A Modular Security Analysis of the TLS Handshake Protocol

P. Morrissey, N.P. Smart and B. Warinschi

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*.

Our first contribution consists of formal models that clarify the security level enjoyed by each of these types of keys. The models that we provide fall under well established paradigms in defining execution, and security notions. 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 the application keys established through the TLS protocol. We show that the transformation used by TLS to derive master keys essentially transforms an *arbitrary* secure pre-master key agreement protocol into a secure master-key agreement protocol. Similarly, the transformation used to derive application keys works when applied to an arbitrary secure master-key agreement protocol. These results are in the random oracle model. The security of the overall protocol then follows from proofs of security for the basic pre-master key generation protocols employed by TLS.

1 Introduction

The SSL key agreement protocol, developed by Netscape, was made publicly available in 1994 [20] and after various improvements [18] has formed the bases for the TLS protocol [16, 17] which is nowadays ubiquitously present in secure communications over the internet. Surprisingly, despite its practical importance, this protocol had never been analyzed 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 (Figure 1). The protocol 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 (it is assumed that at least one party (the server) possess a long term public/private key pair $(\mathrm{PK}_B, \mathrm{SK}_B)$, as well as a certificate $\mathrm{sig}_{\mathrm{CA}}(\mathrm{PK}_B)$ issued by some certification authority CA). The next three phases, which are 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 analyze 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 obtained master secret key is then confirmed by using it to compute two MACs on the transcript of the conversation which are then exchanged.

(5) In the next phase the master key m is used to obtain one or more applications keys: for each application key, the parties exchange random nonces n_A and n_B and compute the shared

application key via $k = k' \mid\mid k'' \leftarrow H(m, n_A, n_B)$. 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.

(6) Finally the application keys are used in an application (and we exhibit one possible use for encrypting some arbitrary messages). We emphasise that many applications can use the same master key by repeated application of Steps 5 and 6.

The proper use of keys in this last stage had been the object of previous studies [4, 23] and is not part of our analysis.

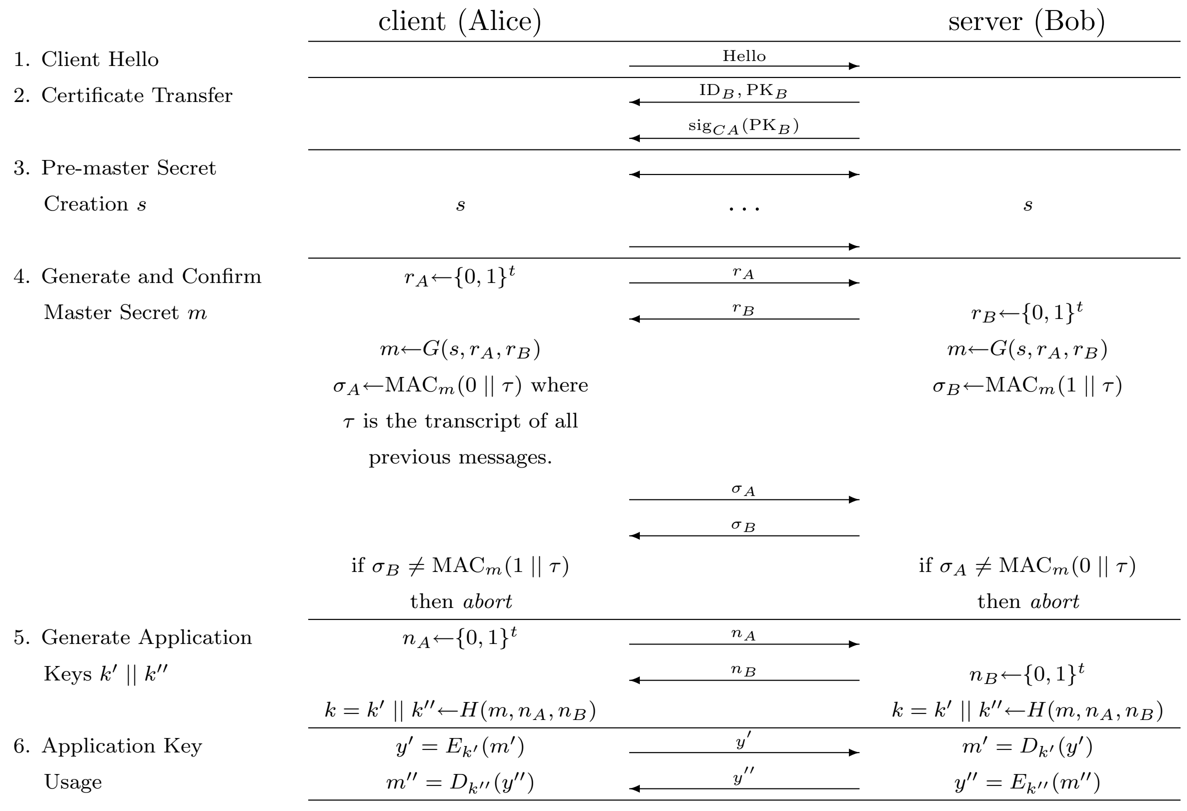

client (Alice)
server (Bob)
1. Client Hello
Hello
2. Certificate Transfer
$\mathrm{ID}_B, \mathrm{PK}_B$
$\mathrm{sig}_{CA}(\mathrm{PK}_B)$
3. Pre-master Secret Creation s
s
...
s
4. Generate and Confirm Master Secret m
$r_A \leftarrow \{0,1\}^t$
r_A
r_B
$r_B \leftarrow \{0,1\}^t$
$m \leftarrow G(s,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

Figure 1: A general TLS like protocol

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 steps in (3) with those in (4) and (5) which leads (as we show in this paper) to secure key agreement protocol 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.

Our Contribution

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 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 path we take is also motivated by the lack of models that capture precisely the security of these keys.

A second important 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 of the 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 to the analysis of its components, thus making the task of proving security more manageable.

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 [9]. In particular we only require that pre-master key agreement protocols are secure in the one-way sense (the adversary cannot recover the entire established key), and that the protocol is secure against man-in-the-middle attacks. 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.

Next, we give a security model for master-key agreement protocols which strengthens the one described above. We still only require secrecy for keys in the one-wayness sense, but now we ask for the protocol to also be secure against unknown-key-share attacks. 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 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. Our model explicitly takes into consideration 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. In particular, we analyze 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. 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.

Our proof is modular and generic. Specifically, we show that the protocol $(\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ 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, when the hash function in the construction is modeled as a random oracle. Similarly, we show that starting from an arbitrary secure master-key agreement protocol Π, the protocol $(\Pi; \mathsf{AK}_{\mathsf{SSL}}(H))$ obtained by appending the flows in phase (5) of TLS is a secure application-key agreement protocol (provided that H is modeled 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 analyze TLS in its entirety for each distinct method for establishing pre-master keys.

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, the RSA key transport mode of TLS uses a randomized padding mechanism to avoid known problems with vanilla

RSA. The original choice was the encryption scheme from PKCS-v1.0. The exact choice is historic, but in modern terms was made to attempt to create an IND-CCA encryption scheme. It turns out that the encryption scheme from PKCS-v1.0 is not in fact IND-CCA secure. This was exploited in the famous reaction attack by Bleichenbacher [11] on SSL, where invalid ciphertext messages were used to obtain pre-master secret keys. Our analysis implies that no randomized padding mechanism is actually needed, as deterministic encryption suffices to guarantee the security of the whole protocol.

Importantly, our models *do* capture security against reaction attacks as long as the full behaviour of the protocol is specified and analyzed. The key aspect is that the analysis should include the behaviour of the parties when the messages that they receive do not follow the protocol (e.g. are malformed). Our analysis of the premaster key agreement based on encryption schemes (e.g. that based on RSA) considers and thus justifies the validity of the patch proposed to cope with reaction attacks (to ensure that the execution when malformed packages are received is indistinguishable from honest executions).

Our models can be used to explicitly capture one-way and mutual authentication via public-key certificate information. We do not model variants of the standard TLS protocol which can include password-based authentication or shared key-based techniques. We leave these extensions for future work.

It is important to observe that 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 modeled and analyzed 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 idealized randomness extractor. As such the typical disclaimer associated to proofs in the random oracle model certainly applies, and we caution against over optimism in their interpretation. A natural and important question is whether a standard model analysis is possible, ideally, assuming that the key derivation function is pseudorandom (as is the function based on HMAC used in the current specification of TLS). Unfortunately, indirect evidence indicates that such a result is extremely hard to obtain. As observed by Jonsson and Kaliski in their analysis of the use RSA in TLS [21], 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 [14]. 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.

Related Work

The work which is closest with ours is the analysis of the use of RSA in TLS by Jonsson and Kaliski [21]. They consider a very simplified security model for the master secret key, for the particular case when the protocol for premaster key is based on encryption. We share the modeling of the key derivation function as a random oracle, and the observation that deterministic encryption may suffice for a secure premaster key had also been made there. However, the present work uses a far more general and modular model for key-exchange, analyzes several pre-master key agreement protocols, including one based on DDH which is offered by TLS.

Other analysis of the TLS protocol used Dolev-Yao models, where security of primitives is postulated, and thus no immediate guarantees are implied for the more concrete world. Such analyses include the one carried out by Mitchel, Shmatikov, and Stern [25] using a model checker, and the one of Paulson who used the inductive method [27]. Wagner and Schneier analyze various security aspects of SSL 3.0 [29], but their treatment is informal. Finally, Bellare and Namprempre [4], and Krawczyk [23] study how to correctly use the application keys derived via TLS. Their treatment is focused exclusively on the use of keys, and not with the security of the entire key agreement protocol.

The first complexity theoretic model for key agreement was the Bellare-Rogaway (BR) model [6, 7]. The main driving forces of this model were the works of [8, 15]. In [6] entity authentication and authenticated key distribution are considered in the two-party symmetric key case where users are modeled 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 [8] and of "matching protocol runs" from [15] are made formal using the notion of a *matching conversation*. Various security attributes are then included in the definition of security by allowing the adversary to make corresponding queries such as Reveal queries. In [7] this was developed to model the three party symmetric key case for entity authentication and key distribution.

Since the BR models there have been a number of other models proposed [9, 10, 3, 13, 5, 12, 1, 24, 28] for various applications and environments. These models can be loosely categorised into two main groups: those that use modular and simulation based techniques [3, 13, 28], and those closer to the original BR model that use an indistinguishability based approach [9, 10, 24]. The TLS protocol could certainly benefit from an analysis in the simulation-based model, e.g. the one of Canetti and Krawczyk [13]. However, under that model the DDH-based instantiation of TLS would be deemed insecure (as the model allows revealing of ephemeral information). Hence we feel that using the older BR-style model is more appropriate to TLS.

The models most relevant to our work are the Blake–Wilson, Johnson and Menezes (BJM) based models [9, 10, 24]. The BJM model of [9] applied the BR model, with some modifications, to authenticated key agreement (AK) and authenticated key agreement with key confirmation (AKC) in the public key case. The work of [9] uses the notion of a No-Matching, first introduced in [6], condition to define a greater separation between AK and AKC protocols and deals with Diffie–Hellman (DH) like protocols. Following on from this [10] deals with the case of key transport using public key encryption (PKE) and key agreement using DH key agreement with digital signatures (DSS). In [24] 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 occurs also in the generic transform proposed by Kudla and Paterson [22, 24] and our techniques are very similar to theirs.

Paper Overview

The structure of the paper is as follows. In Section E we set the scene by giving some basic examples of pre-master key agreement protocols. In Section 2 we recap on the execution model for key agreement protocols. Then in Section 3 we present a security model for such pre-master key agreement protocols and then show that the examples previously given do indeed meet our security definition. In Section 4 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. We then pass on, in Section 5, to the security definition for the derived application keys. Again we show that the TLS transform does indeed produce a secure application key. Finally, in Section I we discuss the secure composition of the key agreement with a confidential application.

In the appendices we present our notational conventions (Appendix A), the hard problems on which we base the security of our protocols (Appendix B), the security notions of various cryptographic primitives we will require (Appendix C), a recap on Bellare and Rogaway's definition of matching conversations (Appendix D), and our security proofs (Appendix E, F, G, and H).

1.1 Acknowledgements

The authors would like to thank Caroline Belrose for various discussions on key agreement protocols during the writing of this paper. The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT and by the EPSRC via the grant GR/T27266. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

2 A Generic Execution model for Two-Party Protocols

The security models that we use in this paper are based on the earlier work of Bellare et. al. [3, 5, 6, 7], as refined by BJM [9]. In this section we give a general description of the common features of these models, and recall some of the intuition behind them. Later, we specialise the general model for the different tasks that we consider in the paper.

Registered and unregistered users. We model a setting with two kinds of users: registered users (with identities in some set $\mathcal{U}$) and non-registered user (with identities in some set $\mathcal{U}'$). Each user in $U \in \mathcal{U}$ 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. There is also a set of participant identities $\mathcal{U}'$, intended to model the users, for which each participant does 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, which he sends back on 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 consider 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) session 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 $\text{role}_{\mathcal{O}} \in \{\textit{initiator}, \textit{responder}, \perp\}$ records the type of session to which the oracle corresponds. Variable $\text{pid}_{\mathcal{O}} \in \mathcal{U}$ 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 $\gamma_{\mathcal{O}} \in \{\perp, \textit{corrupted}\}$ records whether or not the session had been corrupt 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 $\mathsf{NewSession}(U, role)$, with $role \in \{initiator, responder\}$. User U can be either registered or unregistered. We write Π_U^i for the i'th session of user i. 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 answers 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 also 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 intrudoced 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]. We recall their definition in Appendix D.

3 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 analyze the security of the pre-master key agreement protocols used in TLS.

Security Model

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 Section 2 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 $s \in \mathcal{S}_{\mathrm{PMS}} \cup \{\perp\}$ 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: $\perp$ (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 $s_{\mathcal{O}} = \perp$.

The corruption capabilities of the adversary discussed above are modeled using queries $\mathsf{Corrupt}$ and Check formally defined as follows. When the adversary issues a query $\mathsf{Corrupt}(U)$ 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 adversaries may take place. Additionally, no further queries $\mathsf{NewSession}(U, role)$ are permitted.

When the adversary issues the query $\mathsf{Check}(\mathcal{O}, s)$, for $\mathcal{O} = \Pi_U^i$, $i \in \mathbb{N}$, U some uncorrupted party, and $s \in \mathcal{S}_{\text{PMS}}$, then the answer returned to the adversary is **true** if $\delta_{\mathcal{O}} = \textit{accepted-pmk}$ and $s_{\mathcal{O}} = s$ and **false** otherwise. When a given oracle is initialized 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 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 3.1 [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) $\delta_{\mathcal{O}} = \textit{accepted-pmk}$, and (3) $\exists\ V \in \mathcal{U}$ such that V is uncorrupted and $\text{pid}_{\mathcal{O}} = V$.

Security game for pre-master key agreement protocols. We define the security of a pre-master key agreement protocol Π via the following game $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ 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 $s^* \in \mathcal{S}_{\text{PMS}}$.

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 $\mathsf{Exec}_{\Pi,\mathcal{A}}^{\mathsf{OW\text{-}PMS}}(t)$ is set to 1. Otherwise the output of the experiment is set to 0. We write

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

for the advantage of $\mathcal{A}$ in winning the $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ 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 3.2 [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 $\mathcal{S}_{\text{PMS}}$.
- **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 $\mathbf{Adv}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ 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 1. Our security requirements for pre-master secret key agreement are significantly weaker than the standard requirements for key exchange [6, 7]. 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 2. As a consequence of our security requirements our model may deem secure protocols that succumb to unknown-key-share attacks [15]. 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 3. Security under our notion does however guarantee security against man-in-the-middle attacks: if honest parties U and V believe they interact to each other but their pre-master key(s) is in fact shared with the adversary then this is considered security break in our model.

Remark 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 – Figure 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 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 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 modeled as a RO). The same holds true for the rest of the models that we develop in this paper.

Security of the TLS Pre-Master Key Agreement Protocols

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

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. We write $\mathsf{PMK}(\mathsf{Enc})$ for the resulting protocol.

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 of the encryption function of Enc. This results, perhaps surprisingly, in our analysis implying that one can avoid the use of full-fledged $\mathsf{IND\text{-}CCA}$ encryption schemes in favor of the much simpler *deterministic* $\mathsf{OW\text{-}CPA}$ 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 $\mathsf{OW\text{-}CCA}$ security. We formalize the results sketched above in Appendix E.1.

Finally, 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.

Signed Diffie-Hellman pre-master key agreement. The pre-master secret key in TLS can also be produced by exchanging a Diffie-Hellman key g^{xy}, for x and y 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 details are in Appendix E.2). It is known that on its own this protocol does not meet the requirements of an authenticated key agreement protocol, for example see [15] 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 (see Appendix E.2). The results of the following sections will then imply that using signed Diffie-Hellman within the SSL protocol leads to a secure application key exchange protocol.

4 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.

Security Model

The security model is similar to that for pre-master key agreement protocols. We again ask for the adversary not to be 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 a unique session of its intended partner that had 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 $\delta_{\mathcal{O}} =$*accepted-mk* then $m_{\mathcal{O}}$ is returned to $\mathcal{A}$ and $\gamma_{\mathcal{O}}$ is set to *revealed*, while if $\delta_{\mathcal{O}} \neq$*accepted-mk* then the query acts as a no-op. As before when a given oracle is initialized 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 $\delta_{\mathcal{O}} =$*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 a oracle fresh if it is uncorrupted, had successfully finished its execution, its intended partner V is uncorrupted, and none of the revealed oracles that belong to V with which $\mathcal{O}$ 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 $\mathcal{O}$ is not the session with which $\mathcal{O}$ actually interacts.

Definition 4.1 [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) $\delta_{\mathcal{O}} =$ *accepted-mk*, (3) $\exists\ V \in \mathcal{U}$ such that V is uncorrupted and $\mathrm{pid}_{\mathcal{O}} = V$, and

(4) No revealed oracle Π_V^i has had a matching conversation with $\mathcal{O}$.

Security game for master-key agreement protocols. The game $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}MS}}(t)$ 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

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

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.

Besides the secrecy of agreed keys (which we capture via the experiment described above), we are also interested in the issue of key confirmation. 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 4.2 [$\mathsf{No\text{-}Matching}$] 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 $\mathrm{pid}_{\mathcal{O}} = V \in \mathcal{U}$, $\delta_{\mathcal{O}} = \textit{accepted}$, 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 4.3 [Master Key Agreement Security] A 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 master key, and the key 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 $\mathsf{No\text{-}Matching}_{\mathcal{A},\Pi}(t)$ is a negligible function.

Remark 1. Our security requirements for master secret keys are still significantly weaker than the more standard requirements for key exchange [6, 7]. Although the adversarial powers are similar those in existing models (*e.g.*[9]), 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, the master secret key is *not* indistinguishable from a random one since it is used to compute MACs that are sent over the network.

Remark 2. The No Matching property that 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 $\mathcal{O} = \Pi^i_U$ had accepted and has $\Pi_{\mathcal{O}} = V \in \mathcal{U}$, then there exist precisely one session of V with which $\mathcal{O}$ has a matching conversation.

Remark 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 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. 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 SSL 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, and so it does not get written on a transcript.

Security of the TLS master-key derivation protocol

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 $(\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ for the master-key agreement protocol obtained by extending Π with the master-key derivation phase of TLS that is, by appending to the message flows of Π those in Step 4 of Figure 1. We show that starting from a secure pre-master key agreement protocol, the above transformation yields a secure master key agreement protocol.

Theorem 4.4 Let Π be a secure pre-master agreement protocol, Mac be a secure message authentication code, and G a random oracle. Then $(\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ is a secure master-key agreement protocol. ▌

Proof: We need to show that $(\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ is secure in the sense of $\mathsf{OW\text{-}MS}$. We show that if there exists an adversary $\mathcal{A}$ that breaks the security of the construction, then the adversary either breaks the security of the underlying pre-master secret key protocol, or is able to recover the key used in the MAC. More precisely (the details are given in Appendix F) let $\mathcal{A}$ be an adversary against the $\mathsf{OW\text{-}MS}$ security of $(\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ for which 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. Then, there exist adversaries $\mathcal{B}$ against the $\mathsf{OW\text{-}PMS}$ security of Π and $\mathcal{C}$ against the $\mathsf{KR\text{-}CMA}$ security of Mac such that:

$$\mathbf{Adv}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},(\Pi;\mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac},G))}(t) \ \leq \mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{B},\Pi}(t) + (n_S \cdot (n_P + n'_P)) \cdot \mathbf{Adv}^{\mathsf{KR\text{-}CMA}}_{\mathcal{C},\mathsf{Mac}}(t).$$

In addition we need to show that the probability of the event No-Matching is negligible for all adversaries. We show that a successful adversary must either break the security of the underlying pre-master secret key, or be able to forge MACs. More precisely, for any adversary $\mathcal{A}$, there exist adversaries $\mathcal{B}$ against the $\mathsf{OW\text{-}PMS}$ security of Π and $\mathcal{C}$ against the $\mathsf{UF\text{-}CMA}$ of Mac, such that

$$\Pr[\mathsf{No\text{-}Matching}_{\mathcal{A},\Pi}(t)] < \mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{B},\Pi}(t) + (n_S \cdot (n_P + n'_P)) \cdot \mathbf{Adv}^{\mathsf{UF\text{-}CMA}}_{\mathcal{C},\mathsf{Mac}}(t).$$

The details are given in Appendix G. The security of $(\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ in the sense of Definition 4.3 follows from the above results. ▌

5 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 analyze the security of the application keys obtained through the TLS protocol.

Security Model

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 who have established a master-secret key; spawning is done at the request of the adversary. The internal structure and behavior 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 $\tau_{\mathcal{Q}}, m_{\mathcal{Q}}, \mathrm{role}_{\mathcal{Q}}, \mathrm{pid}_{\mathcal{Q}}$ with the same roles as before. In addition, a new variable $k_{\mathcal{Q}} \in \mathcal{S}_{\mathrm{A}}$ holds the application key obtained in the session. (Here $\#\mathcal{S}_{\mathrm{A}} \geq 2^t$, 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 $\delta_{\mathcal{Q}}$ is either $\perp$ or *compromised* (we explain bellow 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 $\mathsf{Spawn}(\mathcal{O})$, 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 j'th 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 $\mathcal{O}$. 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 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, $\mathcal{Q}$ had 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. Formally, we define:

Definition 5.1 [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) $\delta_O = \textit{accepted-ak}$, (4) $\exists\, V \in \mathcal{U}$ such that $\mathrm{pid}_{\mathcal{Q}} = V$, and (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 which 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 Figure 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 NewSession, Send,Spawn, Compromise, Reveal, Check, and Corrupt, queries as it likes throughout the game.

(3) At any point during the game adversary $\mathcal{A}$ makes a single 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 which 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

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

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 5.2 [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 IND-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 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 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. (In Appendix I we discuss the composition of our application key agreement protocol with specific applications, especially confidentiality applications.)

Security of the TLS application-key derivation protocol

In this section we show that the application-key agreement protocol obtained from any secure master-key derivation protocol, and the application-key derivation protocol of TLS (Stage 5 of Figure 1) is secure.

For any master-key agreement protocol Π, and hash function H, we write $(\Pi; \mathsf{AK}_{\mathsf{SSL}}(H))$ 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 Figure 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 Figure 1. The following theorem says that starting with a master-key agreement protocol secure in the sense of Definition 4.3, the above transformation yields a secure application key protocol

Theorem 5.3 Let Π be a secure master-key agreement protocol and H a random oracle. Then $(\Pi; \mathsf{AK}_{\mathsf{SSL}}(H))$ is a secure application-key agreement protocol. ▌

Proof: We prove that for any adversary $\mathcal{A}$ against $\mathsf{IND\text{-}AK}$ security of $(\Pi; \mathsf{AK}_{\mathsf{SSL}}(G))$ there exists an adversary against $\mathsf{OW\text{-}MS}$ security of Π such that

$$\mathbf{Adv}^{\mathsf{IND\text{-}AK}}_{\mathcal{A},(\Pi;\mathsf{AK}_{\mathsf{SSL}}(G))}(t) \leq n_A \cdot (n_P \cdot (n_P + n'_P) \cdot n_S) \cdot \mathbf{Adv}^{\mathsf{OW\text{-}MS}}_{\mathcal{B},\Pi}(t).$$

Where n_P is the number of oracles Π^i_U such that $U \in \mathcal{U}$, n'_P then number of oracles Π^j_V such that $V \in \mathcal{U}'$, n_S is the number of sessions each of these oracles can engage in and n_A is a bound on the number of application key oracles $\Sigma_{\mathcal{O}}$ (i.e. a bound on the number of Spawn queries made by $\mathcal{A}$ for any particular oracle $\mathcal{O}$). See Appendix H for details. ▌

References

[1] M. Abdalla and O. Chevassut and D. Pointcheval. One–Time Verifier–based Encrypted Key Exchange. In *Public Key Cryptography – PKC 2005* Springer-Verlag LNCS 3386, 47–64, 2005.

[2] J.H. An, Y. Dodis and T. Rabin. On the Security of Joint Signature and Encryption. In *Advances in Cryptology – EUROCRYPT 2002*, Springer-Verlag LNCS 2332, 83–107, 2002.

[3] M. Bellare, R. Canetti and 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, 419–428, 1998.

[4] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology – ASIACRYPT 2000*, Springer-Verlag LNCS 1976, 531–545, 2000.

[5] M. Bellare, D. Pointcheval and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – EUROCRYPT 2000*, Springer-Verlag LNCS 1807, 139–155, 2000.

[6] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO '93*, Springer-Verlag LNCS 773, 232–249, 1994.

[7] M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In 27th Symposium on Theory of Computing – STOC 1995, ACM, 57–66, 1995.

[8] R. Bird, I.S. Gopal, A. Herzberg, P.A. Janson, S. Kutten, R. Molva and M. Yung Systematic Design of Two-Party Authentication Protocols. In *Advances in Cryptology – CRYPTO '91* Springer-Verlag LNCS 576, 44–61, 1991.

[9] S. Blake–Wilson, D. Johnson and A.J. Menezes. Key agreement protocols and their security analysis. In *Cryptography and Coding*, Springer-Verlag LNCS 1355, 30–45, 1997.

[10] S. Blake–Wilson and A. Menezes. Entity Authentication and Authenticated Key Transport Protocols Employing Asymmetric Techniques. In *Proceedings of the 5th International Workshop on Security Protocols*, Springer-Verlag LNCS 1361, 137–158, 1998.

[11] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology – CRYPTO '98*, Springer-Verlag LNCS 1462, 1–12, 1998.

[12] E. Bresson, O. Chevassut and D. Pointcheval. Provably Authenticated Group Diffie–Hellman Key Exchange – The Dynamic Case. In *Advances in Cryptology – ASIACRYPT 2001* Springer-Verlag LNCS 2248, 290–309, 2001.

[13] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Advances in Cryptology – EUROCRYPT 2001*, Springer-Verlag LNCS 2045, 453-474, 2001.

[14] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal of Computing*, **33**, 167–226, 2003.

[15] W. Diffie, P.C. van Oorschot and M.J. Weiner. Authentication and authenticated key exchange. *Designs, Codes and Cryptography*, **2**, 107–125, 1992.

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

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

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

[19] P. Fouque, D. Pointcheval and Sébastien Zimmer. HMAC is a Randomness Extractor and Applications to TLS. *Symposium on Information, Computer and Communications Security (ASIACCS'08)*

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

[21] J. Jonsson and B. Kaliski Jr. *On the Security of RSA Encryption in TLS* In *Advances in Cryptology – CRYPTO 2002*, Springer-Verlag, 127-142, 2002.

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

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

[24] C. Kudla and K. Paterson. Modular security proofs for key agreement protocols. In *Advances in Cryptology – ASIACRYPT 2005*, Springer-Verlag LNCS 3788, 549–565, 2005.

[25] John C. Mitchell and Vitaly Shmatikov and Ulrich Stern Finite-state analysis of SSL 3.0. In *SSYM'98: Proceedings of the 7th conference on USENIX Security Symposium, 1998*, 1998.

[26] L. Mazare and B. Warinschi. On the security of encryption under adaptive corruptions. Preprint, 2007.

[27] L. Paulson. Inductive analysis of the Internet protocol TLS. In *ACM Transations on Information and Systems Security*, 2(3):332–351, 1999.

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

[29] D. Wagner and B. Schneier Analysis of the SSL 3.0 protocol. In *"2nd USENIX Workshop on Electronic Commerce",* 1996.

A Notation

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 $\forall\ t > t_c, \epsilon(t) < t^{-c}$.

If S is any set then we denote the action of sampling an element from S uniformly at random and assigning the result to the variable x as $x \overset{R}{\leftarrow} S$. If A is any 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 as $x \leftarrow A^{\mathcal{O}_1(\cdot),\mathcal{O}_2(\cdot)}(y_1, \ldots, y_n)$.

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.

B Hard Problems

The main hard problems used in this work are the Diffie-Hellman related problems.

Definition B.1 [Computational Diffie-Hellman Problem] Given a group $\mathbb{G}$ of prime order q, the computational Diffie-Hellman problem (CDH) in $\mathbb{G}$ is given $g^a, g^b \in \mathbb{G}$, where g is a random generator for $\mathbb{G}$ and $a, b \in \mathbb{Z}_q^\times$ are unknown, to find $g^{ab} \in \mathbb{G}$.

When we say that the CDH problem is hard in some group $\mathbb{G}$ we take this to mean that there does not exist any known polynomial time algorithm for solving the CDH problem in $\mathbb{G}$ with non-negligible probability. Related to the CDH problem is the *decisional Diffie-Hellman problem* (DDH) which is given a triple $(g^a, g^b, g^c) \in \mathbb{G}^3$, where g is some generator for $\mathbb{G}$ and $a, b, c \in \mathbb{Z}_q^\times$ are unknown, to decide if $g^{ab} = g^c$ in $\mathbb{G}$ or not. Related to both these problems is the *gap Diffie-Hellman problem.*

Definition B.2 [The Gap Diffie-Hellman problem] Given a group $\mathbb{G}$ of prime order q, the gap Diffie-Hellman problem ($\mathsf{gap\text{-}DH}$) in $\mathbb{G}$ is to solve the CDH problem in $\mathbb{G}$ given access to an oracle $\mathcal{O}_{\mathsf{DDH}}(g^a, g^b, g^c)$ that solves the DDH problem in $\mathbb{G}$.

If $\mathcal{B}$ is an algorithm which tries to solve the gap-Diffie–Hellman problem then we define its advantage by

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

C Cryptographic Primitives

In this appendix we recap on the various cryptographic constructions that we require, and the security models associated with them that we use.

C.1 Public Key Encryption

Definition C.1 [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: $m \leftarrow \mathcal{D}_{\mathrm{SK}}(c)$.

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. If $\mathcal{B}$ is an adversary against the $\mathsf{OW\text{-}CPA}$ security of a public key encryption scheme $\mathsf{Enc} = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ then we define its advantage by

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

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

If we give the adversary $\mathcal{B}$ 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

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

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

These two security notions are weaker than the standard security notion for public key encryption schemes, namely $\mathsf{IND\text{-}CCA}$. For completeness we present the definition of $\mathsf{IND\text{-}CCA}$ security here. We first consider the following security game $\mathsf{Exec}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B},\mathsf{Enc}}(t)$ between a challenger $\mathcal{C}$ and an adversary $\mathcal{B}$.

(1) The challenger $\mathcal{C}$ sets $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathcal{G}(t)$ then gives PK, t, and a decryption oracle that uses SK to decrypt valid ciphertexts $\mathcal{O}^{\mathrm{SK}}_{\mathcal{D}}(\cdot)$ to $\mathcal{B}$.
(2) The adversary $\mathcal{B}$ makes as many decryption queries on ciphertexts of its choice to $\mathcal{O}^{\mathrm{SK}}_{\mathcal{D}}$ and receives the corresponding plaintexts.
(3) The adversary selects two messages m_0, m_1 such that $|m_0| = |m_1|$ for which it wishes to be challenged and passes these to $\mathcal{C}$.
(4) The challenger $\mathcal{C}$ sets $b \xleftarrow{R} \{0,1\}$ and $c^* \leftarrow \mathcal{E}_{\mathrm{PK}}(m_b; r^*)$ then passes c^* to $\mathcal{B}$.
(5) The adversary again makes as many decryption queries as it likes with the restriction that it cannot make the query $\mathcal{O}^{\mathrm{SK}}_{\mathcal{D}}(c^*)$.
(6) The adversary outputs a bit b^*.

We say $\mathcal{B}$ wins if $b^* = b$. In this case the output of $\mathsf{Exec}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B},\mathsf{Enc}}(t)$ is set to 1 and otherwise set to 0. We then define the advantage of such an adversary $\mathcal{B}$ by

$$\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B},\mathsf{Enc}}(t) = Pr\left[\mathsf{Exec}^{\mathsf{IND\text{-}CCA}}_{\mathcal{B},\mathsf{Enc}}(t) = 1\right].$$

We say that a public key encryption scheme Enc is $\mathsf{IND\text{-}CCA}$ secure if the advantage of any polynomial time adversary is a negligible function in t.

C.2 Digital Signatures

Definition C.2 [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) correspond 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 $\mathrm{ver}_{\mathrm{PK}}(m, \mathrm{sig}_{\mathrm{SK}}(m)) =$ *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 scheme and $\mathcal{C}$ an adversary against this scheme in terms of strong existential forgery under adaptive chosen message attack $\mathsf{SEF\text{-}CMA}$, then we define the advantage of $\mathcal{C}$ by

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

By a strong existential forgery we mean that the message/signature pair (m, s) output by the adversary, is such that the pair (m, s) has not been output by a call to $\mathcal{O}^{\mathrm{SK}}_{\mathrm{sig}}(\cdot)$.

C.3 Message Authentication Codes

Definition C.3 [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 $\mathcal{O}^K_{\mathrm{MAC}}(M)$ which returns a valid tag for the message M. The second oracle is a tag verification oracle $\mathcal{O}^K_{\mathrm{ver}}(M, \mathrm{tag})$ 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 $\mathcal{A}$ an adversary against Mac in terms of recovering the underlying key using a chosen message attack ($\mathsf{KR\text{-}CMA}$) then we define the advantage of $\mathcal{A}$ by

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

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

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

Note that we do not have a notion of strong or weak forgeries as we always assume that the tag generation algorithm for Mac is always a deterministic one.

D Conversations and matching conversations

The definitions of a conversation and matching conversation given here are taken from [6]. Note, unlike [9] we do not use the notion of matching conversation with appendix, despite us working in the public key model.

Definition D.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 it is not the same thing as a complete transcript: it does not contain information such as the decisions reached and session ID's. Also notice that 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 D.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 exists $\mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_R$ 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 exists $\mathsf{t}_0 < \mathsf{t}_1 < \cdots < \mathsf{t}_R$ 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, and the second oracle has conversation C_V^i, and C_V^j matches C_U^i.

E Example Pre-Master Key Agreement Sub-Protocols

In this section we present the two main pre-master key agreement sub-protocols used in the TLS stack as examples of pre-master key agreement protocols and discuss some of the main security issues with them. All the schemes in this section trivially satisfy the correctness requirement for pre-master key agreement protocols, so we do not insist upon this point.

Figure 2: Public Key Encryption Based Pre-Master Key Agreement

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 $s \xleftarrow{R} \mathcal{S}_{\mathrm{PMS}}$
		Else $s \leftarrow \mathcal{D}_{\mathrm{SK}_B}(c)$

E.1 Pre-Master Key Agreement from Arbitrary Encryption Schemes.

Figure 2 describes a key transport mechanism using the encryption algorithm $\mathcal{E}$ of an arbitrary public key encryption scheme Enc. We assume that the pre-master key space $\mathcal{S}_{\mathrm{PMS}}$ is contained in the underlying message space $\mathcal{M}$ of the encryption scheme.

Here, a user selects a random pre-master key $s \in \mathcal{S}_{\mathrm{PMS}}$ and sends to the server the encryption of s under the server's public key. The server obtains s by decrypting the ciphertext that it receives. We write $\mathsf{PMK}(\mathsf{Enc})$ for the resulting pre-master key agreement protocol.

As previously discussed, one would suspect that to obtain a secure key agreement protocol one would need the encryption scheme used to be $\mathsf{IND\text{-}CCA}$ secure. The following theorems state that a simple *deterministic* $\mathsf{OW\text{-}CPA}$ secure scheme or a probabilistic $\mathsf{OW\text{-}CCA}$ secure scheme suffices.

Theorem E.1 If Enc is a $\mathsf{OW\text{-}CPA}$ secure deterministic encryption scheme, then the pre-master secret key agreement protocol $\Pi = \mathsf{PMK}(\mathsf{Enc})$ defined in Section E.1 is secure. ▌

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

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

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 $\mathsf{OW\text{-}PMS}$ security of the pre-master key transport protocol Π of Figure 2, where the encryption scheme is $\mathsf{OW\text{-}CPA}$ secure. The algorithm $\mathcal{B}$ against the $\mathsf{OW\text{-}CPA}$ 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 $\mathsf{OW\text{-}CPA}$ 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 $\mathsf{NewSession}$, Send, $\mathsf{Corrupt}$ and Check queries which $\mathcal{B}$ answers as follows:

- If a $\mathsf{Corrupt}(U)$ query is made of 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 control of all oracles belonging to the corrupted participant U are given to $\mathcal{A}$.

- If the adversary makes a $\mathsf{Send}(\mathcal{O}, \text{msg})$ query of oracle and $\mathcal{O} = \Pi_{U^\dagger}^{i^\dagger}$, 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 $\mathcal{O} = \Pi_U^i \neq \Pi_{U^\dagger}^{i^\dagger}$ 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 $\delta_\mathcal{O} =$ *accpeted–pmk*.
- The $\mathsf{Check}(\mathcal{O}, s)$ queries which $\mathcal{A}$ makes can always be answered by $\mathcal{B}$ since the encryption scheme is deterministic.

If $\mathcal{B}$ does not terminate then eventually $\mathcal{A}$ will terminate and output a pair $(\mathcal{O}^*, s^*) = (\Pi_{U^*}^{i^*}, s^*)$. Since $\mathcal{U}^\dagger \in \mathcal{U} \cup \mathcal{U}'$ then 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 $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$, the adversary $\mathcal{A}$ will be attempting to find the message behind the ciphertext $c^\dagger$ if and only if all 3 of these conditions hold. This happens with probablility $1/(n_P \cdot (n_P + n'_P) \cdot n_S)$. If, in addition to this, $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t) = 1$ 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. ▮

We note that the method of agreeing pre-master secret keys given in Figure 2 and the description of responder oracles in the above proof include the patch used to prevent the attack of Bleichenbacher [11]. In particular, whenever a responder oracle recieves a message that does not decrypt to a valid element of the pre-master secret key space a random element is chosen and the decision value set to *accepted–pmk*. As a result the adversary will not be able to obtain any information as to whether a given ciphertext is the encryption of a correctly formed plaintext for the underlying encryption scheme used. We also note that the value $\text{pid}_\mathcal{O}$ for responder oracles will always remain as $\perp$ since we only consider the case of one-way authentication here.

The next theorem captures that $\mathsf{PMK}(\mathsf{Enc})$ is secure when the encryption algorithm of Enc is a *randomized*. However, in this case we require for Enc to be $\mathsf{OW\text{-}CCA}$.

Theorem E.2 If Enc is a $\mathsf{OW\text{-}CCA}$ secure randomized encryption scheme, then $\mathsf{PMK}(\mathsf{Enc})$ is a secure pre-master key transport protocol. ▮

Proof: 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

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

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.

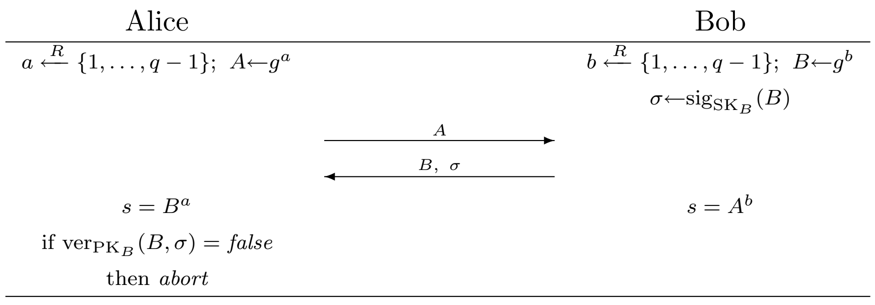
The proof is essentially the same as for the previous theorem. The only difference is that the $\mathsf{Check}(\mathcal{O}, s)$ queries to a given oracle are simulated either by performing a valid decryption using the public/private key pair held by the algorithm B (in the case where the recipient is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

E.2 Pre-Master Key Agreement Via Signed Diffie–Hellman

Let $\mathbb{G} = \langle g \rangle$ be a finite cyclic group of prime order q in which the Diffie–Hellman problem is hard and let $\mathsf{Sig} = (\mathcal{G}, \mathrm{sig}, \mathrm{ver})$ be a public key signature scheme;

We write $\mathsf{PMK}(\mathsf{Sig}, \mathbb{G})$ for the protocol for deriving a pre-master secret via signed Diffie–Hellman given in Figure 3. The set of pre-master keys is set to $\mathbb{G}$.

Figure 3: Signed Diffie–Hellman Based Pre-Master Key Agreement


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
$B,\ \sigma$
$s = B^a$
$s = A^b$
if $\mathrm{ver}_{\mathrm{PK}_B}(B, \sigma) = \mathit{false}$
then *abort*

Notice that we give the protocol for the case of only one party having a public/private key pair. The case for when both have keys is immediate. The following theorem captures the security of this protocol.

Theorem E.3 Let $\mathbb{G}$ be cyclic group for which the gap-Diffie-Hellman assumption holds and let Sig be a secure digital signature scheme. Then $\Pi = \mathsf{PMK}(\mathsf{Sig}, \mathbb{G})$ is a secure pre-master key agreement protocol. ▌

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

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

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 Figure 3. We define E to be the event that at the end of an $\mathsf{Exec}^{\mathsf{OW\text{-}PMS}}_{\mathcal{A},\Pi}(t)$ game 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 oracle in the game. We then note the following

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

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

First assume the event E does not occur. Then we construct the algorithm $\mathcal{B}$ against the $\mathsf{gap\text{-}DH}$ problem as follows. Algorithm $\mathcal{B}$ is given as input a security parameter t, a decisional Diffie-Hellman oracle $\mathcal{O}_{\mathsf{DDH}}$ and an instance of the Diffie-Hellman problem (g, g^a, g^b) where g is a generator of some cyclic group $\mathbb{G}$ of prime order q in which the gap-Diffie-Hellman problem is computationally infeasible and $a, b \in \mathbb{Z}_q^{\times}$ are unknown to $\mathcal{B}$. Next $\mathcal{B}$ acts as a challenger to $\mathcal{A}$ in an $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t)$ 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 $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}$ algorithm $\mathcal{B}$ calls $\mathcal{A}$ using this data.

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

- If a $\mathsf{Corrupt}(U)$ query is made then $\mathcal{B}$ returns SK_U.
- When $\mathcal{A}$ makes a $\mathsf{Send}(\mathcal{O}, \mathrm{msg})$ query to an oracle $\mathcal{O} = \Pi_U^i$ 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 $\mathsf{Check}(\mathcal{O}, s)$ 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 $\mathcal{O}_{\mathsf{DDH}}$. 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 $(\mathcal{O}^*, s^*) = (\Pi_{U^*}^{i^*}, s^*)$.

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 $\mathcal{O}^\dagger = \Pi_{V^\dagger}^{i^\dagger}$. Furthermore, if we have $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t) = 1$ then

$$s^* = (g^{ab})^{r_{\mathcal{O}^*} r_{\mathcal{O}^\dagger}}.$$

The algorithm $\mathcal{B}$ then constructs the solution to the Diffie–Hellman problem as

$$(s^*)^{1/(r_{\mathcal{O}^*} r_{\mathcal{O}^\dagger})}$$

and outputs this. Hence,

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

Next we consider the case in which the event E does occur. In this case we construct the algorithm $\mathcal{C}$ against the $\mathsf{SEF\text{-}CMA}$ 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 $\mathsf{NewSession}$, Send, Check and $\mathsf{Corrupt}$ queries which $\mathcal{C}$ answers in the following way:

- If a $\mathsf{Corrupt}(U)$ query is made and $U \neq U^\dagger$ then $\mathcal{C}$ returns SK_U, else if $U = U^\dagger$ then $\mathcal{C}$ aborts.

- When $\mathcal{A}$ makes a $\mathsf{Send}(\mathcal{O}, \text{msg})$ 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 $x \in \mathbb{Z}_q^\times$ and computes g^x, then calling its signature oracle it obtains the signature on g^x and replies with $(g^x, \mathcal{O}_{\text{sig}}^{\text{SK}}(g^x))$.
- If the adversary makes a $\mathsf{Check}(\mathcal{O}, s)$ query then $\mathcal{C}$ knows the ephemeral Diffie–Hellman secret of the queried oracle and so can compute the associated Diffie–Hellman secret and so is able to answer the Check queries honestly.

Now 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 $\text{pid}_{\mathcal{O}^*} = U^\dagger$ and $\mathsf{Exec}_{\mathcal{A},\Pi}^{\mathsf{OW\text{-}PMS}}(t) = 1$ then there is an entry $(h, \text{sig}_{\text{SK}_{U^\dagger}}(h))$ on the transcript $\tau_{\mathcal{O}^*}$ of $\mathcal{O}^*$ that has correctly verified under $\text{PK}_{U^\dagger}$. 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 $\mathcal{O}_{\text{sig}}^{\text{SK}}$). As a result the pair $(h, \text{sig}_{\text{SK}_{U^\dagger}}(h))$ 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 $\mathsf{SEF\text{-}CMA}$ of the signature scheme. We then find, since the choice of U^* is outside the view of the adversary,

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

And from this the result follows, with the stated advantage statement. ▮

E.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, which also shows security for the mutually authenticated versions of TLS deployed in practice.

F Proof of Theorem 4.4 Part 1

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 formalize the proof. Let $\mathcal{A}$ be an adversary against the $\mathsf{OW\text{-}MS}$ security of the master key agreement protocol $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$. For the remainder of this section if Π_U^i is a pre-master secret oracle then we denote $\Pi_U^{i\,\prime}$ to be the master secret oracle corresponding to Π_U^i.

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 $\Pi_{U^*}^{i^*}{}'$.

We then note the following:

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

The above theorem thus follows from the following two lemmas which capture the above intuition above.

Lemma F.1 Let Π denote a pre-master key agreement protocol and $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ the derived master key agreement protocol. Let the event E be as described above. Then if $\mathcal{A}$ is an adversary against the OW-MS security of Π' then there is an adversary $\mathcal{B}$ against the OW-PMS security of Π such that

$$\Pr[\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}} = 1 \mid E] = \mathbf{Adv}_{\mathcal{B},\Pi}^{\mathsf{OW\text{-}PMS}}(t).$$

Lemma F.2 Let Π denote a pre-master key agreement protocol and $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ the derived master key agreement protocol. Let the event E be as described above. Then if $\mathcal{A}$ is an adversary against the OW-MS security of Π' then there is an adversary $\mathcal{C}$ against the KR-CMA security of the MAC such that

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

Proof of of lemma F.1: First assume the event E does occur. Let $\mathcal{D}$ be a challenger in a 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 $\mathcal{U}$, 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 $(s, r_a, r_b, m, \mathcal{O})$ 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 $\mathcal{O} \in \{\Pi_U^i\} \cup \{\perp\}$. 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 also define the **GetKey** algorithm as Algorithm 1.

Algorithm $\mathcal{A}$ will then make NewSession, Send, Corrupt, Check, Reveal and G queries as part of the $\mathsf{Exec}_{\mathcal{A},\Pi'}^{\mathsf{OW\text{-}MS}}(t)$ game which $\mathcal{B}$ answers as follows.

Send$(\mathcal{O}', \mathrm{msg})$:

- This is identical to the definition of the Send queries in the actual game, except that we use the function **GetKey** to answer the G queries in producing master secret keys $m_{\mathcal{O}'}$.

Corrupt(U):

Algorithm 1: **GetKey** Algorithm.

```
Input: A tuple $(r_a, r_b, \mathcal{O})$
Output: A value $m \in \mathcal{S}_{\mathrm{MS}}$
1 if $\exists\ (s, r_a, r_b, m, \mathcal{O}) \in$ G-List with $\mathsf{Check}(\mathcal{O}, s) =$ true then
2     return $m$;
3 else if $\exists\ (s, r_a, r_b, m, \perp) \in$ G-List with $\mathsf{Check}(\mathcal{O}, s) =$ true then
4     replace $(s, r_a, r_b, m, \perp)$ with $(s, r_a, r_b, m, \mathcal{O})$;
5     return $m$;
6 else
7     $m \overset{R}{\longleftarrow} \mathcal{S}_{\mathrm{MS}}$;
8     add $(\perp, r_a, r_b, m, \mathcal{O})$ to G-List;
9     return $m$;
```

- $\mathcal{B}$ issues a $\mathsf{Corrupt}(U)$ query to $\mathcal{D}$ to obtain SK_U.
- $\mathcal{B}$ outputs SK_U. Note that the query $\mathsf{Corrupt}(U)$ made by $\mathcal{B}$ will ensure that $\gamma_{\mathcal{O}}$ is set to corrupted for each instance i of U.

$G(s, r_a, r_b)$:

- 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 $\mathcal{B}$ returns $\perp$.
- Else if there exists $(s, r_a, r_b, m, \mathcal{O})$ on G-List then $\mathcal{B}$ replies with m.
- Else if there exists an entry $(\perp, r_a, r_b, m, \mathcal{O})$ on G-List and $\mathsf{Check}(\mathcal{O}, s) =$ **true** then $\mathcal{B}$ replaces this entry with $(s, r_a, r_b, m, \mathcal{O})$ and returns m.
- Else $\mathcal{B}$ assigns $m \overset{R}{\longleftarrow} \mathcal{S}_{\mathrm{MS}}$, adds $(s, r_a, r_b, m, \perp)$ to G-List and returns m.

$\mathsf{Reveal}(\mathcal{O}')$:

- If $\delta_{\mathcal{O}'} \neq$ *accepted–mk* then $\mathcal{B}$ returns $\perp$.
- Else if $\delta_{\mathcal{O}'} =$ *accepted–mk* then there will be an entry $(*, r_a, r_b, m, \mathcal{O})$ on G-List where r_a and r_b are the random strings on the transcript of $\mathcal{O}'$ that were exchanged. In this case $\mathcal{B}$ responds with m.

$\mathsf{Check}(\mathcal{O}', m)$:

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

$$\Pr[\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'} = 1 \mid E] = \mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{B},\Pi}(t),$$

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 $(\perp, r_a, r_b, m, \mathcal{O})$ 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 $\mathsf{Corrupt}$ query, $\mathcal{B}$ will scan G-List for entries of the form $(\perp, r_a, r_b, m, \mathcal{O})$ and will use a $\mathsf{Check}(\mathcal{O}, s)$ 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 $(s, r_a, r_b, m, \perp)$ 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 $(s, r_a, r_b, m, \perp)$ 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 $\mathcal{A}$.

Proof of of lemma F.2: First assume the event E does not occur. We then construct the adversary $\mathcal{C}$ against the one-way security of the MAC in a similar way to the the adversary $\mathcal{B}$ in lemma F.1.

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 lemma F.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 $\mathsf{NewSession}, \mathsf{Send}, \mathsf{Corrupt}, \mathsf{Check}, \mathsf{Reveal}$ 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 execpt for the following Send queries:

- If $\mathrm{role}_{\mathcal{O}} = \textit{initiator}$, $\mathrm{msg} = r_b$, 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 $0 \mid\mid \tau'$ and returns σ_a.
- If $\mathrm{role}_{\mathcal{O}} = \textit{responder}$, $\mathrm{msg} = r_a$, and $\mathcal{O}'$ is one of either $\mathcal{O}'_U$ or $\mathcal{O}'_V$, then $\mathcal{B}$ assigns $r_b \overset{R}{\longleftarrow} \{0,1\}^t$, makes a query to the tag generation oracle to obtain a MAC tag σ_b for $1 \mid\mid \tau'$ and returns r_b.
- If $\mathrm{role}_{\mathcal{O}} = \textit{initiator}$, $\mathrm{msg} = \sigma_b$ 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 $\mathrm{role}_{\mathcal{O}} = \textit{responder}$, $\mathrm{msg} = \sigma_a$ 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 probality $1/((n_P + n'_P) \cdot n_S)$ 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 $m \overset{R}{\longleftarrow} \mathcal{S}_{\mathrm{MS}}$. As a result we have

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

Which proves the Lemma[1]. ▮

G Proof of Theorem 4.4 Part 2

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 formalize the proof. For the remainder of this section if Π^i_U is a pre-master secret oracle then we denote ${\Pi^i_U}'$ to be the master secret oracle corresponding to Π^i_U. We let $\mathcal{A}$ be an adversary against the $\mathsf{OW\text{-}MS}$ security of $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$. 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 $G(s^* \mid\mid r^*_a \mid\mid r^*_b)$ 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

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

The theorem then follows from the following two lemmas which capture the above intuition. Due to space reasons, their proofs are provided in the full version of the paper.

Lemma G.1 Let Π denote a pre-master key agreement protocol and $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ the derived master key agreement protocol. Let the event E be as described above. Then if $\mathcal{A}$ is an adversary against the $\mathsf{OW\text{-}MS}$ security of Π' then there is an adversary $\mathcal{B}$ against the $\mathsf{OW\text{-}PMS}$ security of Π such that

$$\Pr[\mathsf{No\text{-}Matching}_{\mathcal{A}}(t) \mid E] \le \mathbf{Adv}^{\mathsf{OW\text{-}PMS}}_{\mathcal{B},\Pi}(t).$$

Lemma G.2 Let Π denote a pre-master key agreement protocol and $\Pi' = (\Pi; \mathsf{MKD}_{\mathsf{SSL}}(\mathsf{Mac}, G))$ the derived master key agreement protocol. Let the event E be as described above. Then if $\mathcal{A}$ is an adversary against the $\mathsf{OW\text{-}MS}$ security of Π' then there is an adversary $\mathcal{C}$ against the $\mathsf{UF\text{-}CMA}$ security of Mac such that

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

[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 $\mathsf{KR\text{-}CMA}$. We have not included this case in order to keep the analysis simple.

Proof of of lemma G.1: Informally this lemma corresponds to the case where the adversary $\mathcal{A}$ causes the event $\textsf{No-Matching}_{\mathcal{A}}(t)$ 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 $\textsf{Mac}$.

We first assume that the event E does occur. Let $\mathcal{D}$ be a challenger in an $\textsf{Exec}^{\textsf{OW-PMS}}_{\mathcal{B},\Pi}(t)$ 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 $\textsf{Exec}^{\textsf{OW-MS}}_{\mathcal{A},\Pi'}(t)$ 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 F.1.

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

$$\Pr[\textsf{No-Matching}_{A}(t) \mid E] \leq \mathbf{Adv}^{\textsf{OW-PMS}}_{\mathcal{B},\Pi}(t),$$

as required. ▌

Proof of of lemma G.2:

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 $\textsf{Mac}$ scheme in use.

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

Algorithm $\mathcal{C}$ selects an oracle at random $\mathcal{O}^{*\prime}$ as a guess for the oracle that the $\textsf{No-Matching}_{\mathcal{A}}(t)$ 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 $\textsf{Send}$ 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 $\textsf{Reveal}(\mathcal{O}^{*\prime})$ 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 $\textsf{No-Matching}_{\mathcal{A}}(t)$ has occured on $\mathcal{O}^{*\prime}$ then there will be a tag and message pair (tag, msg) that is on the transcript of $\mathcal{O}^{*\prime}$ that is a valid forgery for $\textsf{Mac}$. 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 recieved a random string r_B as an incoming message. The adversary $\mathcal{C}$ will have then made a query $\mathcal{O}^K_{\text{MAC}}(0 \mid\mid \tau')$, where τ' is the transcript of $\mathcal{O}^{*\prime}$, recieved tag in response and responded to $\mathcal{A}$ with this. Next the oracle will have recieved from $\mathcal{A}$ some other tag$'$ in response and $\mathcal{C}$ will have made a query $\mathcal{O}^K_{\text{ver}}(1 \mid\mid \tau')$ and recieved an answer **true**. Since $\textsf{No-Matching}_{\mathcal{A}}(t)$ has occured on $\mathcal{O}^{*\prime}$ then 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 $\textsf{No-Matching}_{\mathcal{A}}(t)$ has occured 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 $\mathsf{Exec}^{\mathsf{OW\text{-}MS}}_{\mathcal{A},\Pi'}(t)$ 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 $\mathcal{O}^{K}_{\mathrm{MAC}}(\cdot)$ query) and it verifies correctly with the message $(1 \mid\mid \tau')$ 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 $(\mathrm{tag}, 1 \mid\mid \tau')$ and otherwise $\mathcal{C}$ outputs $(\mathrm{tag}, 0 \mid\mid \tau')$, where τ' is the transcript of $\mathcal{O}^{*\prime}$, as part of the $\mathsf{UF\text{-}CMA}$ game against Mac. Hence, since the adversary $\mathcal{C}$ correctly guesses that the oracle for which the $\mathsf{No\text{-}Matching}_{\mathcal{A}}(t)$ condition will be satisfied is $\mathcal{O}^{*\prime}$ with probability $1/((n_P + n'_P) \cdot n_S)$, we obtain

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

Which proves the Lemma. ▌

H Proof of Theorem 5.3

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 advantage statement let $\mathcal{A}$ be an $\mathsf{IND\text{-}AK}$ adversary against $\Sigma = (\Pi'; \mathsf{AK}_{\mathsf{SSL}}(H))$. 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 scand 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 advantage statement then follows.

I Application Security

An application is a operation which makes use of secret keys. Different applications will clearly have different security definitions, before focusing on confidentiality we give some general definitions which should hold for all applications.

An application consists of a set of participants $\mathcal{I}$. The adversary always has access to two queries:

- $\mathsf{Join}(I)$: This takes a single participant $I \in \mathcal{I}$ and produces a new oracle Ω_I. The behaviour of the Ω_I oracles is dependent on the precise application. However, hidden from view of the adversary there is also generated a secret key $k_I \in \mathcal{S}_{\mathrm{A}}$.
- $\mathsf{Reveal}(\Omega_I)$: This gives the adversary the value of the secret key k_I. An oracle on which $\mathsf{Reveal}(\Omega_I)$ has been performed is called a revealed oracle.

The exact properties of the Ω_I oracles and the security game played by the adversary depends on the precise application. The main application is however one of confidentiality.

I.1 Confidentiality Encryption Applications:

We follow the game from Mazare and Warinschi [26] to define security of symmetric encryption in the presence of adaptive corruptions. For the security game the challenger selects a random bit $b \in \{0, 1\}$. The adversary has access to three additional queries, in addition to the Join and Reveal queries above;

- $\mathcal{O}_{\mathcal{E}}(\Omega_I, m)$: This takes a message m and forms its encryption under the symmetric key k_I. The ciphertext is returned to the user.
- $\mathcal{O}_{\mathcal{D}}(\Omega_I, c)$: This takes a ciphertext c and forms its decryption under the symmetric key k_I. If the decryption algorithm returns $\perp$ then so does this oracle, otherwise the message is returned.
- $\mathcal{O}_{\mathrm{LR}}(\Omega_I, m_0, m_1)$: This query may only be called if the oracle Ω_I has not been revealed, and the reveal oracle cannot be called on Ω_I if the oracle $\mathcal{O}_{\mathrm{LR}}(\Omega_I, m_0, m_1)$ has been called for some values of m_0 and m_1. The $\mathcal{O}_{\mathrm{LR}}(\Omega_I, m_0, m_1)$ oracle takes two messages of equal length, m_0 and m_1, and returns the encryption of m_b under the secret key k_I.

At the end of the game the adversary is required to output a bit b', with the adversary winning the game if $b' = b$. The advantage of an adversary is defined to be

$$\mathbf{Adv}_A(t) = \left|\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]\right|.$$

In [26] this security definition is called IND-ACCCA, and it is shown that an encryption scheme which is IND-CCA secure in the many-time setting is also secure in the above sense.

I.2 Combining Key Agreement and Applications

Having defined security for application key agreement protocols and for application protocols we join them together to form a *system protocol.* Informally, for confidentiality encryption applications, the way in which we do this is to use k' as the encryption key and k'' as the decryption key where $k = k' \mid\mid k''$ is the application key that a given accepted appliction key oracle $\mathcal{Q}$ holds and on which the Join query was made.

The formal definition, and security game, is defined as follows: We take the set up for the application key agreement protocols above, namely the sets $\mathcal{U}$, $\mathcal{U}'$, the public/private key pairs and the oracles Π^i_U, $\Sigma^j_{\Pi^i_U}$ etc. The set $\mathcal{I}$ is defined to be the set of all possible application key agreement oracles $\Sigma^j_{\Pi^i_U}$.

To define security for the whole system protocol we need to define what queries are allowed and what adversarial goal one is trying to achieve: The queries on the Π_U^i and $\Sigma_{\Pi_U^i}^j$ are defined as in the application key agreement game. One also defines a new query on an $\mathcal{Q} = \Sigma_{\Pi_U^i}^j$ oracle which has accepted, denoted Join which returns an application oracle $\Omega_{\mathcal{Q}}$, with the same application key k as is held by the internal state of the $\mathcal{Q}$ oracle. If $\mathsf{Reveal}(\Omega_{\mathcal{Q}})$ is called then we also assume that $\mathsf{Compromise}(\mathcal{Q})$ is also called, i.e. $\mathcal{O}$ becomes a compromised oracle.

To these queries we add the queries available in the specific application game; for example in the encryption game we add the oracles $\mathcal{O}_{\mathcal{E}}(\Omega_I^i, m)$, $\mathcal{O}_{\mathcal{D}}(\Omega_I^i, m)$ and $\mathcal{O}_{\mathrm{LR}}(\Omega_I^i, m_0, m_1)$ where $\mathcal{O}_{\mathcal{E}}$ and $\mathcal{O}_{\mathrm{LR}}$ encrypt using k' and $\mathcal{O}_{\mathcal{D}}$ decrypts using k''. The security goal of the adversary is defined to be the security goal as in the application game.

Theorem I.1 The above combination results in a secure protocol when our key agreement protocol is combined with an ACCCA-secure encryption scheme to produce a confidentiality application. ▌

Proof: The proof is immediate using standard game hopping. We only sketch it here.

First we move to a game in which one application key is chosen at random instead of by following the key agreement protocol. Any adversary which can distinguish between the two games clearly breaks the indistinguishability of the underlying application key agreement. A standard hybrid argument then extends this to all application keys being chosen at random.

Then the game we are in is simply the security game of IND-ACCCA. ▌

Note, that the above proof needs to be carefully applied as the encryption scheme used in the SSL is one of MAC-then-Encrypt, which is not known in general to be $\mathsf{IND\text{-}CCA}$ [4, 23]. However, for the specific MAC and encryption functions used in SSL, to perform MAC-then-Encrypt, one can show that the resulting encryption scheme is $\mathsf{IND\text{-}CCA}$, see [23] for details.