J. Cryptol. (2010) 23: 187–223
DOI: 10.1007/s00145-009-9052-3

Journal of
CRYPTOLOGY

The TLS Handshake Protocol: A Modular Analysis[*]

P. Morrissey, N.P. Smart, and B. Warinschi
Department Computer Science, University of Bristol, Merchant Venturers Building, Woodland Road, Bristol BS8 1UB, UK
paulm@cs.bris.ac.uk; nigel@cs.bris.ac.uk; bogdan@cs.bris.ac.uk

Communicated by Okamoto

Received 12 December 2008 and revised 30 September 2009
Online publication 29 October 2009

Abstract. We study the security of the widely deployed Secure Session Layer/ Transport Layer Security (TLS) key agreement protocol. Our analysis identifies, justifies, and exploits the modularity present in the design of the protocol: the *application keys* offered to higher-level applications are obtained from a *master key*, which in turn is derived through interaction from a *pre-master key*.

We define models (following well-established paradigms) that clarify the security level enjoyed by each of these types of keys. We capture the realistic setting where only one of the two parties involved in the execution of the protocol (namely the server) has a certified public key, and where the same master key is used to generate multiple application keys.

The main contribution of the paper is a modular and generic proof of security for a slightly modified version of TLS. Our proofs shows that the protocol is secure even if the pre-master and the master keys only satisfy only weak security requirements. Our proofs make crucial use of modelling the key derivation function of TLS as a random oracle.

Key words. Provable security, TLS, SSL.

1. Introduction

The SSL key agreement protocol, developed by Netscape, was made publicly available in 1994 [28] and after various improvements [23] has formed the basis for the TLS protocol [20,21] which is nowadays ubiquitously present in secure communications over the Internet. Despite its practical importance, this protocol had never been analysed in detail using the rigorous methods of modern cryptography. In this paper we offer one such analysis. Before describing our results and discussing their implications we recall the structure of the TLS protocol (Fig. 1). In the presentation and in our analysis we depart slightly from the actual protocol in ways that we specify precisely. The protocol

[*] This paper was solicited by the Editors-in-Chief as one of the best papers from ASIACRYPT 2008, based on the recommendation of the program committee.

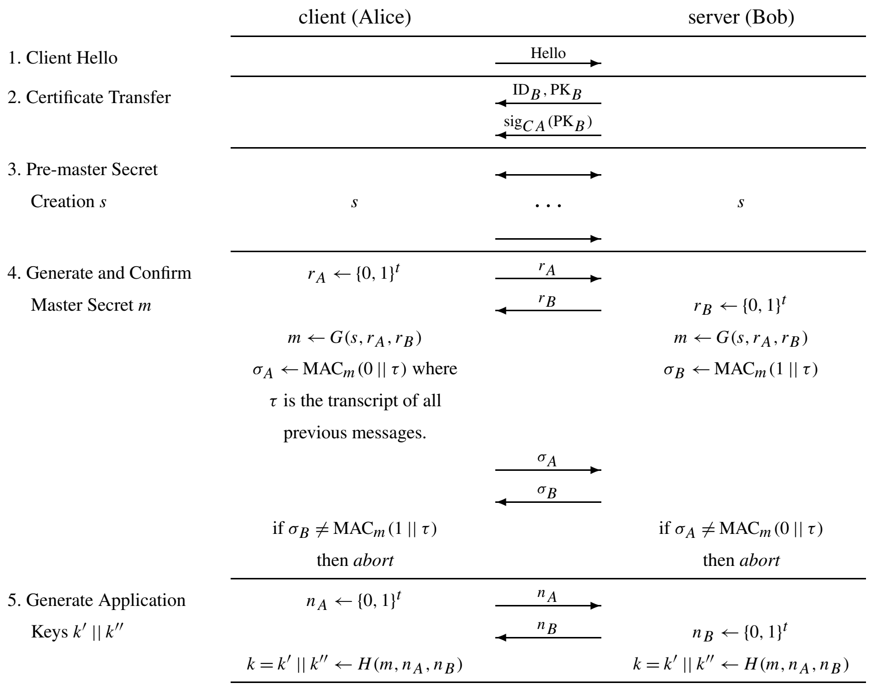
client (Alice)
server (Bob)
1. Client Hello
Hello
2. Certificate Transfer
$\mathrm{ID}_B, \mathrm{PK}_B$
$\mathrm{sig}_{CA}(\mathrm{PK}_B)$
3. Pre-master Secret Creation s
s
...
s
4. Generate and Confirm Master Secret m
$r_A \leftarrow \{0,1\}^t$
r_A
r_B
$r_B \leftarrow \{0,1\}^t$
$m \leftarrow G(s, r_A, r_B)$
$\sigma_A \leftarrow \mathrm{MAC}_m(0 \,||\, \tau)$ where τ is the transcript of all previous messages.
$m \leftarrow G(s, r_A, r_B)$
$\sigma_B \leftarrow \mathrm{MAC}_m(1 \,||\, \tau)$
σ_A
σ_B
if $\sigma_B \neq \mathrm{MAC}_m(1 \,||\, \tau)$ then *abort*
if $\sigma_A \neq \mathrm{MAC}_m(0 \,||\, \tau)$ then *abort*
5. Generate Application Keys $k' \,||\, k''$
$n_A \leftarrow \{0,1\}^t$
n_A
n_B
$n_B \leftarrow \{0,1\}^t$
$k = k' \,||\, k'' \leftarrow H(m, n_A, n_B)$
$k = k' \,||\, k'' \leftarrow H(m, n_A, n_B)$

Fig. 1. A TLS-like protocol: in the actual TLS specification (1) nonces r_A, r_B are exchanged in the first phase of the protocol, (2) the master key m is derived via $m \leftarrow F_s(r_A, r_B)$ where F is a pseudorandom function, (3) the MACs σ_A, σ_B are sent encrypted under k' and k'' respectively, and (4) a FINISHED message is sent authenticated and encrypted under the derived keys.

proceeds in six phases. Through phases (1) and (2) parties confirm their willingness to engage in the protocol, exchange, and verify the validity of their identities and public keys. In our analysis we assume that the server possess a long-term public/private key pair $(\mathrm{PK}_B, \mathrm{SK}_B)$ and a corresponding certificate $\mathrm{sig}_{\mathrm{CA}}(\mathrm{PK}_B)$ issued by some certification authority CA. We do not consider clients with public keys since this is the case that occurs most often in practice. Our analysis can be easily extended to this scenario. The next four phases of the protocol form the focus of this paper and are as follows.

(3) A *pre-master secret* $s \in \mathcal{S}_{\mathrm{PMS}}$ is obtained using one of a number of protocols that include RSA based key transport and signed Diffie–Hellman key exchange (which we describe and analyse later in the paper).
(4) The pre-master secret key s is used to derive a *master secret* $m \in \mathcal{S}_{\mathrm{MS}}$, with $m = G(s, r_A, r_B)$. Here r_A, r_B are random nonces that the two parties exchange, and G is a key derivation function. The master secret key is confirmed by using it to compute two MACs of the transcript of the conversation which are then exchanged. This is a departure from the actual TLS where nonces r_A and r_B are exchanged in the first phase of the protocol and the MACs are sent encrypted with application key derived from m in later steps of the protocol.
(5) In the next phase the master key m is used to obtain one or more application keys: for each application key, the parties exchange random nonces n_A and n_B

and compute the shared application key via $k = k' \,||\, k'' \leftarrow H(m, n_A, n_B)$. Here, H is a key derivation function. Notice, that each application key is actually two keys: one for securing communication from the client to the server, and one from the server to the client. This is important to prevent reflection attacks.

The resulting keys can then be used by further applications. The proper use of keys in this last stage had been the object of previous studies [4,30] and is not part of our analysis.

An interesting aspect of TLS is that the protocols used to obtain the pre-master secret in Step (3) are very simplistic and on their own insecure in the terms of modern cryptography. It is the combination of Step (3) with the steps in phases (4) and (5) which leads (as we show in this paper) to a key agreement protocol secure in the standard sense. Broadly speaking, our goal is to derive sufficient security conditions on the pre-master key agreement protocol which would ensure that the above combination indeed yields a secure key agreement protocol in a standard cryptographic sense.

We caution that in our analysis we disregard Steps (1) and (2), and therefore assume an existing PKI which authenticates all public keys in use in the system. In particular we do not take into account any so-called PKI attacks [35].

Models. Much of the previous work on key agreement protocols in the provable security community has focused on defining security models and then creating protocols which meet the security goals of the models. In some sense, we are taking the opposite approach: we focus on a particular existing protocol, namely TLS, and develop security models that capture the security levels that the various keys derived in one execution of the protocol enjoy. The models that we develop are in the spirit of existing ones and offer a reasonable level of security. It should be noted however that the path we take is mainly motivated by the lack of models that capture *precisely* the security of these keys. A similar route was followed by Canetti and Krawczyk, who modify an existing security model to accommodate a setting where peer identities are not known a priorily [17].

A key aspect of our approach is that unlike in prior work on key agreement protocols, we do not regard the protocol as a monolithic structure. Instead, we identify the structure described above and give security models for each type of keys that are derived in the protocol. A benefit that follows from this modular approach is that we split the analysis of the overall protocol into the analysis of its components, thus making the task of proving security more manageable. In addition, unlike in prior work, we model the realistic setting where only one of the parties involved in the protocol is required to possess a certified public key.

We first provide a model for pre-master key agreement protocols. The model is a weakened version of the Blake–Wilson, Johnson and Menezes (BJM) model [11]. We require from pre-master keys rather weak security guarantees: a man-in-the-middle adversary should not be able to obtain the entire pre-master key (one-wayness).

For master-key agreement protocols, we strengthen the above requirement in that the adversary should not be able to mount an unknown-key-share attack (where parties share a key without being aware of the identity of each other). Secrecy for master keys is still in the one-wayness sense. In addition, we introduce key-confirmation as a requirement for master keys.

Finally, via a further extension, we obtain a model for the security of application key agreement protocols. Our model for application key security is rather standard and resembles the BJM model: we require for the established key to be indistinguishable from a randomly chosen one, and we give the adversary complete control over the network, and various corruption capabilities. Furthermore, the model captures explicitly the possibility that the same master key is used to derive multiple application keys.

Security Analysis of the TLS Handshake Protocol. Based on the models that we developed, we give a security proof for the TLS handshake protocol. We emphasise that we analyse a version where the MAC sent in Step (4) is passed in the clear (and not encrypted under the application keys as in full TLS). It is intuitively clear that the security of the full TLS protocol follows from our analysis (encrypting the MAC should only offer more security to the protocol). While a direct analysis of the latter may be desirable, we choose to trade immediate applicability of our results to full TLS for the modularity afforded by our abstraction: assuming that the MAC is sent in the clear decouples the master-key agreement stage from that of application-key derivation stage.

Our proof is modular and generic. Specifically, we show that the protocol $(\Pi; \mathsf{MKD}_{\mathsf{TLS}}(\mathsf{Mac}, G))$, obtained by appending to an arbitrary pre-master key agreement protocol Π the flows in phase (4) of TLS, is a secure master-key agreement protocol in the sense that we define in this paper. The result holds provided that the message authentication code used in the transformation is secure and the hash function in the construction is modelled as a random oracle. Similarly, we show that starting from an arbitrary secure master-key agreement protocol Π, the protocol $(\Pi; \mathsf{AK}_{\mathsf{TLS}}(H))$, obtained by appending the flows in phase (5) of TLS, is a secure application-key agreement protocol (provided that H is modelled as a random oracle).

An important benefit of the modular approach that we employ surfaces at this stage: to conclude the security of the overall protocol it is sufficient to show that the individual pre-master key agreement protocols of TLS are indeed secure (in the weak sense that we put forth in this paper). The analysis is thus more manageable, and avoids duplicating and rehashing proof ideas, which would be the case if one was to analyse TLS in its entirety for each distinct method for establishing pre-master keys.

We analyse two methods for establishing pre-master keys withing TLS. Encryption-based key transport allows one party to select the pre-master key and to send it encrypted with the public key of the intended recipient. Here we show that *deterministic* encryption that is either one-way (i.e. trapdoor-permutation) or probabilistic OW-CCA encryption suffice to guarantee security. The second option uses Diffie–Hellman keys with authentication implemented via digital signatures. In this case we demonstrate that sufficient conditions for security are the gap-DH assumption and standard security for the signature schemes. We comment on some practical implications of our results next.

Impact on Practice. An implication of practical consequence of our analysis concerns the use of encryption for implementing the pre-master key agreement protocol of TLS. Currently, key-transport in TLS is implemented via RSA PKCS-v1.5, a randomised padding-based encryption used to avoid known problems with vanilla RSA. The exact choice of padding scheme is historic, since the creation of a semantically secure encryption scheme was known to require a probabilistic encryption method. Thus the designers considered using a completely deterministic encryption scheme would introduce security problems. It turns out that the encryption scheme from PKCS-v1.5 is not in fact IND-CCA secure. This was exploited in the famous reaction attack by Bleichenbacher [13] on SSL, where invalid ciphertext messages were used to obtain pre-master secret keys.

As explained above, our results imply that either textbook RSA (thus completely dropping the padding mechanism) or an OW-CCA scheme ensures secure pre-master key agreement. We wish to emphasise that the overall security of the protocol is then guaranteed, provided that one accepts modelling the key-derivation used by TLS as a random oracle. Indeed, our proofs makes crucial use of this assumption, and it is unclear what are the guarantees when key-derivation is implemented via a PRF (as called for by the specification). Since an IND-CCA scheme is also OW-CCA, our results do imply that in the random oracle model TLS where key-transport is implemented with RSA-OAEP [7] (or any other IND-CCA secure scheme in the RO model) is a good application key protocol.

Importantly, since RSA-PKCS-v1.5 implemented in TLS is neither deterministic nor IND-CCA, our analysis does not shed light on its use as a key transport mechanism. Clarifying what are the security guarantees in this case is an interesting research problem of clear practical interest.

We comment that we do not address the security of the protocol when the pre-master key is agreed using password, or other shared key techniques. Also, we focus on the case of one-way authentication (as opposed to mutual authentication), as this is the most common setting in which the protocol is used nowadays. Our definition and analysis can be adapted to the case of mutual authentication.

Our model does not require that the application keys satisfy a notion of key-confirmation (as we require for the master-keys). Indeed, the TLS protocol does not ensure this property. However, one may obtain implicit key confirmation through the use of such keys in further applications. In some sense, this loss is a by-product of the way we have broken up the protocol. One of our goals was to show what security properties each of the stages provides, and therefore we modelled and analysed the security of the application keys. However, if one considers Stages 1–4 as the key agreement protocol and stages 5–6 as the application where the keys are used, then one does obtain an explicit notion of key confirmation. Hence, the loss of explicit key confirmation in Stage 5 should not be considered a design flaw in TLS.

On the Use of the Random Oracle Model. In our proofs we assume that the key derivation function is a random oracle, i.e. an idealised randomness extractor. This assumption is crucial for our proofs, and it is unclear what are the security guarantees that our results imply for the standard model where the key derivation function used is a PRF (as required by the TLS specification).

A natural and important question is whether a standard model analysis is possible, ideally, assuming that the key derivation function is pseudo-random (as is the function based on HMAC used in the current specification of TLS). Unfortunately, indirect evidence indicates that it should be hard to obtain such a result. As observed by Jonsson and Kaliski in their analysis of the use of RSA in TLS [29], the use of the key derivation function in TLS is akin to the use of such functions in deriving DEM keys under the

KEM/DEM paradigm [19]. It is thus likely that a proof as above would immediately imply an efficient RSA-based encryption scheme secure in the standard model, thus solving a long-standing open question in cryptography. We discuss in the related work section some recent progress in this direction by Gajek et al. [25]. Furthermore, when the agreed pre-master key is a Diffie–Hellman key, a generic pseudorandom function is not sufficient to ensure security. A result that could enable a proof in the standard model is the recent work of Fouque et al. [24], who show that the particular function used in TLS is a good randomness extractor.

Related Work

The work which is closest with ours is the analysis of the use of RSA in TLS by Jonsson and Kaliski [29]. They consider a very simplified security model for the master secret key, for the particular case where the protocol for pre-master key is based on encryption. We share the modelling of the key derivation function as a random oracle, and the observation that deterministic encryption may suffice for a secure pre-master key had also been made there. However, the present work uses a far more general and modular model for key-exchange, analyses several pre-master key agreement protocols, including one based on DH keys which is offered by TLS.

Two recent papers present security results regarding TLS. Bhargavan et al. [9] used automation to verify the security of various parts of TLS with respect to computational security properties. They concentrate on the case of encryption-based pre-master key agreement, and separate the analysis of the different parts. As a result, they do not obtain security results regarding TLS overall. They do obtain security results for an actual (functional) implementation of the protocol.

An analysis of the TLS protocol that uses the universal composability framework has been proposed by Gajek et al. [25]. The main drawback of that analysis is that security is only considered against adversaries that corrupt parties statically (prior to protocol execution). Furthermore, here as in other simulation based analysis of key-exchange protocols, security of keys when corruption occurs adaptively posses serious problems. A noteworthy aspect of that work is that it does not make direct use of random oracles, and in particular the key-derivation function used is a PRF. This indicates that should one be able to deal with adaptive corruption, an analysis of TLS in the standard model could potentially be done.

Other analyses of the TLS protocol used Dolev–Yao models, where ideal security of the underlying primitives is postulated, and thus no guarantees are offered for the more concrete world. Such analyses include the one carried out by Mitchell, Shmatikov, and Stern [33] using a model checker, and the one of Paulson who used the inductive method [34]. Wagner and Schneier analyse various security aspects of SSL 3.0 [36], but their treatment is informal. Finally, Bellare and Namprempre [4] and Krawczyk [30] study how to correctly use the application keys derived via TLS. Their treatment is focused exclusively on the use of keys and is not concerned with the security of the entire key agreement protocol.

The first complexity theoretic model for key agreement was the Bellare–Rogaway (BR) model [6,8]. The main driving forces of this model were the works of [10,22]. Since the initial work of Bellare and Rogaway there have been a number of other models proposed for key-exchange in various applications and environments [1,3,5,11,12,14–16,32,35]. These models can be loosely categorised into two main groups: those that use simulation based techniques [3,16,35], and those closer to the original BR model that use an indistinguishability based approach [11,12,15,32].

Some aspects of other indistinguishability-based models relevant to our work are the following. In [6] entity authentication and authenticated key distribution are considered in the two-party symmetric key case where users are modelled as message driven oracles. The adversary in this case acts as the communications channel between users. To define security, the notions of an "error-free history" of [10] and of "matching protocol runs" from [22] are made formal in [6] using the notion of a *matching conversation*. We use this notion in our definitions.

Various security attributes are then included in the definition of security by allowing the adversary to make corresponding queries such as Reveal queries. In [8] this was developed to model the three-party symmetric key case for entity authentication and key distribution. The BJM model of [11] extended the BR model, to authenticated key agreement (AK) and authenticated key agreement with key confirmation (AKC) in the public key case. The work of [11] uses the notion of a No-Matching condition [6], to define a clearer separation between AK and AKC protocols and deals with Diffie–Hellman (DH) like protocols.

As explained before, our analysis uses a model that falls in the original BR category which, as argued elsewhere [15], has certain drawbacks but also several important benefits over the simulation-based approach. The particular models most relevant to our work are the Blake-Wilson, Johnson and Menezes (BJM) based models [11,12,32]. Our execution models are inspired by the BJM model, but our security definitions differ according to the particular key within TLS we consider.

Following on from the BJM models [12] deals with the case of key transport using public key encryption (PKE) and key agreement using Diffie–Hellman key agreement with digital signatures (DSS). In [32] a modular proof technique was used in a modified BJM model to prove security of key agreement protocols relative to a gap assumption. Indeed, the idea of transforming a one-way security definition into an indistinguishability definition also occurs in the generic transform proposed by Kudla and Paterson [31,32], and our proof techniques are similar to theirs.

Finally, an important security model that is related to ours is that of Canetti and Krawczyk (CK) [15]. In addition to the corruption capabilities that we consider, the CK model allows the adversary to obtain the entire internal state of a session and in particular the ephemeral secrets used in sessions. As pointed out by Choo et al., this type of query is the only essential difference between the adversarial capabilities in the model of Bellare and Rogaway and that of Canetti and Krawczyk (see Table 2 of [18]). Clearly, our analysis does not offer guarantees in the face of such extremely powerful types of adversaries, and in fact it can be easily seen that under such attacks the TLS version that uses the DDH-based pre-master secret key agreement is insecure. It may be possible to demonstrate security of TLS under such stronger attacks by assuming secure erasures as done for similar protocols [15,16]. By adopting the style of the BR models over the style of the CK model we also avoid some of the idiosyncrasies of the latter related to the use of session identifiers (which need to be unique, and somehow agreed upon in advance by participating parties) [15,18]. For further discussion on the use of identifiers in the CK model versus the BR model, see [18].

Our work heavily relies on the modular structure that exists in TLS to offer a manageable analysis. This structure is not particular to TLS, it exists in other key exchange protocols, and had been used before in those contexts. For example, the particular phases for exchanging pre-master keys via key transport, and refreshing applications keys (very similar to those in TLS) existed since as early as SKEME [26]. These have later been identified, analysed, and used in a modular analysis of key exchange protocols by Canneti and Krawczyk [15].

The UC analysis of Gajek et al. [25] also follows this line of research. They analyse the master key agreement protocol in a UC manner, and then conclude that it can be composed with arbitrary protocols, in particular with the module for obtaining application keys.

Finally, we mention the general framework recently proposed by Herzberg and Yoffe. A general approach towards systems constructed in a layered manner with lower layers that provide service to upper ones had recently been put forth by Herzberg and Yoffe [27]. It seems that the way that we split the protocol does fit well with their framework and could serve as an interesting test case.

One other aspect of [15] which is somewhat related to our work is a modular framework for designing protocols. In the model of [15] one can first develop a secure protocol under the powerful assumption that all communication is authenticated. Then, a secure protocol in the more realistic setting with no authenticated communication is obtained by applying a generic transformation using an *authenticator*. Obviously, the modular structure of TLS that we observe and exploit is of a different nature. In particular it does not seem possible to regard TLS as the result of applying an authenticator to some other protocol.

Paper Overview

The structure of the paper is as follows. Section 2 surveys our notation and the basic cryptographic notions that we use in this paper. In Sect. 3 we recall the main ideas behind the kind of execution models that we use. We then specialise this model to obtain our security model for pre-master key agreement protocols (Sect. 4). We prove that the standard protocols used in SSL/TLS to obtain pre-master keys do indeed meet our security definition. In Sect. 5 we present the security model for the master key agreement and show that the TLS transform turns a secure pre-master key agreement protocol into a secure master key-agreement protocol. Finally, in Sect. 6 we give our model for the security of application keys and show that the TLS transform does indeed produce a secure application key.

2. Preliminaries

In this section we recall the security notions and assumptions that we use in this paper.

2.1. *Notation*

We write $\{0, 1\}^t$ for the set of binary strings of length t and $\{0, 1\}^*$ for the set of binary strings of arbitrary length. If S is a set, we denote the action of sampling an element from S uniformly at random and assigning the result to the variable x by $x \overset{R}{\leftarrow} S$. If A is an algorithm, then we denote the action of running A on inputs $y_1, \ldots, y_n$, with access to oracles $\mathcal{O}_1(\cdot), \mathcal{O}_2(\cdot)$, and then assigning the output to the variable x by $x \leftarrow A^{\mathcal{O}_1(\cdot),\mathcal{O}_2(\cdot)}(y_1, \ldots, y_n)$. Finally, a function $\epsilon(t)$ is said to be *negligible* in the parameter t if $\forall c \geq \mathbb{N}\ \exists\, t_c \in \mathbb{R}_{>0}$ such that $\forall t > t_c, \epsilon(t) < t^{-c}$.

2.2. *Hard Problems*

We let FGps be a family of prime order groups parametrised by $t \in \mathbb{N}$. For this family, we define the function $\mathbb{G}(\cdot)$ which on input $t \in \mathbb{N}$ outputs a tuple $(\mathbb{G}, q, g) \leftarrow \mathbb{G}(t)$ where $\mathbb{G}$ is a description of the group corresponding to the value t, q its order and g a generator.

We then define the computational Diffie–Hellman advantage of an adversary against the family FGps as follows.

Definition 2.1 (Computational Diffie–Hellman Advantage). We define the Computational Diffie–Hellman advantage of an adversary $\mathcal{A}$ against a family of groups FGps as follows:

$$\mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{A},\mathsf{FGps}}(t) = \Pr\big[(\mathbb{G}, q, g) \leftarrow \mathbb{G}(t);\ a, b \overset{R}{\leftarrow} \mathbb{Z}_q^{\times};\ g^{ab} = \mathcal{A}\big(\mathbb{G}, q, g, g^a, g^b\big)\big].$$

We say the CDH assumption holds in the family FGps if the advantage $\mathbf{Adv}^{\mathsf{CDH}}_{\mathcal{A},\mathsf{FGps}}(t)$ of any p.p.t. adversary $\mathcal{A}$ is negligible in t.

The gap Diffie–Hellman assumption for a group states that the above problem is hard for a given group even if the adversary has access to a decision Diffie–Hellman (DDH) oracle $\mathcal{O}^{\mathbb{G}}_{\mathsf{DDH}}$ for that group. Such an oracle takes as inputs triples of the form g^a, g^b, g^c, where g generates the group $\mathbb{G}$ of order q and $a, b, c \in \mathbb{Z}_q$, and answers "yes" if $c = ab$ and "no" otherwise.

Definition 2.2 (The Gap Diffie–Hellman Advantage). The Gap Diffie–Hellman advantage of an adversary $\mathcal{A}$ against the family of groups FGps is defined as follows:

$$\mathbf{Adv}^{\mathsf{gap\text{-}DH}}_{\mathcal{B},\mathsf{FGps}}(t) = \Pr\big[(\mathbb{G}, q, g) \leftarrow \mathbb{G}(t);\ a, b \overset{R}{\leftarrow} \mathbb{Z}_q^{\times};\ g^{ab} = \mathcal{A}^{\mathcal{O}^{\mathbb{G}}_{\mathsf{DDH}}(\cdot)}\big(\mathbb{G}, q, g, g^a, g^b\big)\big].$$

We say the gap-DH assumption holds in the family FGps if the advantage $\mathbf{Adv}^{\mathsf{gap\text{-}DH}}_{\mathcal{A},\mathsf{FGps}}(t)$ of any p.p.t. adversary $\mathcal{A}$ is negligible in t.

2.3. *Cryptographic Primitives*

Here we recall several cryptographic primitives and the security models associated with them that we use.

2.3.1. *Public Key Encryption*

Definition 2.3 (Public Key Encryption Scheme). A public key encryption scheme Enc is given by a triple of algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ such that:

- $\mathcal{G}$ is a p.p.t. key generation algorithm: $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathcal{G}(t)$. It also returns a description of the message and ciphertext spaces, $\mathcal{M}$ and $\mathcal{C}$.
- $\mathcal{E}$ is a p.p.t. or d.p.t. public key encryption algorithm: $c \leftarrow \mathcal{E}_{\mathrm{PK}}(m; r)$, where $m \in \mathcal{M}$ and $c \in \mathcal{C}$.
- $\mathcal{D}$ is a d.p.t. public key decryption algorithm: $m \leftarrow \mathcal{D}_{\mathrm{SK}}(c)$.

For correctness, we require that for all public/private key pairs $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathcal{G}(t)$ and all $m \in \mathcal{M}$ that $\mathcal{D}_{\mathrm{SK}}(\mathcal{E}_{\mathrm{PK}}(m; r)) = m$.

If $\mathcal{E}$ is a p.p.t., then Enc is called a *probabilistic* public key encryption scheme, and if $\mathcal{E}$ is a d.p.t., then Enc is called a *deterministic* public key encryption scheme.

We will be concerned with security of a public key encryption scheme in a one-way sense under both chosen plaintext attack and chosen ciphertext attack (OW-CPA and OW-CCA respectively). We define the advantage of adversary $\mathcal{B}$ against the OW-CPA security of a public key encryption scheme $\mathsf{Enc} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ by

$$\begin{aligned}&\mathbf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathcal{B},\mathsf{Enc}}(t)\\ &\quad = \Pr\left[(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathcal{G}(t);\ m^* \xleftarrow{R} \mathcal{M};\ c^* \leftarrow \mathcal{E}_{\mathrm{PK}}(m^*; r^*);\ \mathcal{B}(\mathrm{PK}, c^*) = m^*\right].\end{aligned}$$

We say that a public key encryption scheme Enc is OW-CPA secure if the advantage of any polynomial-time adversary is a negligible function in t.

If adversary $\mathcal{B}$ has access to a decryption oracle $\mathcal{O}^{\mathrm{SK}}_{\mathcal{D}}(c)$ that returns the decryption of any valid $c \neq c^*$ under SK, then we define the advantage of $\mathcal{B}$ by

$$\begin{aligned}&\mathbf{Adv}^{\mathsf{OW\text{-}CCA}}_{\mathcal{B},\mathsf{Enc}}(t)\\ &\quad = \Pr\left[(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathcal{G}(t);\ m^* \xleftarrow{R} \mathcal{M};\ c^* \leftarrow \mathcal{E}_{\mathrm{PK}}(m^*; r^*);\ \mathcal{B}^{\mathcal{O}^{\mathrm{SK}}_{\mathcal{D}}(\cdot)}(\mathrm{PK}, c^*) = m^*\right].\end{aligned}$$

We say that a public key encryption scheme Enc is OW-CCA secure if the above advantage of any polynomial-time adversary is a negligible function in t.

2.3.2. *Digital Signatures*

Definition 2.4 (Public Key Signature Scheme). A public key signature scheme Sig is given by a triple of algorithms $(\mathcal{G}, \mathrm{sig}, \mathrm{ver})$ such that:

- $\mathcal{G}$ is a p.p.t. key generation algorithm: $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathcal{G}(t)$.
- sig is a p.p.t. public key signature algorithm: $\sigma \leftarrow \mathrm{sig}_{\mathrm{SK}}(m)$.
- ver is a d.p.t. public key verification algorithm: $\mathrm{ver}_{\mathrm{PK}}(m, \sigma)$, which returns *true* if the pair (m, σ) corresponds to a valid message signature pair and *false* otherwise.

We require that for all public/private key pairs $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathcal{G}(t)$ and all m that $\mathrm{ver}_{\mathrm{PK}}(m, \mathrm{sig}_{\mathrm{SK}}(m)) = \mathit{true}$.

To define the security of a public key signature scheme under chosen message attack we allow adversaries access to a signature oracle $\mathcal{O}^{\mathrm{SK}}_{\mathrm{sig}}(m)$ which returns a valid message/signature pair for m under SK. If $\mathsf{Sig} = (\mathcal{G}, \mathrm{sig}, \mathrm{ver})$ is a public key signature and

$\mathcal{C}$ an adversary against this scheme in terms of strong existential forgery under adaptive chosen message attack SEF-CMA, then we define the advantage of $\mathcal{C}$ by

$$\mathbf{Adv}_{\mathcal{C},\mathsf{Sig}}^{\mathsf{SEF\text{-}CMA}}(t) = \Pr\big[(\mathrm{PK},\mathrm{SK}) \leftarrow \mathcal{G}(t);\ \mathrm{ver}_{\mathrm{PK}}\big((m,s) = \mathcal{C}^{\mathcal{O}_{\mathrm{sig}}^{\mathrm{SK}}(\cdot)}(\mathrm{PK})\big) = \mathrm{true}\big].$$

We say that the scheme is strongly unforgeable if for any probabilistic polynomial-time adversary, its advantage is a negligible function. Above, we ask that the message/signature pair (m,s) output by the adversary is such that the signature s had not been output by the signing oracle on query m.

2.3.3. *Message Authentication Codes*

Definition 2.5 (Message Authentication Code). A message authentication code Mac is given by a triple of polynomial-time algorithms $(\mathcal{K}, \mathrm{MAC}, \mathrm{ver})$ such that:

- $\mathcal{K}$ is a p.p.t. key generation algorithm $K \leftarrow \mathcal{K}(t)$.
- MAC is a d.p.t. tag generation algorithm $\mathrm{tag} \leftarrow \mathrm{MAC}_K(m)$.
- ver is a d.p.t. tag verification algorithm $\{\mathrm{accept}, \mathrm{reject}\} \leftarrow \mathrm{ver}_K(m, \mathrm{tag})$.

We require that for all $K \leftarrow \mathcal{K}(t)$ and $\mathrm{tag} \leftarrow \mathrm{MAC}_K(m)$ that $\mathrm{ver}_K(m, \mathrm{tag}) = \mathrm{accept}$.

To define the security of a message authentication code under chosen message attack, we allow adversaries access to two oracles. The first oracle is a tag generation oracle $\mathcal{O}_{\mathrm{MAC}}^{K}(m)$ which returns a valid tag for the message m. The second oracle is a tag verification oracle $\mathcal{O}_{\mathrm{ver}}^{K}(m, \mathrm{tag})$ which returns accept if tag is a valid tag for the message m under the key K and reject otherwise.

If $\mathsf{Mac} = (\mathcal{K}, \mathrm{MAC}, \mathrm{ver})$ is a message authentication code and $\mathcal{A}$ is an adversary against Mac in terms of recovering the underlying key using a chosen message attack (KR-CMA), then we define the advantage of $\mathcal{A}$ by

$$\mathbf{Adv}_{\mathcal{A},\mathsf{Mac}}^{\mathsf{KR\text{-}CMA}}(t) = \Pr\big[K \leftarrow \mathcal{K}(t);\ K = \mathcal{A}^{\mathcal{O}_{\mathrm{MAC}}^{K}(\cdot),\mathcal{O}_{\mathrm{ver}}^{K}(\cdot,\cdot)}\big].$$

The above is a nonstandard security definition for a Mac; however, a more standard definition is given by an adversary whose goal is to produce an existential forgery under chosen message attack. Let $\mathcal{A}$ denote such a UF-CMA adversary; then we define the advantage of $\mathcal{A}$ by

$$\mathbf{Adv}_{\mathcal{A},\mathsf{Mac}}^{\mathsf{UF\text{-}CMA}}(t) = \Pr\big[K \leftarrow \mathcal{K}(t);\ \mathrm{ver}_K\big((m,\mathrm{tag}) = \mathcal{A}^{\mathcal{O}_{\mathrm{MAC}}^{K}(\cdot),\mathcal{O}_{\mathrm{ver}}^{K}(\cdot,\cdot)}\big) = \mathrm{accept}\big].$$

3. A Generic Execution Model for Two-Party Protocols

In this section we give a general description of the execution model we use. Later, we specialise this general model for the different security levels we consider in the paper according to the various keys agreed within TLS.

Registered and Unregistered Users. We model a setting with two kinds of users: registered users (with identities in some set $\mathcal{U}$) and nonregistered users (with identities in some set $\mathcal{U}'$). Each user $U \in \mathcal{U}$ has a long-term public key PK_U and a corresponding long-term private key SK_U. The set $\mathcal{U}$ is intended to model the set of servers in the standard one-way authentication mode of TLS, the set of identities $\mathcal{U}'$ models users that do not have a long-term public/private key pair.

Models for Interactive Protocols Execution. We are concerned with two-party protocols: interactive programs in which an initiator and a responder communicate via some communication channel. Each of the two parties runs some reactive program: each program expects to receive a message from the communication channel, computes a response, and sends this back to the channel. We refer to one execution of the program for the initiator (respectively, responder) as an initiator session (respectively, a responder session). Each party may engage in multiple, concurrent, initiator and responder sessions.

As standard, we assume an adversary in absolute control of the communication network: the adversary intercepts all messages sent by parties and may respond with whatever message it wants. This situation is captured by considering an adversary (an arbitrary probabilistic, polynomial-time algorithm) who has access to oracles that correspond to some (initiator or responder) sessions of the protocol which the oracle maintains internally. In particular, each oracle maintains an internal state which consists of the variables of the session to which it corresponds, and additional meta-variables used later to define security notions. In our descriptions we typically ignore the details of the local variables of the sessions, and we omit a precise specification of how these sessions are executed. Both notions are standard. The typical meta-variables of an oracle $\mathcal{O}$ include the following. Variable $\tau_{\mathcal{O}} \in \{0,1\}^* \cup \{\perp\}$ that maintains the transcript of all messages sent and received by the oracle, and occasionally, other data pertaining to the execution. Variable $\mathrm{role}_{\mathcal{O}} \in \{\mathit{initiator}, \mathit{responder}, \perp\}$ records the type of session to which the oracle corresponds. Variable $\mathrm{pid}_{\mathcal{O}} \in \mathcal{U}$ keeps track of the identity of the intended partner of the session maintained by $\mathcal{O}$. Variable $\delta_{\mathcal{O}}$ indicates whether the session had finished successfully, or unsuccessfully. We specify the values that this variable takes later in the paper. Finally, variable $\gamma_{\mathcal{O}} \in \{\perp, \mathit{corrupted}\}$ records whether or not the session had been corrupted by the adversary.

After an initialisation phase, in which long-term keys for the parties are generated, the adversary takes control of the execution which he drives forward using several types of queries. The adversary can create a new session of user U playing the role of the initiator/responder by issuing a query $\mathsf{NewSession}(U, \mathit{role})$, with $\mathit{role} \in \{\mathit{initiator}, \mathit{responder}\}$. User U can be either registered or unregistered. We write Π_U^i for the ith session of user U. To any oracle $\mathcal{O}$ the adversary can send a message msg using the query $\mathsf{Send}(\mathcal{O}, \mathrm{msg})$. In return the adversary receives an answer computed according to the session maintained by $\mathcal{O}$. The adversary may also corrupt oracles. Later in the paper, when we specialise the general model, we clarify the different versions of corruptions that can occur and how are they handled by the oracles. The execution halts whenever the adversary decides to do so.

To identify sessions that interact with each other we use the notion of matching conversations introduced by Bellare and Rogaway (which essentially states that the inputs to one session are outputs of the other sessions, and the other way around) [6]. Note that, unlike [11], we do not use the notion of matching conversation with appendix, despite us working in the public-key model.

Definition 3.1 (Conversation). For a given adversary $\mathcal{A}$ and a given oracle Π_U^i, we define its *conversation*, C_U^i, to be a sequence of tuples

$$C_U^i = (\mathsf{t}_1, \alpha_1, \beta_1), (\mathsf{t}_2, \alpha_2, \beta_2), \ldots, (\mathsf{t}_m, \alpha_m, \beta_m),$$

where $\mathsf{t}_m > \mathsf{t}_{m-1} > \cdots > \mathsf{t}_2 > \mathsf{t}_1$ are time steps. A given tuple $(\mathsf{t}_t, \alpha_t, \beta_t)$ means that at time t_t oracle Π_U^i was asked α_t by the adversary and responded with β_t and that after $(\mathsf{t}_m, \alpha_m, \beta_m)$ the adversary terminated without asking any further queries to that oracle.

Notice that a conversation is taken to mean a list of queries and responses of a *complete* execution of the protocol for a given oracle. It only exists upon completion of that particular protocol run. Also note that it is not the same thing as a complete transcript: it does not contain information such as the decisions reached and session IDs. Also, if a given oracle Π_U^i has a conversation prefixed by $(\mathsf{t}_1, \lambda, \beta_1)$, then this oracle will have its role set as the initiator, otherwise its role is set as responder.

We then define a matching conversation for the case of a protocol of R moves, where R is odd, as follows (a similar definition can be given in the case of a protocol with an even number of message flows).

Definition 3.2 (Matching Conversation). Let Π be an R move pre-master key agreement protocol where $R = 2\rho - 1$. Let Π_U^i and Π_V^j be two oracles with conversations C_U^i and C_V^j.

(1) We say that C_V^j is a matching conversation to C_U^i if there exist $\mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_R$ and $\alpha_1, \beta_1, \ldots, \beta_{\rho-1}, \alpha_\rho$ such that C_U^i is prefixed by

$$(\mathsf{t}_0, \lambda, \alpha_1), (\mathsf{t}_2, \beta_1, \alpha_2), (\mathsf{t}_4, \beta_2, \alpha_3), \ldots, (\mathsf{t}_{2\rho-4}, \beta_{\rho-2}, \alpha_{\rho-1}), (\mathsf{t}_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho)$$

and C_V^j is prefixed by

$$(\mathsf{t}_1, \alpha_1, \beta_1), (\mathsf{t}_3, \alpha_2, \beta_2), (\mathsf{t}_5, \alpha_3, \beta_3), \ldots, (\mathsf{t}_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}).$$

(2) We say that C_U^i is a matching conversation to C_V^j if there exist $\mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_R$ and $\alpha_1, \beta_1, \ldots, \beta_{\rho-1}, \alpha_\rho$ such that C_V^j is prefixed by

$$(\mathsf{t}_1, \alpha_1, \beta_1), (\mathsf{t}_3, \alpha_2, \beta_2), (\mathsf{t}_5, \alpha_3, \beta_3), \ldots, (\mathsf{t}_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}), (\mathsf{t}_{2\rho-1}, \alpha_\rho, *)$$

and C_U^i is prefixed by

$$(\mathsf{t}_0, \lambda, \alpha_1), (\mathsf{t}_2, \beta_1, \alpha_2), (\mathsf{t}_4, \beta_2, \alpha_3), \ldots, (\mathsf{t}_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho).$$

We then say that Π_U^j has a matching conversation with Π_V^i if the first oracle has conversation C_U^j, the second oracle has conversation C_V^i, and C_V^j matches C_U^i.

4. Pre-Master Key Agreement Protocols

In this section we specialise the general model described above for the case of pre-master key agreement protocols and analyse the security of the pre-master key agreement protocols used in TLS.

As discussed in the introduction, the design of our models is guided by the security properties that the various subprotocols of TLS satisfy. In particular, we require extremely weak security properties for the pre-master secret key. Specifically, we demand that an adversary is not able to *fully* recover the key shared between two honest parties. In its attack the adversary is allowed to adaptively corrupt parties and obtain their long-term secret key, and is allowed to check if a certain string s equals the pre-master secret key held by some honest session. The latter capability models an extremely limited form of reveal queries: our adversary is not allowed to obtain the pre-master secret key of any of the sessions but can only guess (and then check) their values.

The formal model of security for pre-master key agreement protocols extends the general model in Sect. 3 and makes only mild assumptions regarding the syntax of such protocols. Specifically, we assume that the pre-master key belongs to some space $\mathcal{S}_{\mathrm{PMS}}$. This space is often the support set of some mathematical structure such as a group. We require that if t is the security parameter, then $\#\mathcal{S}_{\mathrm{PMS}} \geq 2^t$. Furthermore, we assume that the initiator and responder programs use a variable $s \in \mathcal{S}_{\mathrm{PMS}} \cup \{\bot\}$ that stores the shared pre-master key. The corresponding variable stored by some oracle $\mathcal{O}$ is $s_{\mathcal{O}}$. For pre-master secret key agreement protocols, the internal variable $\delta_{\mathcal{O}}$ stores one of the following values: $\bot$ (the session had not finished its execution), *accepted-pmk* (the session had finished its execution successfully (which in particular means that $s_{\mathcal{O}}$ holds some pre-master session key in $\mathcal{S}_{\mathrm{PMS}}$) or *rejected* (the session had finished its execution unsuccessfully). Unless $\delta_{\mathcal{O}} =$ *accepted-pmk*, we assume $s_{\mathcal{O}} = \bot$.

The corruption capabilities of the adversary discussed above are modelled using queries Corrupt and Check formally defined as follows. When the adversary issues a query Corrupt(U), the following actions take place. If $U \in \mathcal{U}$, then $\mathrm{SK}_{\mathcal{U}}$ is returned to the adversary, and we say that party U had been corrupted. In all sessions $\mathcal{O} = \Pi_U^i$ for some $i \in \mathbb{N}$, the value of $\gamma_{\mathcal{O}}$ is set to *corrupted*, and no further interaction between these oracles and the adversary may take place. Additionally, no further queries NewSession(U, role) are permitted.

When the adversary issues the query Check$(\mathcal{O}, s)$, for $\mathcal{O} = \Pi_U^i$, $i \in \mathbb{N}$, U some uncorrupted party, and $s \in \mathcal{S}_{\mathrm{PMS}}$, then the answer returned to the adversary is true if $\delta_{\mathcal{O}} =$ *accepted-pmk* and $s_{\mathcal{O}} = s$, and false otherwise. When a given oracle is initialised, all values for the internal states are set to $\bot$. At the end of a protocol, the role, partner ID, and oracle state (but not the pre-master key) are recorded in the transcript.

The following definition captures the class of oracles which are valid targets for the attacker using the notion of "fresh oracles". These are uncorrupted oracles who have successfully finished their execution, and have a known intended partner who is also not corrupted.

Definition 4.1 (Fresh Pre-Master Secret Key Oracle). A pre-master secret oracle $\mathcal{O}$ is said to be fresh if all of the following conditions are satisfied:

1. $\gamma_{\mathcal{O}} = \perp$.
2. $\delta_{\mathcal{O}} = \textit{accepted-pmk}$.
3. $\exists V \in \mathcal{U}$ such that V is uncorrupted and $\mathrm{pid}_{\mathcal{O}} = V$.

Security Game for Pre-Master Key Agreement Protocols. We define the security of a pre-master key agreement protocol Π via the following game $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

(1) The challenger, $\mathcal{C}$, generates public/secret key pairs for each user $U \in \mathcal{U}$ (by running the appropriate key-generation algorithm on the security parameter t) and returns the public keys to $\mathcal{A}$.
(2) Adversary $\mathcal{A}$ is allowed to make as many $\mathsf{NewSession}$, Send, Check, and $\mathsf{Corrupt}$ queries as it likes.
(3) At some point, $\mathcal{A}$ outputs a pair $(\mathcal{O}^*, s^*)$, where $\mathcal{O}^*$ is some pre-master secret oracle, and $s^* \in \mathcal{S}_{\mathrm{PMS}}$.

We say the adversary $\mathcal{A}$ wins if its output $(\mathcal{O}^*, s^*)$ is such that $\mathcal{O}^*$ is fresh, and $s^* = s_{\mathcal{O}^*}$. In this case the output of $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\Pi,\mathcal{A}}(t)$ is set to 1. Otherwise the output of the experiment is set to 0. We write

$$\mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t) = \Pr\bigl[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t) = 1\bigr]$$

for the advantage of $\mathcal{A}$ in winning the $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$ game. The probability is taken over all the random coins used in the game. We deem a pre-master secret key protocol secure if the adversary is not able to fully compute the key held by fresh oracles.

Definition 4.2 (Pre-Master Key Agreement Security). A pre-master key agreement protocol is secure if it satisfies the following requirements:

- **Correctness:** If at the end of the execution of a benign adversary, who correctly relays messages, any two oracles which have had a matching conversation hold the same pre-master key, and the key should be distributed uniformly on the pre-master key space $\mathcal{S}_{\mathrm{PMS}}$.
- **Key Secrecy:** A pre-master key agreement protocol Π satisfies $\mathsf{OW\text{-}PMS}$ key secrecy if for any p.p.t. adversary $\mathcal{A}$, its advantage $\mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$ is a negligible function.

Before proceeding, we discuss the strength of our model for the security of pre-master secret keys, and several authentication issues.

Remark 4.1. Our security requirements for pre-master secret key agreement are significantly weaker than the standard requirements for key exchange [6,8]. In particular, we only require secrecy in the sense of one-wayness (not in the sense of indistinguishability from a random key). Furthermore, the corruption abilities of the adversary are severely limited: the adversary cannot obtain (or "reveal") pre-master secrets established by honest parties (even if these parties are not those under the attack).

Remark 4.2. As a consequence of our security requirements, our model may deem protocols that succumb to unknown-key-share attacks [22] secure. In such attacks, two sessions belonging to honest users U and V locally establish the same pre-master secret key, without intentional interaction with each other.

Remark 4.3. Security under our notion guarantees security against man-in-the-middle attacks: a situation where honest parties U and V believe they interact with each other but their pre-master key(s) is in fact shared with the adversary is a security break in our model.

Remark 4.4. Although the resulting security notion is very weak, it turns out that it suffices to obtain good master-key agreement protocols by appropriately designed protocols to derive such keys (e.g. the protocol in Step 4 of the TLS protocol—Fig. 1). More importantly, the weak notion also allows for many simple protocols to be proved secure. For example, in the next section we prove that deterministic encryption is sufficient to construct such protocols.

Remark 4.5. Our model is not concerned with secure establishment of pre-master secret keys between two unauthenticated parties (the oracle that is under attack always has $\mathrm{pid}_{\mathcal{O}} \neq \perp$). While treating this case is possible using the concept of matching conversations to pair sessions, the resulting definition would be heavier and not particularly illuminating. Instead, we concentrate on the situation more relevant to practice where at least one of the parties that take part in the protocol (the server) has a certified public key.

Remark 4.6. As usual, our security model can be easily adapted to the random oracle model by providing the adversary with access to the random oracle (whenever some hash function is modelled as an RO). The same holds true for the rest of the models that we develop in this paper.

We now discuss the security of the two most widely used pre-master secret key agreement protocols used in TLS.

4.1. *Protocols Based on Public-Key Encryption*

A natural, intuitively appealing, construction for pre-master key agreement protocols is based on the following use of an arbitrary public-key encryption scheme Enc. A user selects a pre-master secret key s from an appropriate space and sends to the server the encryption of s under the server's public-key. The server then obtains s as the decryption of the ciphertext that it receives. The resulting protocol, which we denote by $\mathsf{PMK}(\mathsf{Enc})$, is sketched in Fig. 2. Notice that TLS specifies the behaviour of the receiving party for the case where the decryption does not succeed. This is important to ensure that attacks that use information about the result of decryption do not succeed.

The weak security properties that we define for pre-master key agreement protocols enable us to show security of $\mathsf{PMK}(\mathsf{Enc})$ based on weak security requirements for Enc. Indeed, the one-wayness type secrecy for pre-master keys translates to the one-wayness

Alice		Bob
$s \leftarrow \mathcal{S}_{\mathrm{PMS}};\ c = \mathcal{E}_{\mathrm{PK}_B}(s; r)$		
	$\xrightarrow{c}$	
		If $\mathcal{D}_{\mathrm{SK}_B}(c) \notin \mathcal{S}_{\mathrm{PMS}}$ then $s \overset{R}{\leftarrow} \mathcal{S}_{\mathrm{PMS}}$
		Else $s \leftarrow \mathcal{D}_{\mathrm{SK}_B}(c)$

Fig. 2. Public key encryption based pre-master key agreement.

of the encryption function of Enc. This result of our analysis implies, perhaps surprisingly, that one can avoid the use of full-fledged IND-CCA encryption schemes in favour of the much simpler *deterministic* OW-CPA schemes (e.g. textbook RSA). Of course, probabilistic encryption can also be used, but in this case we show security of the associated pre-master secret key protocol based on OW-CCA security. More generally our results hold under the assumption that the encryption scheme is secure against an attacker with access to a plaintext checking oracle. It is therefore not paradoxical that a deterministic scheme suffices but an IND-CPA scheme does not.

The following theorem states that a simple *deterministic* OW-CPA suffices to obtain a secure pre-master key agreement protocol.

Theorem 4.3. *If* Enc *is an* OW-CPA *secure deterministic encryption scheme, then the pre-master secret key agreement protocol* $\Pi = \mathsf{PMK}(\mathsf{Enc})$ *defined above is secure.*

Proof. Correctness follows by inspection.

We next prove that for any adversary A against $\Pi = \mathsf{PMK}(\mathsf{Enc})$, there exists an adversary $\mathcal{B}$ against OW-CPA security of Enc such that

$$\mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\mathsf{PMK}(\mathsf{Enc})}(t) \le \left(n_P \cdot n_S \cdot \left(n_P + n'_P\right)\right) \cdot \mathbf{Adv}^{\mathsf{OW\text{-}CPA}}_{\mathcal{B},\mathsf{Enc}}(t),$$

where n_P (resp. n'_P) is a bound on the number of participants in $\mathcal{U}$ (resp. $\mathcal{U}'$), and n_S is a bound on the number of sessions each participant can engage in.

Let $\mathcal{A}$ be an adversary against the OW-PMS security of the pre-master key transport protocol Π of Fig. 2, where the encryption scheme is OW-CPA secure. The algorithm $\mathcal{B}$ against the OW-CPA security of $\mathsf{Enc} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ is then constructed as follows.

The algorithm $\mathcal{B}$ is given as input a public key $\mathrm{PK}^\dagger$ and a target ciphertext, $c^\dagger$ as part of the OW-CPA game against Enc. Next $\mathcal{B}$ acts as a challenger to $\mathcal{A}$ in an $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$ game. To do this $\mathcal{B}$ generates n_P identities $\mathcal{U}$ and n'_P identities $\mathcal{U}'$. Then $\mathcal{B}$ selects an element $V^\dagger \in \mathcal{U}$, an element $U^\dagger \in \mathcal{U} \cup \mathcal{U}'$ and an integer $i^\dagger \in \mathbb{Z}^\times_{n_S}$. The public key of $V^\dagger$ is set to be $\mathrm{PK}^\dagger$. Then $\mathcal{B}$ runs the key generation algorithm of the public key scheme to obtain public/private key pairs of all elements in $\mathcal{U} \setminus \{V^\dagger\}$. To finish the setup of $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$, algorithm $\mathcal{B}$ calls algorithm $\mathcal{A}$ using this data.

Algorithm $\mathcal{A}$ will then start to make NewSession, Send, Corrupt and Check queries which $\mathcal{B}$ answers as follows:

- If a Corrupt(U) query is made where $U = \mathcal{U}^\dagger$ or $U = V^\dagger$, then $\mathcal{B}$ terminates. Otherwise $\mathcal{B}$ responds with the private key value (if $U \in \mathcal{U}$), and $\mathcal{A}$ no longer can make queries to oracles belonging to participant U.

- If the adversary makes a $\mathsf{Send}(\mathcal{O}, \text{msg})$ query and $\mathcal{O} = \Pi^{i^\dagger}_{U^\dagger}$, and this oracle is not the initiator, then algorithm $\mathcal{B}$ terminates, otherwise $\mathcal{B}$ responds to the Send query with the message $c^\dagger$.
- If the adversary makes a $\mathsf{Send}(\mathcal{O}, c)$ query for $\mathcal{O} = \Pi^i_U \neq \Pi^{i^\dagger}_{U^\dagger}$ and $\text{role}_{\mathcal{O}} \neq$ *initiator*, then $\mathcal{B}$ first checks that $\mathcal{D}_{\text{SK}_U}(c) \in \mathcal{S}_{\text{PMS}}$. If this is not the case, then $\mathcal{B}$ selects $s_{\mathcal{O}}$ at random from $\mathcal{S}_{\text{PMS}}$ and sets $\delta_{\mathcal{O}} =$ *accepted-pmk*. Otherwise $\mathcal{B}$ sets $s_{\mathcal{O}} \leftarrow \mathcal{D}_{\text{SK}_U}(c)$ and sets $\delta_{\mathcal{O}} =$ *accepted-pmk*.
- The $\mathsf{Check}(\mathcal{O}, s)$ queries which $\mathcal{A}$ makes can always be answered by $\mathcal{B}$ since the encryption function is deterministic.

If $\mathcal{B}$ does not terminate, then eventually $\mathcal{A}$ will terminate and output a pair $(\mathcal{O}^*, s^*) = (\Pi^{i^*}_{U^*}, s^*)$. Since $\mathcal{U}^\dagger \in \mathcal{U} \cup \mathcal{U}'$, with probability $1/((n_P + n'_P) \cdot n_S)$ we have that $U^* = U^\dagger$ and $i^* = i^\dagger$. Furthermore, we have that $\text{pid}_{\mathcal{O}^*} = V^\dagger$ with probability $1/n_P$. Due to the way $\mathcal{B}$ inserts $c^\dagger$ into the message flows of $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$, the adversary $\mathcal{A}$ will be attempting to find the message behind the ciphertext $c^\dagger$ if and only if all three of these conditions hold. This happens with probability $1/(n_P \cdot (n_P + n'_P) \cdot n_S)$. If, in addition to this, $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t) = 1$, then s^* will actually be the corresponding message behind the ciphertext $c^\dagger$. The algorithm $\mathcal{B}$ then outputs s^* as part of the $\mathsf{OW\text{-}CPA}$ security game against Enc.

Since the simulation provided for $\mathcal{A}$ by $\mathcal{B}$ is perfect if $\mathcal{B}$ does not terminate, the choice of the oracle output by $\mathcal{A}$ is independent of the choices of $\mathcal{B}$, and this provides the stated advantage. □

The next theorem captures that $\mathsf{PMK}(\mathsf{Enc})$ is secure when the encryption algorithm of Enc is a *randomised*. In this case we demand that the encryption scheme be $\mathsf{OW\text{-}CCA}$ secure.

Theorem 4.4. *If* Enc *is an* $\mathsf{OW\text{-}CCA}$ *secure randomised encryption scheme, then* $\mathsf{PMK}(\mathsf{Enc})$ *is a secure pre-master key transport protocol.*

Proof. Correctness is immediate.

We prove that for any adversary $\mathcal{A}$ against $\mathsf{PMK}(\mathsf{Enc})$, there exists an adversary $\mathcal{B}$ against the $\mathsf{OW\text{-}CCA}$ security of Enc such that

$$\mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\mathsf{PMK}(\mathsf{Enc})}(t) \leq \big(n_P \cdot n_S \cdot \big(n_P + n'_P\big)\big) \cdot \mathbf{Adv}^{\mathsf{OW\text{-}CCA}}_{\mathcal{B},\mathsf{Enc}}(t),$$

where, as before, n_P (resp. n'_P) is a bound on the number of participants in $\mathcal{U}$ (resp. $\mathcal{U}'$), and n_S is a bound on the number of sessions each participant can engage in.

The proof is essentially the same as for the previous theorem. The only difference is that the $\mathsf{Check}(\mathcal{O}, s)$ queries to a given oracle are simulated either by performing a valid decryption using the public/private key pair held by the algorithm B (in the case where the recipient is not equal to $V^\dagger$), or are performed using the supplied decryption oracle (when the identity of the recipient is equal to $V^\dagger$). □

Notice that since $\mathsf{IND\text{-}CCA}$ implies $\mathsf{OW\text{-}CCA}$, our security analysis *does* apply to the (correct) use of an $\mathsf{IND\text{-}CCA}$ secure public key encryption scheme within the TLS protocol. In particular, when Enc is RSA-OAEP, the pre-master secret key protocol $\mathsf{PMK}(\mathsf{Enc})$ is secure.

4.2. *Signed Diffie–Hellman Pre-Master Key Agreement*

Let FGps be a family of prime-order groups, and let $\mathsf{Sig} = (\mathcal{G}, \mathrm{sig}, \mathrm{ver})$ be a public key signature scheme. If t is a security parameter, then for $(\mathbb{G}, q, g) \leftarrow \mathbb{G}(t)$, the pre-master secret key in TLS can be produced by exchanging a Diffie–Hellman key g^{xy}, for $x, y \in \mathbb{Z}_q^\times$ randomly chosen by the two participants, who also sign the relevant message flow (either g^x or g^y) with their long-term signing keys. The set of pre-master keys is set to $\mathbb{G}$. The resulting protocol, for which we write $\mathsf{PMK}(\mathsf{Sig}, \mathsf{FGps})$, is sketched in Fig. 3. Notice that we give the protocol for the case of only one party having a public/private key pair.

It is known that this protocol does not meet the requirements of an authenticated key agreement protocol; for example, see [22] for a discussion of this protocol and various attacks on it. We prove however that the protocol does satisfy the security requirements that we put forth for pre-master key agreement protocols.

Theorem 4.5. *Let* FGps *be a family of groups of prime order for which the gap-Diffie–Hellman assumption holds, and let* Sig *be a secure digital signature scheme. Then* $\Pi = \mathsf{PMK}(\mathsf{Sig}, \mathsf{FGps})$ *is a secure pre-master key agreement protocol.*

Proof. Correctness follows by inspection.

We prove that for any adversary $\mathcal{A}$ against $\Pi = \mathsf{PMK}(\mathsf{Sig}, \mathsf{FGps})$, there exists an algorithm $\mathcal{B}$ for the gap-Diffie–Hellman problem in FGps and an adversary $\mathcal{C}$ against Sig such that

$$\mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\mathsf{PMK}(\mathsf{Sig},\mathsf{FGps})}(t) < \mathbf{Adv}^{\mathsf{gap\text{-}DH}}_{\mathcal{B},\mathsf{FGps}}(t) + n_P \cdot \mathbf{Adv}^{\mathsf{SEF\text{-}CMA}}_{\mathcal{C},\mathsf{Sig}}(t),$$

where n_P denotes a bound on the number of participants in the set $\mathcal{U}$.

Let $\mathcal{A}$ be an adversary against the $\mathsf{OW\text{-}PMS}$ security of the signed Diffie–Hellman pre-master key agreement protocol Π of Fig. 3. We define E to be the event that at the end of $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$ the oracle $\mathcal{O}^*$ that $\mathcal{A}$ outputs has on its transcript an incoming message $(g^a, \mathrm{sig}(g^a))$ that was not output by *any* other honest oracle in the game. We

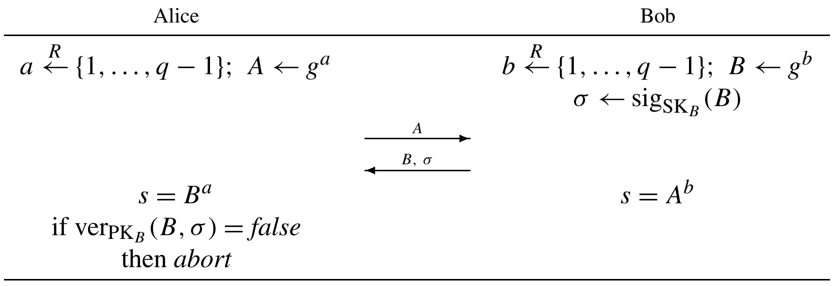
Alice
Bob
$a \xleftarrow{R} \{1, \ldots, q-1\};\ A \leftarrow g^a$
$b \xleftarrow{R} \{1, \ldots, q-1\};\ B \leftarrow g^b$
$\sigma \leftarrow \mathrm{sig}_{\mathrm{SK}_B}(B)$
A
B, σ
$s = B^a$
if $\mathrm{ver}_{\mathrm{PK}_B}(B, \sigma) = \mathit{false}$
then *abort*
$s = A^b$

Fig. 3. Signed Diffie–Hellman based pre-master key agreement.

then note the following:

$$\begin{aligned}\Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}=1\big] &= \Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}=1\cap E\big]+\Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}=1\cap\neg E\big]\\ &= \Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}=1\mid E\big]\cdot\Pr[E]\\ &\quad+\Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}=1\mid\neg E\big]\cdot\Pr[\neg E]\\ &< \Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}=1\mid E\big]+\Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}=1\mid\neg E\big].\end{aligned}$$

We next construct the two algorithms, $\mathcal{B}$ against the gap-DH problem in the family FGps and $\mathcal{C}$ against the SEF-CMA of the underlying signature scheme Sig, according to whether the event E occurs or not.

First assume that the event E does not occur. Then we construct the algorithm $\mathcal{B}$ against the gap-DH problem in FGps as follows. Algorithm $\mathcal{B}$ is given as input a security parameter t, a decisional Diffie–Hellman oracle $\mathcal{O}^{\mathbb{G}}_{\mathsf{DDH}}$ and an instance of the Diffie–Hellman problem $(\mathbb{G},q,g,g^a,g^b)$ in the group $(\mathbb{G},q,g)\leftarrow\mathbb{G}(t)$. Next $\mathcal{B}$ acts as a challenger to $\mathcal{A}$ in an $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$ game. To do this $\mathcal{B}$ generates n_P identities $\mathcal{U}$ and n'_P identities $\mathcal{U}'$. Then $\mathcal{B}$ runs the key generation algorithm of the public key signature scheme Sig with security parameter t to obtain public/private key pairs of all elements in $\mathcal{U}$. To finish the setup of $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}$ algorithm $\mathcal{B}$ calls $\mathcal{A}$ using this data.

Algorithm $\mathcal{A}$ will then start to make NewSession, Send, Check and Corrupt queries which $\mathcal{B}$ answers in the following way:

- If a Corrupt(U) query is made, then $\mathcal{B}$ returns SK_U, and $\mathcal{A}$ may make no further queries of oracles belonging to participant U.
- When $\mathcal{A}$ makes a Send$(\mathcal{O},\mathrm{msg})$ query to an oracle $\mathcal{O}=\Pi^i_U$, then $\mathcal{B}$ generates a random value $r_{\mathcal{O}}\in\{1,\ldots,q-1\}$. If U is an initiator, then $\mathcal{B}$ sets $h=(g^a)^{r_{\mathcal{O}}}$, otherwise $\mathcal{B}$ sets $h=(g^b)^{r_{\mathcal{O}}}$.

  Then if $U\in\mathcal{U}'$ and this is the first message received by that oracle, then $\mathcal{B}$ replies with h. If this is not the first message, then $\mathcal{B}$ responds with $\perp$.

  Otherwise, $U\in\mathcal{U}$, and $\mathcal{B}$ replies with $(h,\mathrm{sig}_{\mathrm{SK}_U}(h))$.
- If the adversary makes a Check$(\mathcal{O},s)$ query, then $\mathcal{B}$ obtains the Diffie–Hellman exchanges transmitted to and from the oracle from the transcript and submits these, along with the value to be checked, to the oracle $\mathcal{O}^{\mathbb{G}}_{\mathsf{DDH}}$. Algorithm $\mathcal{B}$ then relays the response of this to $\mathcal{A}$.

In this way $\mathcal{B}$ can always answer all of the queries that $\mathcal{A}$ makes and will hence perfectly simulate the environment of $\mathcal{A}$. As a result, eventually $\mathcal{A}$ will terminate and output a pair $(\mathcal{O}^*,s^*)=(\Pi^{i^*}_{U^*},s^*)$.

Since we have assumed that the event E does not occur, then there will be an entry of the form $(g^{\alpha r_{\mathcal{O}^\dagger}},\mathrm{sig}_{\mathrm{SK}_{V^\dagger}}(g^{\alpha r_{\mathcal{O}^\dagger}}))$, where $\alpha\in\{a,b\}$, on the transcript of $\mathcal{O}^*$ produced by some oracle $\mathcal{O}^\dagger=\Pi^{i^\dagger}_{V^\dagger}$. Furthermore, if we have $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)=1$, then $s^*=(g^{ab})^{r_{\mathcal{O}^*}r_{\mathcal{O}^\dagger}}$. Therefore algorithm $\mathcal{B}$ can construct the solution to the Diffie–Hellman problem for security parameter t as $(s^*)^{1/(r_{\mathcal{O}^*}r_{\mathcal{O}^\dagger})}$. Hence,

$$\mathbf{Adv}^{\mathsf{gap\text{-}DH}}_{\mathcal{B},\mathsf{FGps}}(t)=\Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}=1\mid\neg E\big].$$

Next we consider the case in which the event E does occur. Here we construct the algorithm $\mathcal{C}$ against the SEF-CMA of the signature scheme used as follows. The algorithm $\mathcal{C}$ is given as input a security parameter t, a public verification key PK and a corresponding signature oracle $\mathcal{O}_{\mathrm{sig}}^{\mathrm{SK}}$.

Algorithm $\mathcal{C}$ acts as a challenger for $\mathcal{A}$ in an $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ game. To do this $\mathcal{C}$ generates n_P identities $\mathcal{U}$ and n'_P identities $\mathcal{U}'$, and it selects an oracle $U^\dagger \in \mathcal{U}$ and sets $\mathrm{PK}_{U^\dagger} = \mathrm{PK}$ and $\mathrm{SK}_{U^\dagger} = \perp$. Then $\mathcal{C}$ runs the key generation algorithm of the public key signature scheme with security parameter t to obtain public/private key pairs of all elements $U \in \mathcal{U} \setminus \{U^\dagger\}$. To finish the setup of $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ the set of identities and public keys are passed to algorithm $\mathcal{A}$.

Algorithm $\mathcal{A}$ will then start to make NewSession, Send, Check and Corrupt queries which $\mathcal{C}$ answers in the following way:

- If a Corrupt(U) query is made and $U \neq U^\dagger$, then $\mathcal{C}$ returns SK_U, and $\mathcal{A}$ can no longer query any oracle belonging to participant U. If $U = U^\dagger$, then $\mathcal{C}$ aborts.
- When $\mathcal{A}$ makes a Send$(\mathcal{O}, \mathrm{msg})$ query to an oracle with identity not equal to $U^\dagger$, then $\mathcal{C}$ responds as in the correct execution of the protocol.

  Otherwise $\mathcal{C}$ and selects a random $x \in \mathbb{Z}_q^\times$, computes g^x, and obtains a signature on g^x using its signature oracle. The reply to the adversary is $(g^x, \mathcal{O}_{\mathrm{sig}}^{\mathrm{SK}}(g^x))$.
- If the adversary makes a Check$(\mathcal{O}, s)$ query, then $\mathcal{C}$ knows the ephemeral Diffie–Hellman secret of the queried oracle and so can compute the associated Diffie–Hellman secret and therefore is able to answer the Check queries honestly.

If $\mathcal{C}$ does not abort, then the environment of $\mathcal{A}$ is perfectly simulated and as a result eventually $\mathcal{A}$ will terminate and output a pair $(\Pi_{U^*}^{i^*}, s^*)$.

If $\mathrm{pid}_{\mathcal{O}^*} = U^\dagger$ and $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t) = 1$, then there is an entry $(h, \mathrm{sig}_{\mathrm{SK}_{U^\dagger}}(h))$ on the transcript $\tau_{\mathcal{O}^*}$ of $\mathcal{O}^*$ that has correctly verified under $\mathrm{PK}_{U^\dagger}$. Furthermore, since the event E has occurred, the entry did not come from $U^\dagger$ (i.e. it was not a message/signature pair that was output by $\mathcal{O}_{\mathrm{sig}}^{\mathrm{SK}}$). As a result, the pair $(h, \mathrm{sig}_{\mathrm{SK}_{U^\dagger}}(h))$ is a valid forgery. Algorithm $\mathcal{C}$ scans $\tau_{\mathcal{O}^*}$ to find this pair and then uses the pair as its output in the game against the SEF-CMA of the signature scheme. Since the choice of U^* is outside the view of the adversary, we obtain that

$$\mathbf{Adv}_{\mathcal{C}}^{\mathsf{SEF\text{-}CMA}}(t) \geq \frac{1}{n_P} \cdot \Pr\left[\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}} = 1 \mid E\right],$$

from which the desired result follows. □

4.3. *Pre-Master Key Agreement from Signcryption Schemes*

In the TLS standard, when used with RSA-based key transport, the mechanism used to provide mutual authentication is for the client to sign its encryption of the pre-master secret under the server's public key. In essence this is using the encrypt-then-sign paradigm of creating a signcryption scheme [2]. By combining techniques of the proofs of the previous theorems it is easy to show that one can prove a similar security result to that above for general signcryption-based key transport mechanisms [37], which also shows security for the mutually authenticated versions of TLS deployed in practice.

Mutual authentication can also be shown when the pre-master key is obtained using signed Diffie–Hellman, based on the security of the digital signature scheme.

5. Master Key Agreement Protocols

In this section we introduce a security model for master key agreement protocols. We then show that master key agreement protocols obtained from secure pre-master key agreement protocols via the transformation used in TLS satisfy our notion of security.

Our security model for master key agreement protocols is similar to that for pre-master key agreement protocols. We again ask that the adversary is not able to *fully* recover the master secret key of the session under attack. Moreover, we ask for a key confirmation guarantee: if a session of some user U accepts a certain master-key, then there exists at most one session of its intended partner that has accepted the same key. In addition to the queries previously defined for the adversary, we also let the adversary obtain the master keys agreed in different sessions of the protocol, without corrupting the user to which this session belongs, i.e. we allow so-called Reveal queries.

In the formal model that we give below we make the following assumptions about the syntax of a master-key agreement protocol. We assume that the master key belongs to some space $\mathcal{S}_{\mathrm{MS}}$ for which we require that $\#\mathcal{S}_{\mathrm{MS}} \geq 2^t$ and assume that the programs that specify a master key agreement protocol use a variable m to store the agreed master key. For such protocols, the variable $\delta_{\mathcal{O}}$ now takes values in $\{\perp, \textit{accepted-mk}, \textit{reject}\}$ with the obvious meaning. Furthermore, the variable $\gamma_{\mathcal{O}}$ can also take the value *revealed* to indicate that the stored master key has been given to the adversary (see below).

In addition to the queries allowed in the experiment for pre-master key security, the adversary is also allowed to issue queries of the form $\mathsf{Reveal}(\mathcal{O})$. This query is handled as follows: if $\delta_{\mathcal{O}} = \textit{accepted-mk}$, then $m_{\mathcal{O}}$ is returned to $\mathcal{A}$, and $\gamma_{\mathcal{O}}$ is set to *revealed*, while if $\delta_{\mathcal{O}} \neq \textit{accepted-mk}$, then the query acts as a no-op. As before, when a given oracle is initialised, all values for the internal states are set to $\perp$. At the end of a protocol the role, partner ID and oracle state (but not the master key) are recorded in the transcript. Unless $\delta_{\mathcal{O}} = \textit{accepted-mk}$, we assume $m_U^i = \perp$.

The definition of freshness needs to be adapted to take into account the new adversarial capabilities. We call an oracle $\mathcal{O}$ fresh if it is uncorrupted, has successfully finished its execution, its intended partner V is uncorrupted, and none of the revealed oracles belonging to V has had a matching conversation with $\mathcal{O}$. The latter condition essentially says that the adversary can issue $\mathsf{Reveal}(\mathcal{Q})$ for any $\mathcal{Q}$ (including those that belong to the intended partner of $\mathcal{O}$), as long as $\mathcal{Q}$ is not the session with which $\mathcal{O}$ actually interacts.

Definition 5.1 (Fresh Master Secret Oracle). A master secret oracle $\mathcal{O}$ is said to be fresh if all of the following conditions hold:

1. $\gamma_{\mathcal{O}} = \perp$.
2. $\delta_{\mathcal{O}} = \textit{accepted-mk}$.
3. $\exists V \in \mathcal{U}$ such that V is uncorrupted and $\mathrm{pid}_{\mathcal{O}} = V$.
4. No revealed oracle Π_V^i has had a matching conversation with $\mathcal{O}$.

Security Game for Master-Key Agreement Protocols. The game, denoted by $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi}(t)$, for defining the security of master-key agreement protocol Π in the presence of adversary $\mathcal{A}$ is similar to that for pre-master key, with the modification that $\mathcal{A}$ is also allowed to make any number of Reveal queries, in addition to the $\mathsf{NewSession}$, Send, $\mathsf{Corrupt}$, Reveal and Check queries. Here, check queries are with respect to the master secret keys only. When the adversary stops, it outputs a pair $(\mathcal{O}^*, m^*)$, where $\mathcal{O}^*$ identifies one of its oracles, and m^* is some element of $\mathcal{S}_{\mathrm{MS}}$. We say that $\mathcal{A}$ wins if its output $(\mathcal{O}^*, m^*)$ is such that O^* is fresh and $m^* = m_{\mathcal{O}^*}$. In this case the output of $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi}(t)$ is set to 1. Otherwise the output of the experiment is set to 0. We write

$$\mathbf{Adv}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi}(t) = \Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi}(t) = 1\big]$$

for the advantage of $\mathcal{A}$ in winning the $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi}(t)$ game. The probability is taken over all random coins used in the execution.

The following definition describes a situation where some party U had engaged in a session which terminated successfully with some party V, but no session of V has a matching conversation with U.

Definition 5.2 ($\mathsf{No\text{-}Matching}$). Let $\mathsf{No\text{-}Matching}_{\mathcal{A},\Pi}(t)$ be the event that at some point during the execution of $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi}(t)$ for two uncorrupted parties $U \in \mathcal{U} \cup \mathcal{U}'$ and $V \in \mathcal{U}$, there exists an oracle $\mathcal{O} = \Pi^i_U$ with $\mathrm{pid}_{\mathcal{O}} = V \in \mathcal{U}$, $\delta_{\mathcal{O}} = \mathit{accepted}$, and yet no oracle Π^i_V has had a matching conversation with $\mathcal{O}$.

The following definition says that a protocol is a secure master-key agreement protocol if the key established in an honest session is secret (in the one-wayness sense) and no honest party can be coaxed into incorrectly accepting.

Definition 5.3 (Master Key Agreement Security). A master key agreement protocol is secure if it satisfies the following requirements:

- **Correctness:** At the end of the execution of a benign adversary, who correctly relays messages, any two oracles which have had a matching conversation hold the same master key, which is distributed uniformly over the master key space $\mathcal{S}_{\mathrm{MS}}$.
- **Key Secrecy:** A master key agreement protocol Π satisfies $\mathsf{OW\text{-}MS}$ key secrecy if for any p.p.t. adversary $\mathcal{A}$, its advantage $\mathbf{Adv}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi}(t)$ is a negligible function.
- **No Matching:** For any p.p.t. adversary $\mathcal{A}$, the probability of event $\mathsf{No\text{-}Matching}_{\mathcal{A},\Pi}(t)$ is negligible.

Remark 5.1. Our security requirements for master secret keys are still significantly weaker than the more standard requirements for key exchange [6,8]. Although the adversarial powers are similar to those in existing models (e.g. [11]), we still require the adversary to recover the entire key. The weaker requirement is motivated by our use of TLS as guide in designing the security model. In this protocol, as well as in others, the master secret key is *not* indistinguishable from a random one since it is used to accomplish additional tasks (e.g. key-confirmation, or various versions of authentication).

Remark 5.2. The No Matching property we require is essentially the one based on matching conversations introduced by Bellare and Rogaway [6], adapted to our setting where only one of the parties involved in the execution is required to hold a certified key (and thus have a verifiable identity). One could potentially replace matching conversations with weaker versions of partnering, but only at the expense of making the definitions and results less clear. Bellare and Rogaway also show that if the No Matching property is satisfied, then agreement is injective. In our terms, with overwhelming probability it holds that if $\mathcal{O} = \Pi_U^i$ had accepted and has $\mathrm{pid}_{\mathcal{O}} = V \in \mathcal{U}$, then there exist precisely one session of V with which $\mathcal{O}$ has a matching conversation.

Remark 5.3. Notice that, together, the first and third conditions in the above definitions imply a key confirmation guarantee: if one session has accepted a certain key, then there exists a unique session of the intended partner who has accepted the same key.

Remark 5.4. The addition of Reveal queries implies security against "unknown-key-share" attacks: if parties U and V share a master-key without being aware that they interact with each other, the adversary can obtain the key of U by performing a Reveal query on the appropriate session of V, thus breaking security in the sense defined above.

Remark 5.5. Notice that an adversary against the master-secret key does not have any query that allows it to obtain information about the pre-master secret key. This is consistent with the TLS specification which states that the pre-master secret should be converted to the master secret immediately and that the pre-master secret should be securely erased from memory. In particular this means that the pre-master secret does not form part of the state of the master key agreement oracle.

In this section we show that the master-key agreement protocol obtained from a secure pre-master key agreement protocol by using the transformation used in TLS is secure. Let Π be an arbitrary pre-master key agreement protocol, G a hash function, and $\mathsf{Mac} = (\mathcal{K}, \mathrm{MAC}, \mathrm{ver})$ a message authentication code. We write $(\Pi; \mathsf{MKD}_{\mathsf{TLS}}(\mathsf{Mac}, G))$ for the master-key agreement protocol obtained by extending Π with the master-key derivation phase of TLS, i.e. by appending to the message flows of Π those in Step 4 of Fig. 1. Starting from a secure pre-master key agreement protocol, the above transformation yields a secure master key agreement protocol.

Theorem 5.4. *Let Π be a secure pre-master agreement protocol,* Mac *a secure message authentication code, and G a random oracle. Then* $(\Pi; \mathsf{MKD}_{\mathsf{TLS}}(\mathsf{Mac}, G))$ *is a secure master-key agreement protocol.*

The correctness of the protocol follows by inspection. We only need to prove the remaining two conditions of Definition 5.3. We prove separately that $(\Pi; \mathsf{MKD}_{\mathsf{TLS}}(\mathsf{Mac}, G))$ is secure in the sense of $\mathsf{OW\text{-}MS}$ (Part 1) and that the probability of the event No-Matching is negligible for all adversaries (Part 2). For clarity of split, the proof across two different subsections.

Proof of Theorem 5.4 (One-wayness). Before going into the details of the proof we give an informal description. An adversary $\mathcal{A}$ against a master secret key agreement protocol can win in one of two ways:

Breaking the PMS: The adversary is able to break the pre-master secret security of the underlying protocol, and so using a G query is able to break the master-secret security of the protocol.

Breaking the MAC: The adversary is able to, for a given message authentication code under an unknown key, compute the key for the message authentication code.

We now formalise the proof. Let $\mathcal{A}$ be an adversary against the OW-MS security of the master key agreement protocol $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{TLS}}(\mathsf{Mac}, G))$.

We define E to be the event that $\mathcal{A}$ outputs an oracle $\Pi_{U^*}^{i^*\,\prime}$ and that $\mathcal{A}$ had, at some point during the security game $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}$, made a query to G of the form $G(s^*, r_a^*, r_b^*)$, where s^* is the pre-master secret of $\Pi_{U^*}^{i^*}$, and r_a^*, r_b^* are the random strings exchanged after s^* was agreed that are on the transcript of $\Pi_{U^*}^{i^*\,\prime}$. We have that

$$\begin{aligned}
\Pr\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1\big] &= \Pr\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \cap E\big] + P\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \cap \neg E\big] \\
&= \Pr\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \mid E\big] \cdot \Pr[E] \\
&\quad + \Pr\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \mid \neg E\big] \cdot \Pr[\neg E] \\
&< \Pr\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \mid E\big] + \Pr\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \mid \neg E\big].
\end{aligned}$$

The desired result thus follows from the following lemma which captures the intuition above.

Lemma 5.5. *Let Π denote a pre-master key agreement protocol, and let the derived master key agreement protocol be denoted by $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{TLS}}(\mathsf{Mac}, G))$. Let the event E be as described above. Then if $\mathcal{A}$ is an adversary against the* OW-MS *security of Π', then*

1. *There is an adversary $\mathcal{B}$ against the* OW-PMS *security of Π such that*

$$\Pr\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \mid E\big] = \mathbf{Adv}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t) + \frac{n_S(n_P + n'_P) + n_G}{2^t}.$$

2. *There is an adversary $\mathcal{C}$ against the* KR-CMA *security of the MAC such that*

$$\Pr\big[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \mid \neg E\big] \le \big(\big(n_P + n'_P\big) \cdot n_S\big) \cdot \mathbf{Adv}_{\mathcal{C},\mathsf{Mac}}^{\mathsf{KR\text{-}CMA}}(t),$$

where n_P, n'_P and n_S are as before, and n_G denotes the maximum number of queries made by $\mathcal{A}$ to the random oracle G.

Proof. We distinguish two cases, depending on whether event E occurs or not.

Event E Occurs. Let $\mathcal{D}$ be a challenger in an OW-PMS security game $\mathsf{Exec}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ of Π against $\mathcal{B}$. We then construct the adversary $\mathcal{B}$ against $\mathcal{D}$ as follows. For a given security parameter t, the challenger $\mathcal{D}$ generates n_P identities $\mathcal{U}$ and n'_P identities $\mathcal{U}'$, and then obtains public keys for each element in $\mathcal{U}$. Algorithm $\mathcal{D}$ then passes $\mathcal{U}' \cup \mathcal{U}$ and the set of public keys to $\mathcal{B}$.

Algorithm $\mathcal{B}$ acts as a challenger in an $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}(t)$ game against $\mathcal{A}$. In order to answer the queries of $\mathcal{A}$ in a consistent manner, $\mathcal{B}$ creates an, initially empty, list G-List. The entries of G-List are tuples of the form $(s, r_a, r_b, m, \mathcal{O}, \mathcal{O}')$, where $s \in \mathcal{S}_{\mathrm{PMS}} \cup \{\perp\}$, $r_a, r_b \in \{0,1\}^t \cup \{\perp\}$, $m \in \mathcal{S}_{\mathrm{MS}}$, and $\mathcal{O}, \mathcal{O}' \in \{\Pi_U^i\} \cup \{\perp\}$. The entry $\mathcal{O}$ corresponding to a client, i.e. a party who transmitted r_A and received r_B, whereas $\mathcal{O}'$ corresponding to a server, i.e. a party who transmitted r_B and received r_A. To complete the setup of $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}(t)$ algorithm $\mathcal{B}$ passes $\mathcal{U} \cup \mathcal{U}'$ and the set of public keys to $\mathcal{A}$.

In the following we assume that the Algorithm $\mathcal{B}$ maintains transcripts for each oracle as part of master secret game $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}$ by using copies of the transcripts from the pre-master secret game $\mathsf{Exec}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ and appending any additional messages to this. We denote τ' the transcript formed from the pre-master secret transcript τ with any additional messages appended.

We define the **GetKey** algorithm as Algorithm 1. Note that this is only called for un-corrupted oracles $\mathcal{O}$. We also define an algorithm to answer random oracle queries to G as in Algorithm 2.

Algorithm 1: **GetKey** Algorithm.

```
Input: A tuple $(r_a, r_b, \mathcal{O})$, with $r_a$ and $r_b$ being the random strings exchanged
       on the transcript of $\mathcal{O}$
Output: A value $m \in \mathcal{S}_{\mathrm{MS}}$
1  if $\mathcal{O}$ is a client and $\exists\ (s, r_a, r_b, m, \perp, *) \in$ G-List with $\mathsf{Check}(\mathcal{O}, s) = \mathsf{true}$ then
2      replace $(s, r_a, r_b, m, \perp, *)$ with $(s, r_a, r_b, m, \mathcal{O}, *)$;
3      return $m$;
4  else if $\mathcal{O}$ is a server and $\exists\ (s, r_a, r_b, m, *, \perp) \in$ G-List with $\mathsf{Check}(\mathcal{O}, s) = \mathsf{true}$
   then
5      replace $(s, r_a, r_b, m, *, \perp)$ with $(s, r_a, r_b, m, *, \mathcal{O})$;
6      return $m$;
7  else if $\exists\ (s, r_a, r_b, m, *, *) \in$ G-List with $\mathsf{Check}(\mathcal{O}, s) = \mathsf{true}$ then
8      return $m$;
9  else
10     $m \xleftarrow{R} \mathcal{S}_{\mathrm{MS}}$;
11     if $\mathcal{O}$ is a client then
12         add $(\perp, r_a, r_b, m, \mathcal{O}, \perp)$ to G-List;
13     else
14         add $(\perp, r_a, r_b, m, \perp, \mathcal{O})$ to G-List;
15     return $m$;
```

Algorithm 2: G Query Responses.

```
Input: $(s, r_a, r_b)$
Output: A value $m \in \mathcal{S}_{\mathrm{MS}}$
1  if $s, r_a, r_b$ are not such that $r_a, r_b \in \{0,1\}^t$ and $s \in \mathcal{S}_{\mathrm{PMS}}$ then
2      return $\perp$;
3  else if $\exists(s, r_a, r_b, m, *, *) \in$ G-List then
4      return $m$;
5  else if $\exists(\perp, r_a, r_b, m, \mathcal{O}, *) \in$ G-List and $\mathsf{Check}(\mathcal{O}, s) = \mathsf{true}$ then
6      replace $(\perp, r_a, r_b, m, \mathcal{O}, *)$ with $(s, r_a, r_b, m, \mathcal{O}, *)$;
7      return $m$;
8  else if $\exists(\perp, r_a, r_b, m, *, \mathcal{O}) \in$ G-List and $\mathsf{Check}(\mathcal{O}, s) = \mathsf{true}$ then
9      replace $(\perp, r_a, r_b, m, *, \mathcal{O})$ with $(s, r_a, r_b, m, *, \mathcal{O})$;
10     return $m$;
11 else
12     $m \xleftarrow{R} \mathcal{S}_{\mathrm{MS}}$;
13     add $(s, r_a, r_b, m, \perp, \perp)$ to G-List;
14     return $m$;
```

It is worth discussing in more detail the subtle differences in how $\mathcal{B}$ answers G queries and how $\mathcal{B}$ uses **GetKey** to compute m values. In particular, notice that **GetKey** queries never add a value of s to an entry on G-List. Instead $\mathcal{B}$ computes m values either using existing entries on G-List (to ensure consistency with earlier **GetKey** and G queries) or by selecting them at random and adding a partially completed entry onto G-List (one without an s value allowing for consistency with later queries). When $\mathcal{B}$ answers G queries, it uses a similar technique; it first ensures consistency with previous **GetKey** and G queries and in addition can add s values to entries. It will not however add *new* pre-master secret oracles to entries on G-List; these are only added to entries during **GetKey** queries, and values of s can be added to existing partially completed entries or new entries with oracle values missing. This ensures that the adversary cannot tell it is in a simulation based on the order of its queries, yet still allows $\mathcal{B}$ to find the value of s used by $\mathcal{A}$ in its G query corresponding to the event E.

If in Algorithm 1 line 1 or line 1 we find two entries in the list which could be replaced, then we abort. This can only happen if the nonce selected by the oracle $\mathcal{O}$ (i.e. r_a or r_b) has collided with another nonce chosen by another oracle. This occurs with probability

$$\frac{n_S(n_P + n'_P) + n_G}{2^t}.$$

We shall hence ignore this possibility in the rest of the proof.

Algorithm $\mathcal{A}$ also makes NewSession, Send, Corrupt, Check and Reveal queries as part of the $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t)$ game; adversary $\mathcal{B}$ answers these queries as follows.

$\mathsf{Send}(\mathcal{O}', \mathrm{msg})$ queries: The way $\mathcal{B}$ answers these queries depends upon whether the underlying pre-master key agreement oracle $\mathcal{O}$ has finished its execution or not; i.e.

upon the value of $\delta_{\mathcal{O}}$. If $\delta_{\mathcal{O}} = \perp$, then $\mathcal{O}$ has not finished its pre-master secret key agreement execution, and so $\mathcal{B}$ simply forwards all Send queries to $\mathcal{O}$ and replies to $\mathcal{A}$ with the responses. If $\delta_{\mathcal{O}} = \textit{rejected}$, then $\mathcal{B}$ responds with $\perp$. In the case of $\delta_{\mathcal{O}} = \textit{accepted-pmk}$ the adversary $\mathcal{B}$ has to simulate the master key agreement message flows. To do this $\mathcal{B}$ computes any $m_{\mathcal{O}'}$ values using the **GetKey** algorithm to simulate any key derivation computations. Then $\mathcal{B}$ uses G to comput the MAC tags using this value of $m_{\mathcal{O}'}$.
Query Corrupt(U): $\mathcal{B}$ issues a Corrupt(U) query to $\mathcal{D}$ to obtain SK_U. Then $\mathcal{B}$ outputs SK_U to $\mathcal{A}$. Note that the corrupt query made by $\mathcal{B}$ ensures that $\gamma_{\mathcal{O}}$ is set to corrupted for each instance i of U.
Query Reveal($\mathcal{O}'$): If $\delta_{\mathcal{O}'} \neq \textit{accepted-mk}$, then $\mathcal{B}$ returns $\perp$. Else if $\delta_{\mathcal{O}'} = \textit{accepted-mk}$, then there will be an entry $(*, r_a, r_b, m, \mathcal{O}', *)$ (resp. $(*, r_a, r_b, m, *, \mathcal{O}')$) on G-List where (r_a, r_b) (resp. (r_b, r_a)) are the random strings on the transcript of $\mathcal{O}'$ that were exchanged. In this case $\mathcal{B}$ responds with m.
Query Check($\mathcal{O}', m$): If $\delta_{\mathcal{O}'} \neq \textit{accepted-mk}$, then $\mathcal{B}$ returns $\perp$. Else if $\delta_{\mathcal{O}'} = \textit{accepted-mk}$, then there will either be an entry $(*, r_a, r_b, m, \mathcal{O}', *)$ (resp. $(*, r_a, r_b, m, *, \mathcal{O}')$) on G-List where (r_a, r_b) (resp. (r_b, r_a)) are the random strings on the transcript of $\mathcal{O}'$ that were exchanged. Then if $m^* = m$, $\mathcal{B}$ responds with true and otherwise with false.

Eventually $\mathcal{A}$ will terminate and output a pair $(\mathcal{O}^{*\prime}, m^*)$. Since the event E has occurred, there will exist an entry $(s^*, r_a^*, r_b^*, m^*, *)$ on G-List where s^* is the pre-master secret of $\mathcal{O}^*$. Furthermore, if $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}(t) = 1$, then $\mathcal{O}^*$ will be a fresh pre-master secret oracle. To find this entry $\mathcal{B}$ scans the G-List and for each entry $(s, r_a, r_b, m, \mathcal{O}, *)$ (or $(s, r_a, r_b, m, *, \mathcal{O})$) with $\mathcal{O} = \mathcal{O}^*$ makes a query Check($\mathcal{O}, s$) to $\mathcal{D}$. If this query returns true, then $\mathcal{B}$ outputs $(\mathcal{O}, s)$ and terminates. We note that since E has occurred, there will exist such an entry, and so $\mathcal{B}$ will always terminate. Hence, if $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}(t) = 1$, then $\mathcal{B}$ will win its game against $\mathcal{D}$, i.e. $\mathsf{Exec}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t) = 1$. From this we get

$$\Pr\left[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \mid E\right] = \mathbf{Adv}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t) + \frac{n_S(n_P + n_P') + n_G}{2^t},$$

as required.

We note that, in the above proof, it may be the case that knowledge of the long-term secret key SK_U of a given user $U \in \mathcal{U}$ may allow the adversary $\mathcal{A}$ to compute any agreed pre-master secret keys. The adversary $\mathcal{B}$ is still able to answer all Send and G queries that $\mathcal{A}$ asks such that $\mathcal{A}$ cannot tell that this is a simulation. To see why this occurs we consider the two main cases below:

- The adversary $\mathcal{A}$ may ask a number of Send queries to have some oracle $\mathcal{O}$ agree upon a master key m with some other oracle using random values r_a and r_b. In this case there will be an entry $(\perp, r_a, r_b, m, \mathcal{O}, *)$ or $(\perp, r_a, r_b, m, *, \mathcal{O})$ on G-List resulting from $\mathcal{B}$ running **GetKey**$(r_a, r_b, \mathcal{O})$. If the adversary then asks a query $G(s, r_a, r_b)$, where s is the correct pre-master secret computed using a Corrupt query, $\mathcal{B}$ will scan G-List for entries of the form $(\perp, r_a, r_b, m, \mathcal{O}, *)$ and $(\perp, r_a, r_b, m, *, \mathcal{O})$ and will use a Check($\mathcal{O}, s$) query to ensure the correct value of m is returned.

- The adversary $\mathcal{A}$ may ask a $G(s, r_a, r_b)$ query first, and hence there will be an entry $(s, r_a, r_b, m, \perp, \perp)$ on G-List. If $\mathcal{A}$ later makes a series of Send queries that results in $\mathcal{O}$ agreeing upon a master secret key using r_a and r_b, then, as part of the **GetKey** algorithm, $\mathcal{B}$ will scan G-List for entries of the form $(s, r_a, r_b, m, \perp, *)$ or $(s, r_a, r_b, m, *, \perp)$ and use a $\mathsf{Check}(\mathcal{O}, s)$ query to ensure the correct value of m is agreed by this oracle. If at any stage the adversary makes a $\mathsf{Corrupt}$ query, the answers that $\mathcal{B}$ gives will always be consistent with the value of s computed by $\mathcal{A}$.

Event E Does Not Occur. In this case we construct the adversary $\mathcal{C}$ against the one-way security of the MAC in a similar way to the adversary $\mathcal{B}$ above.

At the start of the $\mathsf{KR\text{-}CMA}$ game against Mac the algorithm $\mathcal{C}$ is given a security parameter t and access to a tag generation oracle $\mathcal{O}^K_{\mathrm{MAC}}$ and a tag verification oracle $\mathcal{O}^K_{\mathrm{ver}}$ for Mac with an unknown key K. Algorithm $\mathcal{C}$ then acts as a challenger in an $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t)$ game against $\mathcal{A}$. To do this $\mathcal{C}$ sets up the game exactly as the adversary $\mathcal{B}$ in Part 1 but with the following changes. The algorithm $\mathcal{C}$ selects some master secret oracle $\mathcal{O}'_U$ for $U \in \mathcal{U} \cup \mathcal{U}'$ which it "hopes" the adversary $\mathcal{A}$ will output as part of $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t)$. We let $\mathcal{O}'_V$ denote the session oracle that $\mathcal{O}'_U$ agrees a master secret key with for $V \in \mathcal{U} \cup \mathcal{U}'$. To complete the setup of $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t)$ the algorithm $\mathcal{C}$ then calls $\mathcal{A}$ using the setup data.

Algorithm $\mathcal{A}$ will then start to make $\mathsf{NewSession}$, Send, $\mathsf{Corrupt}$, Check, Reveal and G queries as part of $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t)$ which $\mathcal{C}$ answers by simulating the real game except for the following Send queries:

- If $\mathrm{role}_{\mathcal{O}} = \mathit{initiator}$, $\mathrm{msg} = r_b$, and $\mathcal{O}'$ is one of either $\mathcal{O}'_U$ or $\mathcal{O}'_V$, then $\mathcal{C}$ makes a query to the tag generation oracle to obtain a MAC tag σ_a for $0 \,||\, \tau'$ and returns σ_a.
- If $\mathrm{role}_{\mathcal{O}} = \mathit{responder}$, $\mathrm{msg} = r_a$, and $\mathcal{O}'$ is one of either $\mathcal{O}'_U$ or $\mathcal{O}'_V$, then $\mathcal{B}$ assigns $r_b \overset{R}{\leftarrow} \{0,1\}^t$, makes a query to the tag generation oracle to obtain a MAC tag σ_b for $1 \,||\, \tau'$ and returns r_b.
- If $\mathrm{role}_{\mathcal{O}} = \mathit{initiator}$, $\mathrm{msg} = \sigma_b$ and $\mathcal{O}'$ is one of either $\mathcal{O}'_U$ or $\mathcal{O}'_V$, then the response is made using the MAC verification oracle available to $\mathcal{C}$.
- Else if $\mathrm{role}_{\mathcal{O}} = \mathit{responder}$, $\mathrm{msg} = \sigma_a$ and $\mathcal{O}'$ is one of either $\mathcal{O}'_U$ or $\mathcal{O}'_V$, then the response is made using the MAC verification oracle available to $\mathcal{C}$.

Also, algorithm $\mathcal{C}$ answers the $\mathsf{Reveal}(\mathcal{O}')$ queries of $\mathcal{A}$ in the standard way, except if $\mathcal{O}'$ is one of either $\mathcal{O}'_U$ or $\mathcal{O}'_V$, in which case $\mathcal{C}$ aborts.

Now if $\mathcal{C}$ does not abort, then the environment of $\mathcal{A}$ is perfectly simulated, and so $\mathcal{A}$ will eventually terminate and output a pair $(\mathcal{O}^*, m^*)$. With probability $1/((n_P + n'_P) \cdot n_S)$ algorithm $\mathcal{B}$ will have guessed that the oracle $\mathcal{A}$ will output would be $\mathcal{O}^* = \mathcal{O}'_U$. In this case $\mathcal{A}$ will be trying to guess the value for the unknown key used in Mac. Furthermore, if $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t) = 1$, then $m^* = K$. In this case $\mathcal{C}$ outputs m^* and otherwise outputs $m \xleftarrow{R} \mathcal{S}_{\mathsf{MS}}$. As a result, we have

$$\Pr\big[\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'} = 1 \mid \neg E\big] \le \big((n_P + n'_P) \cdot n_S\big) \cdot \mathbf{Adv}^{\mathsf{KR\text{-}CMA}}_{\mathcal{C},\mathsf{Mac}}(t, q_m, q_v),$$

which proves the lemma.[1] □

Proof of Theorem 5.4 (No-Matching). We first give an informal description of the proof. In order to get an oracle to accept when it has no partner, this means that the adversary must have, at some point, been able to forge a MAC tag under a given master secret m (this has to be the case since there does not exist an oracle that has had a matching conversation, so no other oracle would have produced the MAC signature that the oracle accepted with). The adversary may have done this by either computing the master secret key m or by forging the MAC without computing the key. If the adversary has computed the key m, then it must have done this by first computing the pre-master secret key s, since m is obtained via the random oracle G. Based on these cases, we then construct the corresponding adversaries.

We now formalise the proof. We let $\mathcal{A}$ be an adversary against the OW-MS security of $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{TLS}}(\mathsf{Mac}, G))$. Let $\mathcal{O}^{*\prime}$ denote the oracle which satisfies the No Matching condition in the $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t)$ game. We define E to be the event that $\mathcal{A}$, at some point during $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t)$, makes a query $G(s^*, r_a^*, r_b^*)$ such that $\mathcal{O}^{*\prime}$ has randomness exchanged on its transcript of r_a^* and r_b^*, and the pre-master secret key of $\mathcal{O}^*$ is s^*.

We then have

$$\begin{aligned}\Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t)\big] &= \Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \cap E\big] + \Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \cap \neg E\big] \\ &= \Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \mid E\big] \cdot \Pr[E] \\ &\quad + \Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \mid \neg E\big] \cdot \Pr[\neg E] \\ &< \Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \mid E\big] + \Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \mid \neg E\big].\end{aligned}$$

The theorem then follows from the following lemma which captures the above intuition.

Lemma 5.6. *Let* Π *denote a pre-master key agreement protocol, and let the derived master key agreement protocol be denoted by* $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{TLS}}(\mathsf{Mac}, G))$. *Let the event* E *be as described above. Then if* $\mathcal{A}$ *is an adversary against the* OW-MS *security of* Π', *then*

1. *There is an adversary* $\mathcal{B}$ *against the* OW-PMS *security of* Π *such that*

$$\Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \mid E\big] \le \mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{B},\Pi}(t) + \frac{n_S(n_P + n'_P) + n_G}{2^t}.$$

2. *There is an adversary* $\mathcal{C}$ *against the* UF-CMA *security of* Mac *such that*

$$\Pr\big[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \mid \neg E\big] \le \big((n_P + n'_P) \cdot n_S\big) \cdot \mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{C},\mathsf{Mac}}(t).$$

[1] We note that if the adversary $\mathcal{A}$ outputs the oracle $\mathcal{O}'_V$ and $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t) = 1$, then the adversary $\mathcal{C}$ can again win the security game against KR-CMA. We have not included this case in order to keep the analysis simple.

Event E Occurs. Informally this lemma corresponds to the case where the adversary $\mathcal{A}$ causes the event $\mathsf{No\text{-}Matching}_{\mathcal{A}}(t)$ to occur by first computing the pre-master secret s^* of some fresh oracle $\mathcal{O}^*$ and then, by querying G on s^*, obtains the master secret key m^* of $\mathcal{O}^{*\prime}$ and uses this to produce a forgery for Mac.

We first assume that the event E does occur. Let $\mathcal{D}$ be a challenger in an $\mathsf{Exec}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ security game of Π against $\mathcal{B}$. We then construct the adversary $\mathcal{B}$ against $\mathcal{D}$ as follows. $\mathcal{B}$ acts as a challenger in an $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}(t)$ security game for Π' against the adversary $\mathcal{A}$. Algorithm $\mathcal{B}$ simulates the environment for $\mathcal{A}$ and answers queries of $\mathcal{A}$ in exactly the same way as in Lemma 5.5, Part 1.

Eventually $\mathcal{A}$ will terminate and output an oracle (not necessarily $\mathcal{O}^*$) and some element of $\mathcal{S}_{\mathrm{MS}}$. Since the event E has occurred there will exist some entry $(s^*, r_a^*, r_b^*, m^*, \mathcal{O}^*, *)$ or $(s^*, r_a^*, r_b^*, m^*, *, \mathcal{O}^*)$ on the G-List, such that $\mathsf{Check}(\mathcal{O}^*, s^*) = \mathsf{true}$. Furthermore, since No Matching has occurred on $\mathcal{O}^{*\prime}$ this means $\mathcal{O}^*$ is fresh. To find this entry $\mathcal{B}$ scans G-List and for each entry $(s, r_a, r_b, m, \mathcal{O}, *)$ or $(s, r_a, r_b, m, *, \mathcal{O})$ and queries $\mathsf{Check}(\mathcal{O}, s)$. Algorithm $\mathcal{B}$ then outputs $(\mathcal{O}^*, s^*)$ for the security game $\mathsf{Exec}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ against $\mathcal{D}$. As a result we obtain

$$\Pr\bigl[\mathsf{No\text{-}Matching}_{A}(t) \mid E\bigr] \leq \mathbf{Adv}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t) + \frac{n_S(n_P + n'_P) + n_G}{2^t}.$$

as required.

Event E Does not Occur. Informally, this lemma corresponds to the case where $\mathcal{A}$ causes a No Matching condition to occur on an oracle $\mathcal{O}^{*\prime}$ by forging a MAC tag without computing the underlying key for the Mac scheme in use.

Recall that the event E does not occur. We then construct the adversary $\mathcal{C}$ against the unforgeability security of Mac as follows. At the start of the $\mathsf{UF\text{-}CMA}$ game against Mac the algorithm $\mathcal{C}$ is given a security parameter t, access to a tag generation oracle $\mathcal{O}_{\mathrm{MAC}}^{K}(\cdot)$, and tag verification oracle $\mathcal{O}_{\mathrm{ver}}^{K}(\cdot)$ for Mac with an unknown key K. Algorithm $\mathcal{C}$ then acts as a challenger in an $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}(t)$ game against $\mathcal{A}$. To do this $\mathcal{C}$ sets up the game similarly to how this was done by $\mathcal{B}$ in Lemma 5.5, Part 2.

Algorithm $\mathcal{C}$ selects an oracle at random $\mathcal{O}^{*\prime}$ as a guess for the oracle that the $\mathsf{No\text{-}Matching}_{\mathcal{A}}(t)$ condition will be satisfied for. Algorithm $\mathcal{C}$ then calls $\mathcal{A}$ with the setup data and begins to answer any queries that $\mathcal{A}$ makes.

The adversary $\mathcal{C}$ then continues to simulate the responses to all queries correctly except for those involving $\mathcal{O}^{*\prime}$. In this case any Send queries that involve the computation or verification of MAC tags are answered using the tag generation and verification oracles provided to $\mathcal{C}$. Also if a $\mathsf{Reveal}(\mathcal{O}^{*\prime})$ query is made by $\mathcal{A}$, then $\mathcal{C}$ aborts.

Eventually the adversary $\mathcal{A}$ will terminate and output a pair $(\mathcal{O}, m)$. If the event $\mathsf{No\text{-}Matching}_{\mathcal{A}}(t)$ has occurred on $\mathcal{O}^{*\prime}$, then there will be a tag and message pair $(\mathrm{tag}, \mathrm{msg})$ that is on the transcript of $\mathcal{O}^{*\prime}$ that is a valid forgery for Mac. To see why this is true we consider the case where $\mathcal{O}^{*\prime}$ is an initiator (the case of a responder is similar). Here the oracle $\mathcal{O}^{*\prime}$ will have at some point received a random string r_B as an incoming message. The adversary $\mathcal{C}$ will have then made a query $\mathcal{O}_{\mathrm{MAC}}^{K}(0 \,||\, \tau')$, where τ' is the transcript of $\mathcal{O}^{*\prime}$, received tag in response and responded to $\mathcal{A}$ with this. Next the oracle will have received from $\mathcal{A}$ some other tag' in response, and $\mathcal{C}$ will have made a query $\mathcal{O}^K_{\text{ver}}(1 \,||\, \tau')$ and received an answer true. Since $\text{No-Matching}_{\mathcal{A}}(t)$ has occurred on $\mathcal{O}^{*\prime}$, there will be some user $V \in \mathcal{U}$ such that $\text{pid}_{\mathcal{O}^{*\prime}} = V$. The only oracles that possibly could have a matching conversation with $\mathcal{O}^{*\prime}$ are those that belong to V, since V would have to be on the transcript of any oracle that does have a matching conversation with $\mathcal{O}^{*\prime}$. We conclude that the way in which $\text{No-Matching}_{\mathcal{A}}(t)$ has occurred is that there is no oracle (either belonging to V or not) that has had a matching conversation with $\mathcal{O}^{*\prime}$ (rather than some oracle not belonging to V having a matching conversation with $\mathcal{O}^{*\prime}$). This means that the adversary $\mathcal{A}$ must have produced tag$'$ itself: no other oracle in the game $\text{Exec}^{\text{OW-MS}}_{\mathcal{A},\Pi'}(t)$ has the same transcript as $\mathcal{O}^{*\prime}$, so no other oracle would have produced it. Since the adversary $\mathcal{A}$ has produced this tag (as opposed to it being obtained from a $\mathcal{O}^K_{\text{MAC}}(\cdot)$ query) and it verifies correctly with the message $(1 \,||\, \tau')$, it is a valid forgery of this message.

To find this entry $\mathcal{C}$ scans the transcript of $\mathcal{O}^{*\prime}$. Then if the role of $\mathcal{O}^{*\prime}$ is the initiator, $\mathcal{C}$ outputs $(\text{tag}, 1 \,||\, \tau')$ and otherwise $\mathcal{C}$ outputs $(\text{tag}, 0 \,||\, \tau')$, where τ' is the transcript of $\mathcal{O}^{*\prime}$, as part of the UF-CMA game against Mac. Hence, since the adversary $\mathcal{C}$ correctly guesses that the oracle for which the $\text{No-Matching}_{\mathcal{A}}(t)$ condition will be satisfied is $\mathcal{O}^{*\prime}$ with probability $1/((n_P + n'_P) \cdot n_S)$, we obtain

$$\Pr\left[\text{No-Matching}_{\mathcal{A}}(t) \mid \neg E\right] \le \left(\left(n_P + n'_P\right) \cdot n_S\right) \cdot \mathbf{Adv}^{\text{UF-CMA}}_{\mathcal{C},\text{Mac}}(t).$$

6. Application Key Agreement

In this section we extend the model developed so far to deal with application keys obtained from master-secret keys and the analyse the security of the application keys obtained through the TLS protocol.

As discussed in the introduction, we focus on protocols with a particular structure: first, a master-key is agreed by the parties via some master-key agreement protocol Π, and then this key is used as input to an application key derivation protocol, Σ. The same master-key can be used in multiple executions of the application key protocol which can take place in parallel and concurrently.

We capture this setting by modifying the model for master-key agreement protocols as follows. We consider two types of oracles: MK-oracles which correspond to sessions where the master secret key is derived (i.e. sessions of protocol Π), and AK-oracles, which correspond to sessions of the application key derivation protocol (i.e. sessions of Σ). The AK-oracles are spawned by MK-oracles that have established a master-secret key; spawning is done at the request of the adversary. The internal structure and behaviour of MK-oracles are as defined in the previous section. To describe AK-oracles, we again impose some syntactic restrictions on the protocols (and thus on the oracles). We require that AK-oracle $\mathcal{Q}$ maintain variables $\tau_{\mathcal{Q}}, m_{\mathcal{Q}}, \text{role}_{\mathcal{Q}}, \text{pid}_{\mathcal{Q}}$ with the same roles as before. In addition, a new variable $k_{\mathcal{Q}} \in \mathcal{S}_{\text{A}}$ holds the application key obtained in the session. (Here $\#\mathcal{S}_{\text{A}} \ge 2^t$, where t is the security parameter.) The state variable $\delta_{\mathcal{Q}}$ now assumes values in $\{\perp, \textit{accepted-ak}, \textit{rejected}\}$, with the obvious semantics. Finally, the corruption variable $\gamma_{\mathcal{Q}}$ is either $\perp$ or *compromised* (we explain below when the latter value is set).

In addition to the powers previously granted to the adversary, now the adversary can also create new AK-oracles by issuing queries of the form $\text{Spawn}(\mathcal{O})$, with $\mathcal{O}$ an MK-oracle that had successfully finished its execution. As a result, a new oracle $\mathcal{Q} = \Sigma^j_{\mathcal{O}}$ is created (where j indicates that $\mathcal{Q}$ is the jth oracle spawned by $\mathcal{O}$). Oracle $\mathcal{Q}$ inherits the variables $\tau_{\mathcal{Q}}$, $m_{\mathcal{Q}}$, $\mathrm{role}_{\mathcal{Q}}$, and $\mathrm{pid}_{\mathcal{Q}}$ from $\mathcal{O}$ in the obvious way. The adversary may also compromise AK-oracles: when a query $\mathsf{Compromise}(\mathcal{Q})$ is issued, if $\mathcal{Q}$ has accepted, then $k_{\mathcal{Q}}$ is returned to the adversary, and $\delta_{\mathcal{Q}}$ is set to *compromised.* Notice that the $\mathsf{Compromise}$ queries are the analogue of Reveal queries for AK-oracles. We chose to have different names for clarity.

The security of keys is captured via a Test query. When $\mathsf{Test}(\mathcal{Q})$ is issued, a bit $b \in \{0, 1\}$ is chosen at random. Then if $b = 0$, then $k_{\mathcal{Q}}$ is returned to the adversary, otherwise a randomly selected element from $\mathcal{S}_{\mathrm{A}}$ is returned to the adversary (who then has to guess b; see the game defined below).

An AK-oracle $\mathcal{Q}$ is a valid target for the adversary if the parent oracle of $\mathcal{Q}$ is fresh, $\mathcal{Q}$ has finished successfully its execution, its intended partner, say V, is not corrupt, and any session of V with which $\mathcal{Q}$ has a matching conversation is not compromised.

Definition 6.1 (Fresh Application Key Oracle). Let $\mathcal{O}$ be a master key agreement oracle, and $\mathcal{Q}$ denote one of its children. The oracle $\mathcal{Q}$ is said to be fresh if the following conditions hold:

1. $\mathcal{O}$ is a fresh master key agreement oracle.
2. $\gamma_{\mathcal{Q}} = \perp$.
3. $\delta_O =$ *accepted-ak*.
4. $\exists V \in \mathcal{U}$ such that $\mathrm{pid}_{\mathcal{Q}} = V$.
5. No compromised session $\Sigma_{\mathcal{Q}'}$ that belongs to V has had a matching conversation with $\mathcal{Q}$.

Note that here, we are implicitly assuming that knowing a master key automatically gives the adversary all derived application keys. Whilst this will not be true of all protocols one can think of, it is true for all application key derivation protocols that we consider here and in particular in Stage 5 of the protocol of Fig. 1.

Security Game for Application-Key Agreement Protocols. We define the security of an application-key protocol $\Pi; \Sigma$ via a game $\mathsf{Exec}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},\Pi;\Sigma}(t)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

(1) $\mathcal{C}$ generates public-secret key pairs for each user $U \in \mathcal{U}$ and returns the public keys to $\mathcal{A}$.
(2) $\mathcal{A}$ is allowed to make as many $\mathsf{NewSession}$, Send, Spawn, $\mathsf{Compromise}$, Reveal, Check and $\mathsf{Corrupt}$ queries as it likes throughout the game.
(3) At any point during the game adversary $\mathcal{A}$ makes a single $\mathsf{Test}(\mathcal{Q}^*)$ query.
(4) The adversary outputs a bit b'.

We say that $\mathcal{A}$ wins if $\mathcal{Q}^*$ is fresh at the end of the game and its output bit b' is such that $b = b'$ (where b is the bit internally selected during the Test query). In this case the result of $\mathsf{Exec}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},\Pi;\Sigma}(t)$ is set to 1. Otherwise the output of the experiment is set to 0. We write

$$\mathbf{Adv}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},(\Pi;\Sigma)}(t) = \left|\Pr\left[\mathsf{Exec}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},\Pi;\Sigma}(t) = 1\right] - \frac{1}{2}\right|$$

for the advantage of $\mathcal{A}$ in winning the $\mathsf{Exec}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},\Pi;\Sigma}(t)$ game.

Using this security game, we can now define the security of a application key agreement protocol.

Definition 6.2 (Application Key Agreement Security). An application key agreement protocol is secure if it satisfies the following conditions:

- **Correctness:** In the presence of an adversary which faithfully relays messages, two oracles running the protocol accept holding the same application key and session ID, and the application key is distributed uniformly at random on the application key space.
- **Key secrecy:** An application key agreement protocol $\Pi; \Sigma$ satisfies $\mathsf{IND\text{-}AK}$ key secrecy if for any p.p.t. adversary $\mathcal{A}$, its advantage $\mathbf{Adv}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},\Pi;\Sigma}(t)$ is negligible in t.

Remark 6.1. The model that we develop ensures strong security guarantees for the application keys, in the standard sense of indistinguishability against attackers with powerful corruption capabilities. In this sense our model is close to existing ones but has the added feature that we explicitly consider the setting where more than one application key can be derived from the same master key.

Remark 6.2. Notice that at the application key layer we do not require key confirmation anymore. Indeed, a trivial attack on the standard notion of key confirmation can be mounted against application keys derived using the TLS protocol. However, implicit key confirmation for application keys may still be achieved, depending how the application key is actually used.

The loss of this property is in some sense a result of how we chose to break down the protocol for analysis, since one of our goals was to identify what security properties each of the stages provides. However, if one considers Stages 1–4 as the key agreement protocol, and Stages 5–6 as the application, then one does obtain an explicit notion of key confirmation. Hence, the loss of explicit key confirmation in Stage 5 should not be considered a design flaw in TLS.

We now show that the application-key agreement protocol obtained by combining any secure master-key derivation protocol and the application-key derivation protocol of TLS (Stage 5 of Fig. 1) is secure.

For any master-key agreement protocol Π and hash function H, we write $(\Pi; \mathsf{AK}_{\mathsf{TLS}}(H))$ for the application-key agreement protocol obtained by extending Π with the application-key derivation protocol of TLS. Informally, this means that we derive an application key agreement protocol from a master key agreement protocol using Stage 5 of Fig. 1. We make no assumption as to whether the master key agreement protocol itself is derived from a pre-master key agreement protocol as in Fig. 1. The following theorem says that starting with a master-key agreement protocol secure in the sense of Definition 5.3, the above transformation yields a secure application key protocol.

Theorem 6.3. *Let Π be a secure master-key agreement protocol, and H a random oracle. Then $(\Pi; \mathsf{AK}_{\mathsf{TLS}}(H))$ is a secure application-key agreement protocol.*

Proof. That the protocol is correct in the presence of benign adversaries is clear. We sketch the rest of the proof, as the details are modifications of the previous proofs.

To prove the statement let $\mathcal{A}$ be an $\mathsf{IND\text{-}AK}$ adversary against $\Sigma = (\Pi'; \mathsf{AK}_{\mathsf{TLS}}(H))$. In the proof we shall model H as a random oracle. From this will shall construct an $\mathsf{OW\text{-}MS}$ adversary $\mathcal{B}$ against the master secret key agreement protocol Π' in a game $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{B},\Pi'}(t)$ as follows. The algorithm $\mathcal{B}$ acts as a challenger in an $\mathsf{Exec}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},\Sigma}(t)$ security game against $\mathcal{A}$. The algorithm $\mathcal{B}$ simulates H by maintaining a list, the H-list, of queries and responses to the oracle H. The input to adversary $\mathcal{B}$ as part of the $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{B},\Pi'}(t)$ game is used to create the input to adversary $\mathcal{A}$.

At the start of the game the adversary $\mathcal{B}$ selects an oracle $\mathcal{O}^{*\prime}$ as in our other proofs, which it "hopes" the Test query will involve a child of $\mathcal{O}^{*\prime}$. The algorithm $\mathcal{B}$ answers $\mathcal{A}$'s Check, Reveal and $\mathsf{Corrupt}$ queries by passing the queries directly to the challenger of $\mathcal{B}$ and relaying the response back to $\mathcal{A}$. The Spawn and Send queries are handled by $\mathcal{B}$ in the obvious manner.

The $\mathsf{Compromise}$ queries, in the case where the query is made of an oracle which is not a child of $\mathcal{O}^{*\prime}$, are handled by $\mathcal{B}$ making an appropriate Reveal query and then using the random oracle H to produce the required answer. In the case where the query is for a child of $\mathcal{O}^{*\prime}$, in which case we are not allowed to use the Reveal query, the adversary $\mathcal{B}$ simulates the output using the H-List and the Check oracle.

At some point $\mathcal{A}$ will make a Test query of some oracle $\Sigma_{\mathcal{O}}$. If $\mathcal{O} \neq \mathcal{O}^{*\prime}$, then algorithm $\mathcal{B}$ aborts, otherwise algorithm $\mathcal{B}$ returns a random key from the space $\mathcal{S}_{\mathrm{A}}$ to the adversary $\mathcal{A}$.

Eventually $\mathcal{A}$ will terminate with its guess for the bit b. If $\mathsf{Exec}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},\Sigma}(t) = 1$, then, since H is modelled as a random oracle, $\mathcal{A}$ must have queried the oracle H with the inputs corresponding to the underlying master secret key, and message flows, of the application key oracle $\Sigma_{\mathcal{O}^{*\prime}}$. In addition the underlying master key oracle $\mathcal{O}^{*\prime}$ must be fresh in the security game $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{B},\Pi'}(t)$.

Algorithm $\mathcal{B}$ then scans the H-list and checks whether the first component of the input corresponds to the underlying master secret key of $\mathcal{O}^{*\prime}$. It does this by calling Check for the oracle $\mathcal{O}^{*\prime}$. When it finds the correct key, it outputs $(\mathcal{O}^{*\prime}, m_{\mathcal{O}^{*\prime}})$ and terminates.

The theorem then follows. □

Acknowledgements

The authors would like to thank Caroline Belrose for various discussions on key agreement protocols during the writing of this paper and Martin Abadi for interesting insights into various aspects of TLS. The work described in this paper has been supported in part by the EU FP6 project eCrypt and an EPSRC grant, and the second author was supported by a Royal Society Wolfson merit award.

References

[1] M. Abdalla, O. Chevassut, D. Pointcheval, One-time verifier-based encrypted key exchange, in *Public Key Cryptography—PKC 2005*. LNCS, vol. 386 (Springer, Berlin, 2005), pp. 47–64

[2] J.H. An, Y. Dodis, T. Rabin, On the security of joint signature and encryption, in *Advances in Cryptology—EUROCRYPT 2002*. LNCS, vol. 2332 (Springer, Berlin, 2002), pp. 83–107

[3] M. Bellare, R. Canetti, H. Krawczyk, A modular approach to the design and analysis of authentication and key exchange protocols, in *30th Symposium on Theory of Computing—STOC 1998* (ACM, New York, 1998), pp. 419–428

[4] M. Bellare, C. Namprempre, Authenticated encryption: Relations among notions and analysis of the generic composition paradigm, in *Advances in Cryptology—ASIACRYPT 2000*. LNCS, vol. 1976 (Springer, Berlin, 2000), pp. 531–545

[5] M. Bellare, D. Pointcheval, P. Rogaway, Authenticated key exchange secure against dictionary attacks, in *Advances in Cryptology—EUROCRYPT 2000*. LNCS, vol. 1807 (Springer, Berlin, 2000), pp. 139–155

[6] M. Bellare, P. Rogaway, Entity authentication and key distribution, in *Advances in Cryptology—CRYPTO '93*. LNCS, vol. 773 (Springer, Berlin, 1994), pp. 232–249

[7] M. Bellare, P. Rogaway, Optimal asymmetric encryption, in *Advances in Cryptology—EUROCRYPT 1994* (1994), pp. 92–111

[8] M. Bellare, P. Rogaway, Provably secure session key distribution: The three party case, in *27th Symposium on Theory of Computing—STOC 1995* (ACM, New York, 1995), pp. 57–66

[9] K. Bhargavan, R. Corin, C. Fournet, E. Zalinescu, Cryptographically verified implementations for TLS, in *Conference on Computer and Communication Security—CCS 2008* (ACM, New York, 2008), pp. 459–468

[10] R. Bird, I.S. Gopal, A. Herzberg, P.A. Janson, S. Kutten, R. Molva, M. Yung, Systematic design of two-party authentication protocols, in *Advances in Cryptology—CRYPTO '91*. LNCS, vol. 576 (Springer, Berlin, 1991), pp. 44–61

[11] S. Blake-Wilson, D. Johnson, A.J. Menezes, Key agreement protocols and their security analysis, in *Cryptography and Coding*. LNCS, vol. 1355 (Springer, Berlin, 1997), pp. 30–45

[12] S. Blake-Wilson, A.J. Menezes, Entity authentication and authenticated key transport protocols employing asymmetric techniques, in *IWSP*. LNCS, vol. 1361 (Springer, Berlin, 1998), pp. 137–158

[13] D. Bleichenbacher, Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1, in *Advances in Cryptology—CRYPTO '98*. LNCS, vol. 1462 (Springer, Berlin, 1998), pp. 1–12

[14] E. Bresson, O. Chevassut, D. Pointcheval, Provably authenticated group Diffie–Hellman key exchange—The dynamic case, in *Advances in Cryptology—ASIACRYPT 2001*. LNCS, vol. 2248 (Springer, Berlin, 2001), pp. 290–309

[15] R. Canetti, H. Krawczyk, Analysis of key-exchange protocols and their use for building secure channels, in *Advances in Cryptology—EUROCRYPT 2001*. LNCS, vol. 2045 (Springer, Berlin, 2001), pp. 453–474

[16] R. Canetti, H. Krawczyk, Universally composable notions of key exchange and secure channels, in *Advances in Cryptology—EUROCRYPT 2002*. LNCS, vol. 2332 (Springer, Berlin, 2002), pp. 337–351

[17] R. Canetti, H. Krawczyk, Security analysis of IKE's signature-based key-exchange protocol, in *Advances in Cryptology—CRYPTO 2002*. LNCS, vol. 2442 (Springer, Berlin, 2002), pp. 143–161

[18] K.-K.R. Choo, C. Boyd, Y. Hitchcock, Examining indistinguishability-based proof models for key establishment protocols, in *Advances in Cryptology—ASIACRYPT 2005*. LNCS, vol. 3788 (Springer, Berlin, 2005), pp. 585–604

[19] R. Cramer, V. Shoup, Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.* **33**, 167–226 (2003)

[20] T. Dierks, C. Allen, *The TLS Protocol Version 1.0*. RFC 2246, January 1999

[21] T. Dierks, C. Allen, *The TLS Protocol Version 1.2*. RFC 4346, April 2006

[22] W. Diffie, P.C. van Oorschot, M.J. Weiner, Authentication and authenticated key exchange. *Des. Codes Cryptogr.* **2**, 107–125 (1992)

[23] A.O. Freier, P. Karlton, P.C. Kocher, *The SSL Protocol Version 3.0*. Internet Draft, 1996

[24] P.-A. Fouque, D. Pointcheval, S. Zimmer, HMAC is a randomness extractor and applications to TLS, in *AsiaCCS 2008* (ACM Press, New York, 2008), pp. 21–32

[25] S. Gajek, M. Manulis, O. Pereira, A. Sadeghi, J. Schwenk, Universally composable security analysis of TLS, in *Provable Security—ProvSec 2008*. LNCS, vol. 5324 (Springer, Berlin, 2008), pp. 313–327

[26] H. Krawczyk, SKEME: a versatile secure key exchange mechanism for Internet, in *Proceedings of the 1996 Symposium of Network and Distributed System Security (SNDSS'96)* (IEEE Computer Society, Los Alamitos, 1996), p. 114

[27] A. Herzberg, I. Yoffe, The layered games framework for specifications and analysis of security, in *LNCS*, vol. 4948 (Springer, Berlin, 2008), pp. 125–141

[28] K.E.B. Hickman, *The SSL Protocol Version 2.0*. Internet Draft, 1994

[29] J. Jonsson, B. Kaliski Jr., On the security of RSA encryption in TLS, in *Advances in Cryptology—CRYPTO 2002*. LNCS, vol. 2442 (Springer, Berlin, 2002), pp. 127–142

[30] H. Krawczyk, The order of encryption and authentication for protecting communications (or: How secure is SSL?), in *Advances in Cryptology—CRYPTO 2001*. LNCS, vol. 2139 (Springer, Berlin, 2001), pp. 310–331

[31] C. Kudla, Special signature schemes and key agreement protocols. PhD Thesis, Royal Holloway University of London, 2006

[32] C. Kudla, K. Paterson, Modular security proofs for key agreement protocols, in *Advances in Cryptology—ASIACRYPT 2005*. LNCS, vol. 3788 (Springer, Berlin, 2005), pp. 549–565

[33] J.C. Mitchell, V. Shmatikov, U. Stern, Finite-state analysis of SSL 3.0, in *USENIX Security Symposium—SSYM 1998*, 1998

[34] L. Paulson, Inductive analysis of the Internet protocol TLS. *ACM Trans. Inf. Syst. Secur.* **2**(3), 332–351 (1999)

[35] V. Shoup, On formal models for secure key exchange (version 4). Preprint, 1999

[36] D. Wagner, B. Schneier, Analysis of the SSL 3.0 protocol, in *2nd USENIX Workshop on Electronic Commerce*, 1996

[37] S. Williams, The security of signcryption as a key agreement protocol. BSc Dissertation, University of Bristol, 2008