited on pages 23, 27, and 32.)

[43] Victor Shoup and Aviel D. Rubin. Session key distribution using smart cards. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 321–331, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany. (Cited on page 13.)

[44] Dan Simon, Bernhard Aboba, and Ryan Hurst. The EAP-TLS Authentication Protocol. RFC 5216 (Proposed Standard), March 2008. Available at http://tools.ietf.org/html/rfc5216. (Cited on page 4.)

[45] Stefan Winter, Mike McCauley, Stig Venaas, and Klaas Wierenga. Transport Layer Security (TLS) encryption for RADIUS. RFC 6614 (Experimental), IETF RFC Editor, May 2012. https://tools.ietf.org/html/rfc6614. (Cited on pages 6 and 37.)