Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*

Ran Canetti[†] Hugo Krawczyk[‡]

May 17, 2001

Abstract

We present a formalism for the analysis of key-exchange protocols that combines previous definitional approaches and results in a definition of security that enjoys some important analytical benefits: (i) any key-exchange protocol that satisfies the security definition can be composed with symmetric encryption and authentication functions to provide provably secure communication channels; and (ii) the definition allows for simple modular proofs of security: one can design and prove security of key-exchange protocols in an idealized model where the communication links are perfectly authenticated, and then translate them using general tools to obtain security in the realistic setting of adversary-controlled links. We exemplify the usability of our results by applying them to obtain the proof of two main classes of key-exchange protocols, Diffie-Hellman and key-transport, authenticated via symmetric or asymmetric techniques.

Further contributions of the paper include the formalization of "secure channels" in the context of key-exchange protocols, and establishing sufficient conditions on the symmetric encryption and authentication functions to realize these channels.

---

*This document describes work in progress. Please check for future updates of this work at http://eprint.iacr.org. An extended abstract of this work appears in the proceedings of Eurocrypt 2001, LNCS Vol. 2045.

[†]IBM T.J. Watson Research Center, Yorktown Heights, New York 10598. Email: canetti@watson.ibm.com.

[‡]EE Department, Technion, Haifa, Israel. Email: hugo@ee.technion.ac.il. Supported by Irwin and Bethea Green & Detroit Chapter Career Development Chair.

Contents

1 Introduction

Key-exchange protocols (KE, for short) are mechanisms by which two parties that communicate over an adversarially-controlled network can generate a common secret key. KE protocols are essential for enabling the use of shared-key cryptography to protect transmitted data over insecure networks. As such they are a central piece for building secure communications (a.k.a "secure channels"), and are among the most commonly used cryptographic protocols (contemporary examples include SSL, IPSec, SSH, among others).

The design and analysis of secure KE protocols has proved to be a non-trivial task, with a large body of work written on the topic, including [18, 39, 12, 9, 19, 7, 8, 32, 2, 43] and many more. In fact, even today, after two decades of research, some important issues remain without satisfactory treatment. One such issue is how to guarantee the adequacy of KE protocols for their most basic application: the generation of shared keys for implementing secure channels. Providing this guarantee (with minimal requirements from KE protocols) is the main focus and objective of this work. The other central goal of the paper is in simplifying the usability of the resultant security definitions via a modular approach to the design and analysis of KE protocols. We exemplify this approach with a proof of security for two important classes of KE protocols.

This paper adopts a methodology for the analysis of KE protocols that results from the combination of two previous works in this area: Bellare and Rogaway [7] and Bellare, Canetti and Krawczyk [2]. A main ingredient in the formalization of [7] is the use of the indistinguishability approach of [24] to defining security: roughly speaking, a key-exchange protocol is called secure if under the allowed adversarial actions it is infeasible for the attacker to distinguish the value of a key generated by the protocol from an independent random value. Here we follow this exact same approach but replace the adversarial model of [7] with an adversarial model derived from [2]. This combination allows to achieve the above two main objectives. We elaborate on these main aspects of our work.

First, the formalization of [2] captures not only the specific needs of KE protocols but rather develops a more general model for the analysis of security protocols. This allows formulating and proving the statement that KE protocols proven secure according to our definition (we call these protocols SK-secure) can be used in standard ways to provide "secure channels". More specifically, consider the common security practice by which pairs of parties establish a "secure channel" by first exchanging a *session key* using a KE protocol and then using this key to encrypt and authenticate the transmitted data under symmetric cryptographic functions. We prove that if in this setting one uses an SK-secure KE protocol together with secure MAC and encryption functions combined appropriately then the resultant channel provides both authentication and secrecy (in a sense that we define precisely) to the transmitted data. While this property of ensuring secure channels seems as an obvious requirement from a secure KE protocol it turns out that formalizing and proving this property is non-trivial. In fact, there are "seemingly secure" key exchange protocols that do not necessarily guarantee this (e.g. those that use the session key during the exchange itself), as well as proposed definitions of secure key-exchange that do not suffice to guarantee this either (e.g., the definitions in [7, 10, 11, 2]). Moreover, although several works have addressed this issue (see Section 1.1), to the best of our knowledge the notion of secure channels was never formalized in the context of KE protocols, let alone demonstrating that some definition of KE protocols suffices for this basic task. Indeed, one of the contributions of this work is a formalization of the secure channels task. While this formalization is not intended to provide general composability properties for arbitrary cryptographic settings, it arguably provides sufficient security guarantee for the central task of protecting the integrity and authenticity of communications over adversarially-controlled links.

Second, the approach of [2] allows for a substantial simplification in designing KE protocols and proving their security. This approach postulates a two-step methodology by which protocols can first be designed and analyzed in a much simplified adversarial setting where the communication links are assumed to be ideally authenticated (i.e., the attacker is not allowed to insert or change information transmitted over the communication links between parties). Then, in a second step, these protocols are "automatically" transformed into secure protocols in the realistic scenario of fully adversary-controlled communications by applying a protocol translation tool (or "compiler") called an *authenticator*. Fortunately, simple and efficient realizations of authenticators based on different cryptographic functions exist [2] thus making it a useful and practical design and analysis tool. (We stress that our framework does not *mandate* this methodology; i.e., it is possible of course to prove security of a KE protocol directly in the fully adversarial model.)

We use this approach to prove the security of two important classes of key-exchange protocols: Diffie-Hellman and key-transport protocols. All one needs to do is to simply prove the security of these protocols in the ideal authenticated-links model and then, thanks to the above modular approach, one obtains versions of these protocols that are secure in a realistic adversary-controlled network. The "authenticated" versions of the protocols depend on the authenticators in use. These can be based either on symmetric or asymmetric cryptographic techniques (depending on the trust model) and result in natural and practical KE protocols. The security guarantees that result from these proofs are substantial as they capture many of the security concerns in real communications settings including the asynchronous nature of contemporary networks, the run of multiple simultaneous sessions, resistance to man-in-the-middle and known-key attacks, maintaining security of sessions even when other sessions are compromised, and providing "perfect forward secrecy", i.e., protection of past sessions in case of the compromise of long-term keying material.

1.1 Related work

Since its introduction in the seminal work of Diffie and Hellman [18] the notion of a key-exchange protocol has been the subject of many works (see [37] for an extensive bibliography). Here we mention some of the works that are more directly related to the present work. We further expand our discussion of these works in Appendix A.

Among the early works on this subject we note [39, 12, 9, 19] as being instrumental in pointing out to the many subtleties involved in the analysis of KE protocols. The first complexity-theoretic treatment of the notion of security for KE protocols is due to Bellare and Rogaway [7] who formalize the security of KE protocols in the realistic setting of concurrent sessions running in an adversary-controlled network. As said above, [7] apply the indistinguishability definitional approach that we follow here as well. While [7] focused on the shared-key model of authentication, other works [10, 11, 6] extended the techniques to the public-key setting. One important contribution of [6] is in noting and fixing a shortcoming in the original definition of [7]; this fix, that we adopt here, is fundamental for proving our results about secure channels.

Bellare, Canetti, and Krawczyk [2] present a model for studying general session-oriented security protocols that we adopt and extend here. They also introduce the "authenticator" techniques that allow for greatly simplifying the analysis of protocols and that we use as a basic tool in our work. In addition, [2] proposes a definition of security of KE protocols rooted in the simulatability (or "ideal third party") approach used to define security of multiparty computation [23, 38, 1, 13]. While this definitional approach is intuitively appealing the actual KE security definition of [2] comes short of the expectations. On one hand, it seems over-restrictive, in the sense that it rules out protocols that seem to provide sufficient security (and as demonstrated here can be safely used to obtain secure channels); on the other, it is not clear whether their definition suffices to prove composition theorems even in the restricted sense of secure channels as dealt with in this paper.

More recently, Shoup [43] presents a framework for the definition of security of KE protocols that follows the basic simulatability approach as in [2] but introduces significant modifications in order to overcome some of the shortcomings of the KE definition in [2] as well as to seek composability with other cryptographic applications. In particular, [43] states as a motivational goal the construction of "secure sessions" (similar to our secure channels), and it informally claims the sufficiency of its definitions to achieve this goal. A more rigorous and complete elaboration of that work will be needed to assess the correctness of these claims. In addition, [43] differs from our work in several other interesting aspects (see Appendix A).

A promising general approach for the analysis of reactive protocols and their concurrent composition has been developed by Pfitzmann, Schunter and Waidner [41, 40, 42] and Canetti [14]. This approach, that follows the simulatability tradition, can be applied to the task of key exchange to obtain a definition of KE protocols that guarantees secure concurrent composition with any set of protocols that make use of the generated keys. See more details in [16].

A subjective discussion. The works mentioned above follow two main distinct approaches to defining security of KE protocols: *simulation-based* and *indistinguishability-based.* The former is more intuitively appealing (due to its modeling of security via an ideally-trusted third party), and also appears to be more amenable to demonstrating general composability properties of protocols. On the other hand, the complexity of the resulting definitions, once all details are filled in, is considerable and makes for definitions that are relatively complex to work with. In contrast, the indistinguishability-based approach yields definitions that are simpler to state and easier to work with, however their adequacy for modeling the task at hand seems less clear at first glance. The results in this paper indicate the suitability of the indistinguishability-based approach in the context of KE protocols — if the goal is the application of KE protocols to the specific task of secure channels as defined here. By following this approach we gain the benefit of simpler analysis and easier-to-write proofs of security. At the same time, our work borrows from the simulation-based approach the modularity of building proofs via the intermediate ideally-authenticated links model, thus enjoying the "best of both worlds".

Organization. Section 2 presents an overview of the protocol and adversary models used throughout this work. This overview is intended to introduce the elements of this model in a "reader-friendly" way. The formal technical treatment appears in Section 3. The definition of SK-security for KE protocols is presented in Section 4. Section 5 proves the security of several protocols and illustrates the modular methodology used in our analysis. Finally, in Section 6 we introduce a formalization of "secure channels" and demonstrate the suitability of our notion of security for KE protocols for realizing secure channels.

2 Protocol and Adversary Models: An Overview

In order to to define what is meant by the security of a key-exchange (KE) protocol we first need to establish a formalism for the most basic notions: what is meant by a protocol in general and a key-exchange protocol in particular, what are sessions, and what is an 'attacker' against such protocols. Here we use a formalism based on the approach of [2], where a general framework for studying the security of session-based multi-party protocols over insecure channels is introduced. We extend and refine this formalism to better fit the needs of practical KE protocols.

In order to motivate and make the formalism easier to understand, *we start by providing a* *high-level overview of our model.* The precise technical description appears in Section 3. (We note that the precise technical details are essential for a full development and proof of our results. However, we recommend first reading this overview in order to make the technical part more understandable.) After introducing the protocol and adversary models we proceed to define the security of KE protocols in Section 4.

2.1 Protocols, Sessions and Key-Exchange

Message-driven protocols We consider a set of parties (probabilistic polynomial-time machines), which we usually denote by $P_1, \ldots, P_n$, interconnected by point-to-point links over which messages can be exchanged.[1] Protocols are collections of interactive procedures, run concurrently by these parties, that specify a particular processing of incoming messages and the generation of outgoing messages. Protocols are initially triggered at a party by an external "call" and later by the arrival of messages. Upon each of these events, and according to the protocol specification, the protocol processes information and may generate and transmit a message and/or wait for the next message to arrive. We call these message-driven protocols. (We note the asynchronous nature of protocols defined in this way which reflects the prevalent form of communication in today's networks.)

Sessions and protocol output. Protocols can trigger the initiation of sub-protocols (i.e. interactive subroutines) or other protocols, and several copies of such protocols may be simultaneously run by each party. We call each copy of a protocol run at a party a session. Technically, a session is an interactive subroutine executed inside a party. Each session is identified by the party that runs it, the parties with whom the session communicates and by a session-identifier. These identifiers are used in practice to bind transmitted messages to their corresponding sessions. Each invocation of a protocol (or session) at a given party creates a local state for that session during execution, and produces local outputs by that party. This output can be a quantity (e.g a session key) returned to the calling program, or it can be the recording of a protocol event (such as a successful or failed termination). These local outputs serve to represent the "history" of a protocol and are important to formalize security. When a session ends its run we call it complete and assume that its local state is erased.

Key-exchange protocols. Key-exchange (KE) protocols are message-driven protocols (as defined above) where the communication takes place between pairs of parties and which return, upon completion, a secret key called a session key. More specifically, the input to a KE protocol within each party P_i is of the form $(P_i, P_j, s, role)$, where P_j is the identity of another party, s is a session id, and *role* can be either initiator or responder. A session within P_i and a session within P_j are called matching if their inputs are of the form $(P_i, P_j, s, \mathsf{initiator})$ and $(P_j, P_i, s, \mathsf{responder})$. The inputs are chosen by a "higher layer" protocol that "calls" the KE protocol. We require the calling protocol to make sure that the session id's of no two KE sessions in which the party participates are identical. Furthermore, we leave it to the calling protocol to make sure that two parties that wish to exchange a key will activate matching sessions. Note that this may require some communication before the actual KE sessions are activated.[2]

Upon activation, the partners P_i and P_j of two matching sessions exchange messages (the initiator goes first), and eventually generate local outputs that include the name of the partners of the session,

[1] This formalization postulates a fixed number of parties in a network. An alternative, more general formalization allows the adversary to adaptively increase the number of participants. We prefer this simpler formalization since the difference seems inconsequential with respect to realistic KE protocols.

[2] Indeed, in practice protocols for setting up a secure session typically exchange some messages before the actual cryptographic key-exchange starts. The IKE protocol of the IPSEC standard is a good example [28].

the session identifier, and the value of the computed session key. A key establishment event is recorded only when the exchange is completed (this signals, in particular, that the exchanged key can be used by the protocol that called the KE session). We note that a session can be completed at one partner but not necessarily at the other.

After describing these 'mechanics" of a KE protocol we need to define what is meant by a "secure" KE protocol. This is the subject of Section 4 and it is based on the adversarial model that we introduce next.

2.2 The unauthenticated-links adversarial model (UM)

In order to talk about the security of a protocol we need to define the adversarial setting that determines the capabilities and possible actions of the attacker. We want these capabilities to be as generic as possible (as opposed to, say, merely representing a list of possible attacks) while not posing unrealistic requirements. We follow the general adversarial formalism of [2] but specialize and extend it here for the case of KE protocols. Using the terminology of [2] we call this model the Unauthenticated Links Model (UM).

Basic attacker capabilities. We consider a probabilistic polynomial-time (PPT)[3] attacker that has full control of the communications links: it can listen to all the transmitted information, decide what messages will reach their destination and when, change these messages at will or inject its own generated messages. The formalism represents this ability of the attacker by letting the attacker be the one in charge of passing messages from one party to another. The attacker also controls the scheduling of all protocol events including the initiation of protocols and message delivery.

Obtaining secret information. In addition to these basic adversarial capabilities (given "for free" to the attacker), we let the attacker obtain secret information stored in the parties memories via explicit attacks. We consider all the secret information stored at a party as potentially vulnerable to break-ins or other forms of leakage. However, when defining security of a protocol it is important to guarantee that the leakage of some form of secret information has the least possible effect on the security of other secrets. For example, we will want to guarantee that the leakage of information specific to one session (such as the leakage of a session key or ephemeral state information) will have no effects on the security of other sessions, or that even the leakage of crucial long-term secrets (such as private keys) that are used across multiple sessions will not necessarily compromise secret information from all past sessions. In order to be able to differentiate between various vulnerabilities and to be able to guarantee as much security as possible in the event of information exposures, we classify attacks into three categories depending on the type of information accessed by the adversary:

Party corruption. The attacker can decide at any point to corrupt a party, in which case the attacker learns *all* the internal memory of that party including long-term secrets (such as private keys or master shared keys used across different sessions) and session-specific information contained in the party's memory (such as internal state of incomplete sessions and session-keys corresponding to completed sessions). Since by learning its long term secrets the attacker can impersonate a party in all all its actions then a party is considered completely controlled by the attacker from the time of corruption and can, in particular, depart arbitrarily from the protocol specifications.

Session-key query. The attacker provides a party's name and a session identifier of a *completed* session at that party and receives the value of the key generated by the named session This attack provides the formal modeling for leakage of information on specific session keys that may result from

[3]When proving specific protocols one can replace this generic PPT modeling with specific cryptographic assumptions.

events such as break-ins, cryptanalysis, careless disposal of keys, etc. It will also serve, indirectly, to ensure that the unavoidable leakage of information produced by the use of session keys in a security application (e.g., information leaked on a key by its use as an encryption key) will not help in deriving further information on this and other keys.
Note: one could define yet another adversary operation that would provide the attacker with partial information on session keys (to specifically model information leaked via key usage), however it turns out that such an addition, while adding complexity to the model specification, does not change the power of the model since session-key queries as defined here already suffice to capture leakage of any partial information on the session keys.

Session-state reveal. The attacker provides the name of a party and a session identifier of a *yet incomplete* session at that party and receives the internal state of that session (since we see sessions as procedures running inside a party then the internal state of a session is well defined). An important point here is what information is included in the local state of a session; this is to be specified by each KE protocol. Therefore, our definition of security is parameterized by the type and amount of information revealed in this attack. For instance, the information revealed in this way may be the exponent x used by a party to compute a value g^x in a Diffie-Hellman key-exchange protocol, or the random bits used to encrypt a quantity under a probabilistic encryption scheme during a session. (An example where such state information may be vulnerable to attack is applications – such as those running in low-powered devices – that pre-compute, or upload, a file of pairs (x, g^x) for use during later "real-time" establishment of KE sessions. In this case one would like to prevent that the exposure of such a file, or part of it, will compromise future sessions that do not use these values.)

We stress that while the first two forms of attack, party corruptions and session-key queries, are fundamental to the definition of security of KE protocols, the significance of the session-state reveal operation depends on the security model of an implementation. The differentiation between party corruptions and session-state reveal operations assumes that corrupting a session state does not imply learning about long-term secrets; this implicitly assumes a separate security module where the operations involving these long-term secrets are performed. In settings where this is an unrealistic assumption, our model can be weakened by deleting the session-state reveal operation from the attacker's capabilities. Certainly, protocols proven secure under our model will remain secure in the weakened model.

Terminology: if a session is subject to any of the above three attacks (i.e. a session-state reveal, a session-key query or the corruption of the party holding the session) then the session is called locally exposed. If a session or its matching session is locally exposed then we call the session exposed.

Session expiration. One important additional element in our security model is the notion of session expiration. This takes the form of a protocol action that when activated causes the erasure of the named session key (and any related session state) from that party's memory. We allow a session to be expired at one party without necessarily expiring the matching session. The effect of this action in our security model is that the value of an expired session key cannot be found via any of the above attacks if these attacks are performed after the session expired. This has two important consequences: it allows us to model the common (and good) security practice of limiting the life-time of individual session keys and it allows for a simple modeling of the notion of perfect forward secrecy (see Section 4.2). We note that in order for a session to be locally exposed (as defined above) the attack against the session must happen *before* the session expires.

Bootstrapping the security of key-exchange protocols. Key-exchange protocols, as other cryptographic applications, require the bootstrapping of security (especially for authentication) via some assumed-secure means. Examples include the secure generation of parties' private keys, the installation of public keys of other parties, or the installation of shared "master" keys. Here too we follow the approach of [2] where the bootstrapping of the authentication functions is abstracted into an initialization function that is run prior to the initiation of any key-exchange protocol and that produces in a secure way (i.e. without adversarial participation) the required (long-term) information. By abstracting out this initial phase we allow for the combination of different protocols with different initialization functions: in particular, it allows our analysis of protocols (such as Diffie-Hellman) to be applicable under the two prevalent settings of authentication: symmetric and a-symmetric authentication. Two points to note are (1) the specification of the initialization function is part of the definition of each KE protocol; and (2) secret information generated by this function at a given party can be discovered by the attacker only upon corruption of that party. We stress that while this abstraction adds to the simplicity and applicability of our analysis techniques, the bootstrapping of security in actual protocols is an element that must be carefully analyzed (e.g., the interaction with a CA in the case of public-key based protocols). Integrating these explicit elements into the model can be done either directly as done in [43], or in a more modular way via appropriate protocol composition.

2.3 The AM, protocol emulation and authenticators

A central ingredient in our analyses is the methodology introduced in [2] by which one can design and analyze a protocol under the highly-simplifying assumption that the attacker cannot change information transmitted between parties, and then transform these protocols and their security assurance to the realistic UM where the adversary has full control of the communication links. We refer the reader to [2] for the details and also present a technical summary in Section 3.4.

First, an adversarial model called authenticated-links model (denoted AM) is defined in a way that is identical to the UM with one fundamental difference: *the attacker is restricted to only deliver messages truly generated by the parties without any change or addition to them.* Then, the notion of "emulation" is introduced in order to capture the equivalence of functionality between protocols in different adversarial models, in particular between the UM and AM. Roughly speaking, a protocol π' emulates protocol π in the UM if for any adversary that interacts with π' in the UM there exists an adversary that interacts with π in the AM such that the two interactions "look the same" to an outside observer. Finally, special algorithms called authenticators are developed with the property that on input the description of a protocol π the authenticator outputs the description of a protocol π' such that π' emulates protocol π in the UM. That is, authenticators act as an automatic "compiler" that translate protocols in the AM into equivalent (or "as secure as") protocols in the UM.

In order to simplify the construction of authenticators, [2] offers the following methodology. First consider a very simple one-flow protocol in the AM, called MT, whose sole functionality is to transmit a single message from sender to recipient. Now build a restricted-type authenticator, called MT-authenticator, required to provide emulation for this particular MT protocol only. Finally, to any such MT-authenticator λ one associates an algorithm (or compiler) C_λ that translates any input protocol π into another protocol π' as follows: to each of the messages defined in protocol π apply the MT-authenticator λ. It is proven in [2] that C_λ is an authenticator (i.e., the resultant protocol π' emulates π in the UM). Particular realizations of MT-authenticators are presented in [2] based on different type of cryptographic functions (e.g., digital signatures, public-key encryption, MAC, etc.)

3 The models

This section presents a technical description of the protocol and adversary models used throughout the paper. We strongly recommend first reading Section 2 which presents an overview of these models and their motivation.

3.1 Message-Driven Protocols

An n-party message-driven protocol is a collection of n programs, where each program is to be run by a different party. (Formally, each program is an interactive PPT Turing machine, as defined in [25].) Each program has the following interface. It is first invoked with some initial input (that includes the party's identity), random input, and some value for the security parameter. Once invoked, the program waits for an activation. An activation can be caused by two types of events: either the arrival of an incoming message from the network, or an action request coming from other programs run by the party. (Defining valid action requests is part of the specification of the protocol.[4]) Upon activation, the program processes the incoming data, starting from its current internal state, and as a result it can generate outgoing messages to the network and action requests to other programs run by the party. In addition, a local output value is generated. Once the activation is completed, the program waits for the next activation. We regard the local output as cumulative. That is, initially the local output is empty; in each activation the current output is appended to the previous one. We will let a protocol label some of its local output as 'secret' (e.g. the value of a secret key generated by the protocol). This will have effect on the adversary's actions that we define below.

An invocation of a protocol is called a session. Note that a session of a protocol π may involve several sessions of other protocols that are called by π. (When treating the special case of key-exchange protocols in Section 3.3 the semantics of sessions in that context will be given more specific meaning.)

3.2 The unauthenticated-links adversarial model (UM).

The adversarial model UM defines the attacker's capabilities and its interaction with a protocol. Figure 1 summarizes the way protocols are executed in the presence of a UM adversary. Here we describe this in some more detail. Consider an n-party message-driven protocol π, with parties denoted by $P_1 ... P_n$. Each party P_i has input x_i and random input r_i. In addition, we introduce an adversarial entity, called a UM-adversary $\mathcal{U}$. (The UM-adversary is another program, or a PPT interactive Turing machine, with an interface described below.) The execution of protocol π in the UM consists of a sequence of activations of π within different parties. The activations are controlled and scheduled by $\mathcal{U}$. That is, initially the protocol is invoked within each party with a local input, random input and a value for the security parameter. Next, and upon the completion of each activation, $\mathcal{U}$ decides which party to activate next, and on which incoming message or request. The outgoing messages and outgoing local action requests become known to $\mathcal{U}$. Local outputs become known to $\mathcal{U}$ except for those labeled 'secret'.

Note that $\mathcal{U}$ is free to choose to activate any party with any activation allowed by the protocol and in any order. Also, $\mathcal{U}$ can activate any party with any incoming message and any specified

[4]An action request can be, for instance, a request to send a message or exchange a key with some specified party (we will see specific examples in the sequel). We assume that every message specifies the sender of the message and its intended recipient.

Protocol execution in the UM

Participants: Parties $P_1, ..., P_n$ running an n-party protocol π on inputs $x_1, ..., x_n$, respectively, and an adversary $\mathcal{U}$.

1. Initialization: Each party P_i invokes π on local input x_i, security parameter k and random input. Next, P_i gets $I(r,k)_i$ and $I(r,k)_0$, where r is randomly chosen.
2. While $\mathcal{U}$ has not terminated do:
   (a) $\mathcal{U}$ may activate π within some party, P_i. An activation can take two forms:
      i. An action request q. This activation models requests or invocations coming from other programs run by the party.
      ii. An incoming message m with a specified sender P_j. This activation models messages coming from the network.

   If an activation occurred then the activated party P_i runs its program and hands $\mathcal{U}$ the resulting outgoing messages and action requests. (We stress that $\mathcal{U}$ is free to choose any scheduling of activations and determine the values of incoming messages.) Local outputs produced by the protocol are known to $\mathcal{U}$ except for those labeled 'secret'.

   (b) $\mathcal{U}$ may corrupt a party P_i. Upon corruption, $\mathcal{U}$ learns the current internal state of P_i, and a special message is added to P_i's local output. From this point on, P_i is no longer activated and does not generate further local output.

   (c) $\mathcal{U}$ may issue a session-state reveal for a specified session within some party P_i. In this case, $\mathcal{U}$ learns the current internal state of the specified session within P_i. This event is recorded through a special note in P_i's local output.

   (d) $\mathcal{U}$ may issue a session-output query for a specified session within some party P_i. In this case, $\mathcal{U}$ learns any output from the specified session that was labeled 'secret'. This event is recorded through a special note in P_i's local output.
3. The global output of the execution is the concatenation of the outputs of $\mathcal{U}$ and $P_1, ..., P_n$.

Figure 1: Protocol execution in the UM.

sender. In particular, incoming messages need not correspond in any way to messages that have been sent. (That is, $\mathcal{U}$ is free to generate, inject, modify, and deliver any message of its choice.)

In addition to activating parties and controlling the network, $\mathcal{U}$ can perform the following activities. First, it can corrupt parties at will. Upon corruption of P_i, $\mathcal{U}$ learns the entire *current* state of P_i, including any long-term secret, session states and secret session outputs in the party's memory. From this point on, $\mathcal{U}$ can deliver any message of its choice in which P_i is specified as the sender. The corrupted party P_i appends a special note to its output, specifying that it has been corrupted. P_i is no longer activated and does not generate further local output. (A corrupted party is totally controlled by the adversary, and its actions are taken by the attacker itself.) Another type of activity is session-state reveal of a certain session within party P_i. The effect is that the internal state of the corresponding session within P_i (i.e., the local working space of the procedure whose invocation constitutes the session) becomes known to $\mathcal{U}$, and a special message is added to the party's local output; no further output is generated for this session.[5] A third

[5]We do not specify how a session is identified; this will have to be part of the specification of a protocol. In the

adversarial activity is a session-output query. By issuing such a query the adversary learns any output from that session that was labeled 'secret'. (This type of queries is particularly important in the context of key-exchange protocols below where this action is called a 'session-key query'.)

The initialization function I. Finally, we augment the protocol π with an initialization function I that models an initial phase of out-of-band and authenticated information exchange between the parties. (This function models the necessary trusted bootstrapping of cryptographic functions, e.g. by letting the parties choose private and public keys for some asymmetric crypto-system and trustfully distributing the public keys.) Function I takes a random input r and the security parameter k, and outputs a vector $I(r,k) = I(r,k)_0 \ldots . I(r,k)_n$. The component $I(r,k)_0$ is the public information and becomes known to all parties and to the adversary. For $i > 0$, $I(r,k)_i$ becomes known only to P_i. Note, however, that upon corruption of P_i the attacker learns $I(r,k)_i$.

Global output. The global output of running a protocol in the UM is the concatenation of the cumulative local outputs of all the parties, together with the output of the adversary. The output of the adversary is a function of its internal states at the end of the interaction. We use the following notation. Let $\text{UM-ADV}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r})$ denote the output of adversary $\mathcal{U}$ when interacting with parties running protocol π on security parameter k, input $\vec{x} = x_1 \ldots x_n$ and random input $\vec{r} = r_0 \ldots r_n$ as described above (r_0 for $\mathcal{U}$; x_i and r_i for party P_i). (The initialization function I is part of the description of protocol π.) Let $\text{UNAUTH}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r})_i$ denote the cumulative output of party P_i after running protocol π on security parameter k, input $\vec{x}$ and random input $\vec{r}$, and with an AM-adversary $\mathcal{U}$. Let $\text{UNAUTH}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r}) = \text{UM-ADV}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r}), \text{UNAUTH}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r})_1 \ldots \text{UNAUTH}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r})_n$. Let $\text{UNAUTH}_{\pi,\mathcal{U}}(k,\vec{x})$ denote the random variable describing $\text{UNAUTH}_{\pi,\mathcal{U}}(k,\vec{x},\vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{UNAUTH}_{\pi,\mathcal{U}}$ denote the ensemble $\{\text{UNAUTH}_{\pi,\mathcal{U}}(k,\vec{x})\}_{k\in\mathbf{N},\vec{x}\in\{0,1\}^*}$.
We have summarized the structure of a protocol execution in the UM in Figure 1.

3.3 Key Exchange Protocols

Key-exchange protocols are a special case of n-party message-driven protocols. As such they inherit the syntax of general message-driven protocols as introduced before. In addition, in order to capture the specific semantics of key exchange, and the specific capabilities of attackers against such protocols, we specify some additional syntax for these protocols. (The intention of this syntax is to represent, in an abstract but direct way, the mechanics of key exchange protocols in actual systems.)

Recall that a message-driven protocol is a collection of n programs, where each program is run by a different party. (We envision that the program is invoked once within each party at the onset of the computation and remains active throughout.) Once invoked, it is activated either by a message coming from the network, or by an action request from other protocols or programs run by the party. In the case of a key-exchange (KE) protocol π, the program within each party, P_i, takes action requests of the form establish-session$(P_i, P_j, s, \mathit{role})$ where P_j is another party (with which a key is to be exchanged), s is a string called the session-id, and $\mathit{role} \in \{\text{initiator}, \text{responder}\}$. (This action request will typically be triggered by other protocols run by the party that "call" the KE protocol, see for example Section 6.)
Local outputs of a KE protocol are of the form (P_i, P_j, s, κ), where P_j, s are as above and κ is a session key. A null value of κ is interpreted as a "session abortion" and will usually represent the termination of the session with a returned error message. Non-null session-key values are labeled

context of KE protocols we will identify sessions via a session-id and the partners of the session; see more details in the next section.

'secret'. (Recall that the local outputs are thought of as values returned by the session to the "calling protocol" that issued the initial establish-session activation.)

We further specify the internal structure of each of the n programs of a KE protocol, as follows. Each such program, running within P_i, consists of a main procedure (can be thought of as a "shell") and a special subroutine, called a KE-subroutine. An invocation of the KE-subroutine is called a KE-session and is aimed at exchanging a single key with a specified party. The main procedure proceeds as follows. Upon activation with action request establish-session$(P_i, P_j, s, \mathit{role})$, it first verifies that no KE-session was previously invoked (within P_i) with inputs $(P_i, P_j, s, \mathit{role}')$ for some $\mathit{role}' \in \{\text{initiator}, \text{responder}\}$ (namely, the main procedure makes sure that the identity of the session is unique among the sessions that P_i was requested to establish with P_j). If the verification fails, then an appropriate error message is generated. Otherwise, a KE-session is invoked with inputs $(P_i, P_j, s, \mathit{role})$. From this point on, whenever the KE protocol within P_i receives a message that specifies sender P_j and session-id s, it forwards this message to the relevant KE-session within P_i. Once a KE-session returns (typically, after a number of messages have been exchanged between P_i and P_j) with output (P_i, P_j, s, κ), the KE protocol records a session establishment event with parameters (P_i, P_j, s, κ) in its local output. From these parameters only the value κ of the session key is labeled 'secret'. A KE-session that returns with a non-null value of κ is called completed. If $\kappa = \mathit{null}$ then the KE-session is aborted and a special note is recorded in the local output. It is assumed by convention that, once a KE-session returns, its entire local state, except for the output value, is securely erased. Note that this means that a session-state reveal after the session has returned will produce an empty output for the attacker.

Matching sessions. We also use the following terminology: if in an execution of a KE protocol P_i has a session with input $(P_i, P_j, s, \mathit{role})$ and party P_j has a session with input $(P_j, P_i, s', \mathit{role}')$, and $s = s'$ then we say that the two sessions are matching. (Note that we do not require that $\mathit{role} \neq \mathit{role}'$.) We call P_i and P_j the partners of session s. (Note that P_i may have completed a session with partner P_j, while P_j may never complete the matching session; completion of sessions depends on the delivery of the protocol's message which is subject to adversarial control.)

Session expiration: an extension to the UM. The adversarial actions against a KE protocol in the UM are essentially the same as the generic UM attacker described above, including party corruption, session-state reveals, and session-output queries. For clarity, we will use the term session-key query instead of session-output query when referring to KE sessions (namely, a session-key query on a completed session provides the attacker with the value of that session key, the only secret output of a KE session). We add, however, one more element to this model. We will consider a protocol action called session expiration. A session expiration action can be scheduled by the attacker for any completed session $(P_i, P_j, s, \mathit{role})$ within party P_i. The effect of this activity is that the secret output of the session, i.e. the session key, is erased from the party's memory. In addition, a special note recording the session expiration is added to P_i's local output, and this KE-session is labeled expired, with the following consequences. Adversary $\mathcal{U}$ is not allowed to perform a session-key query for an expired session. In addition, when $\mathcal{U}$ corrupts a party, it does *not* see the local outputs of the expired sessions (thus, upon party corruption the attacker learns the party's session-keys for *unexpired* sessions only.) As explained in Section 2 expiration of sessions is motivated by the common practice to limit the life time of session keys and, in particular, is instrumental for capturing the notion of perfect forward secrecy. Figure 1 needs to be updated by adding the session expiration activity to the list of possible activities in Step 2.

Exposed sessions. Finally we introduce the following terminology. A KE-session $(P_i, P_j, s, \mathit{role})$ within P_i is called *locally exposed (within P_i)* if the attacker performed any of the following actions

on said session: (i) a session-state reveal; (ii) a session-key query; (iii) corruption of P_i before session $(P_i, P_j, s, role)$ expired within P_i (this includes the case in which P_i is corrupted before the session is even invoked or completed).
A KE-session is called exposed if it is locally exposed or it has a matching session that is locally exposed. A session which is not exposed is called unexposed.

3.4 The AM Model and Authenticators

The material in this Section is taken from [2].

The authenticated-links adversarial model (AM). The authenticated-links model of computation is identical to the unauthenticated-links one, with the following fundamental exception. The AM-adversary, denoted $\mathcal{A}$, can activate parties only with incoming messages that were generated and sent by other parties in the protocol. That is, the attacker cannot inject or modify messages (except if the specified sender is a corrupted party or if the message belongs to an exposed session). In addition, any message may be delivered at most once. (Namely, $\mathcal{A}$ may decide not to deliver a message at all, but if $\mathcal{A}$ delivers a message m then it can do so only to the proper destination of m, only once, and without changing m or the specified sender.)

We define $\text{AUTH}_{\pi,\mathcal{A}}$ analogously to $\text{UNAUTH}_{\pi,\mathcal{U}}$, where the computation is carried out in the unauthenticated-links model.

Emulation of protocols. Central to the methodology of [2] and the current paper is the concept of "protocol translation", especially between the AM to the UM. We want to be able to start with any protocol π that has some guaranteed functionality (or security) in the AM and generate out of it a protocol π' with *equivalent functionality* in the UM. For this we first need to formalize the notion of "equivalence". This is done in the next definition from [2] (and which follows a general approach used for defining secure multi-party protocols [23, 38, 1, 13]).

Definition 1 *Let π and π' be an n-party message-driven protocols. We say that π'* emulates *π* in the unauthenticated-links model *if for any UM-adversary $\mathcal{U}$ there exists an AM-adversary $\mathcal{A}$ such that $\text{AUTH}_{\pi,\mathcal{A}}$ and $\text{UNAUTH}_{\pi',\mathcal{U}}$ are computationally indistinguishable.*

Armed with the emulation definition we can turn to define what is meant by "protocol translation" from AM to UM. This is done in the next definition [2] in terms of "compilers" and "authenticators".

Definition 2 *A* compiler *$\mathcal{C}$ is an algorithm that takes for input descriptions of protocols and outputs descriptions of protocols. An* authenticator *is a compiler $\mathcal{C}$ where for any protocol π, the protocol $\mathcal{C}(\pi)$ emulates π in the unauthenticated-links model.*

Constructing authenticators: the MT protocol. Thus, an authenticator can take for input protocols designed for ideally authenticated links (AM), and turn them into 'equivalent' protocols for adversary-controlled unauthenticated links (UM). But can such authenticators be constructed? The answer is yes. The following methodology for constructing authenticators is used in [2]. Consider the following simple protocol, called the message transmission (MT) protocol. The protocol takes empty input. Upon activation within P_i on action request send(P_i, P_j, m), party P_i sends the message (P_i, P_j, m) to party P_j, and outputs `''`P_i `sent` m `to` P_j`''`. Upon receipt of a message (P_i, P_j, m), P_j outputs `''`P_j `received` m `from` P_i`''`. When run in the AM this protocol *represents a perfectly authenticated message transmission protocol.* Now, let λ be a protocol that emulates MT in unauthenticated networks. We call such protocols MT-authenticators and we will see that they can be constructed efficiently. On the basis of λ, define a compiler $\mathcal{C}_\lambda$ that on input a protocol π produces a protocol $\pi' = \mathcal{C}_\lambda(\pi)$ defined as follows. When π' is activated at a party P_i it first invokes λ. Then, for each message sent in protocol π, protocol π' activates λ with the action request for sending the same message to the same specified recipient. Whenever π' is activated with some incoming message, it activates λ with the same incoming message. When λ outputs ''P_i `received` m `from` P_j'', protocol π is activated with incoming message m from P_j. It is shown:

Theorem 3 ([2]) *Let λ be an* MT*-authenticator. Then $\mathcal{C}_\lambda$ is an authenticator.*

Thus, in order to see that authenticators can be constructed it suffices to show constructions of MT-authenticators. This is done in [2] where several such schemes are shown based on different cryptographic functions (such as digital signatures and encryption).

In Section 6.2 we extend the MT protocol to a setting of multiple concurrent sessions. We call the resultant protocol SMT. It is straightforward to extend the proof of the above theorem to cover the case of SMT-authenticators as well.

4 Session-Key Security

After having defined the basic formal model for key-exchange protocols and adversarial capabilities, we proceed to define *what is meant for a key-exchange protocol to be secure.* While the previous sections were largely based on the work of [2], our definition of security closely follows the definitional approach of [7]. The resultant notion of security, that we call *session-key security* (or *SK-security*), focuses on ensuring the security of individual session-keys as long as the session-key value is not obtained by the attacker via an explicit key exposure (i.e. as long as the session is *unexposed* – see the terminology in the previous section). We want to capture the idea that the attacker "does not learn anything about the value of the key" from interacting with the key-exchange protocol and attacking other sessions and parties. As it is standard in the semantic-security approach this is formalized via the infeasibility to distinguish between the real value of the key and an independent random value.

We stress that this formulation of SK-security is very careful about tuning the definition to offer enough strength as required for the use of key-exchange protocols to realize secure channels (Section 6), as well as being realistic enough to avoid over-kill requirements which would prevent us from proving the security of very useful protocols (Section 5). We further discuss these aspects after the presentation of the definition.

4.1 Definition of SK-Security

We first present the definition for the UM. The formalization in the AM is analogous. We start by defining an "experiment" where the attacker $\mathcal{U}$ chooses a session in which to be "tested" about information it learned on the session-key; specifically, we will ask the attacker to differentiate the real value of the chosen session key from a random value. (Note that this experiment is an artifact of the definition of security, and not an integral part of the actual key-exchange protocols and adversarial intervention.)

For the sake of this experiment we extend the usual capabilities of the adversary, $\mathcal{U}$, in the UM by allowing it to perform a *test-session query.* That is, in addition to the regular actions of $\mathcal{U}$ against a key-exchange protocol π, we let $\mathcal{U}$ to choose, at any time during its run, a *test-session*

among the sessions that are completed, unexpired and unexposed at the time. Let κ be the value of the corresponding session-key. We toss a coin b, $b \xleftarrow{R} \{0,1\}$. If $b=0$ we provide $\mathcal{U}$ with the value κ. Otherwise we provide $\mathcal{U}$ with a value r randomly chosen from the probability distribution of keys generated by protocol π. The attacker $\mathcal{U}$ is now allowed to continue with the regular actions of a UM-adversary but *is not allowed to expose the test-session* (namely, it is not allowed session-state reveals, session-key queries, or partner's corruption on the test-session or its matching session.[6]) At the end of its run, $\mathcal{U}$ outputs a bit b' (as its guess for b).
We will refer to an attacker that is allowed test-session queries as a KE-adversary.

Definition 4 *A* KE *protocol π is called* SK-secure *if the following properties hold for any* KE-*adversary $\mathcal{U}$ in the* UM.

1. *Protocol π satisfies the property that if two* uncorrupted *parties complete matching sessions then they both output the same key; and*

2. *the probability that $\mathcal{U}$ guesses correctly the bit b (i.e., outputs $b'=b$) is no more than 1/2 plus a negligible fraction in the security parameter.*

If the above properties are satisfied for all KE-*adversaries in the* AM *then we say that π is* SK-secure in the AM.

The first condition is a "consistency" requirement for sessions completed by two *uncorrupted* parties. We have no requirement on the session-key value of a session where one of the partners was corrupted before the session completed – in fact, most KE protocols allow a corrupted party to strongly influence the exchanged key. The second condition is the "core property" for SK-security. We note that the term 'negligible' refers, as customary, to any function (in the security parameter) that diminishes asymptotically faster than any polynomial fraction. (This formulation allows, if so desired, to quantify security via a concrete security treatment. In this case one quantifies the attacker's power via specific bounds on computation time, number of corruptions, etc., while its advantage is bounded through a specific parameter ε.)

Remark. We highlight three aspects of Definition 4.

- The attacker can keep running and attacking the protocol even after receiving the response (either real or random) to its test-session query. This ability (which represents a substantial strengthening of security relative to [7], see also [6]) is *essential* for proving the main property of SK-security shown in this paper, namely its guarantee of security when used to generate secure channels as described in Section 6. See the Appendix for historic background on, as well as some technical rationale for this requirement.

- The attacker is not allowed to corrupt partners to the test-session or issue any other exposure command against that session while unexpired. This reflects the fact that there is no way to guarantee the secure use of a session-key that was exposed via an attacker's break-in (or cryptanalysis). In particular, this restriction is instrumental for proving the security of specific important protocols (e.g., Diffie-Hellman key exchange) as done in Section 5.

[6]We stress, however, that the attacker *is allowed* to corrupt a partner to the test-session as soon as the test-session (or its matching session) expires at that party. See the discussion below. This may be the case even if the other partner has not yet expired the matching session or not even completed it.

- The above restriction on the attacker by which it cannot corrupt a partner to the test-session is lifted as soon as the session expires at that partner. In this case the attacker should remain unable to distinguish between the real value of the key from a random value. This is the basis to the guarantee of "perfect forward secrecy" provided by our definition and further discussed in Section 4.2.

We stress that in spite of its "compact" formulation Definition 4 is very powerful and can be shown to ensure many specific properties that are required from a good key-exchange protocol (see, for example, chapter 12 of [37]). Some of these properties include the guarantee that session-keys belong to the right probability distribution of keys (except if one of the partners is corrupted at time of exchange), the "authenticity" of the exchange (namely, a correct and consistent binding between keys and parties' identities), resistance to man-in-the-middle attacks (for protocols proven SK-secure in the UM), resistance to known-key attacks, forward secrecy, and more. However, we note that all these properties (which are sometimes listed as a replacement to a formal definition of security) in combination do not suffice to guarantee the most important aspect of key-exchange security that SK-security enjoys: namely, the composition of the key-exchange protocols with cryptographic functions to enable secure channels (e.g., the original definition of security in [7] does satisfy the above list of properties but is insufficient to guarantee secure channels).

We finally remark that Definition 4 makes security requirements from a KE protocol only in case that the protocol completes KE-sessions. No guarantee is made that KE-sessions will ever return, or that they will not be aborted, i.e., that the corresponding session key will not be null. (In fact, a KE protocol where all KE-sessions "hang" and never return satisfies the definition.) One can add an explicit termination requirement for sessions in which the parties are uncorrupted and all messages are correctly delivered by the attacker. For simplicity, we choose to leave the analysis of the termination properties of protocols out of the scope of the definition of security.

4.2 Forward Secrecy

Informally, the notion of "perfect forward secrecy" (PFS) [26, 19] is stated as the property that "compromise of long-term keys does not compromise past session keys". In terms of our formalism this means that even if a party is corrupted (in which case all its stored secrets – short-term and long-term – become known to the attacker) then nothing is learned about sessions within that party that were previously unexposed and *expired* before the party corruption happened.

The provision that *expired* session-keys remain indistinguishable from random values even if a partner to that session is corrupted guarantees the perfect forward secrecy of SK-secure protocols. Put in other words, when proving a protocol to be SK-secure using Definition 4 one automatically gets a proof that that protocol guarantees PFS.

On the other hand, while PFS is a very important security property it is not required for all application scenarios, e.g., when only authentication is required, or when short-term secrecy suffices. Indeed, it is common to find in practice protocols that do not provide PFS and still are not considered insecure. One such typical case are "key-transport protocols" in which public key encryption is used to communicate a session-key from one party to another. (In this case, even if session-keys are erased from memory when no longer required, the corruption of a party may allow an attacker to compute, via the discovered long-term private keys, all the past session-keys.) Due to the importance of such protocols (they are commonly used in, e.g., SSL), and given that achieving PFS usually has a non-negligible computational cost, we define a notion of "SK-security without PFS" by simply disallowing the protocol's action of key expiration. That is, under this modified model, *session-keys never expire.* This results in a weaker notion of security since now by virtue of Definition 4 the attacker is *never allowed* to corrupt a partner to the test-session (or in other words, this weaker definition of security does not guarantee the security of a session-key for which one of the partners is ever corrupted).

Definition 5 *We say that a* KE *protocol satisfies* SK-security without PFS *if it enjoys SK-security relative to any* KE*-adversary in the* UM *that is* not *allowed to expire keys. (Similarly, if the above holds for any such adversaries in the* AM *then we say that π is* SK-secure without PFS in the AM.*)*

Section 5.3 describes a protocol that satisfies SK-security without PFS but not regular SK-security.

5 SK-Secure Protocols

This section demonstrates the usability of our definition of SK-security for proving the security of some simple and important key-exchange protocols. One is the original Diffie-Hellman protocol, the other is a simple "key transport" protocol based on public-key encryption. We first show that these protocols are secure in the simpler authenticated-links model (AM). Then, using the methodology from [2] we can apply to these protocols a variety of (symmetric or asymmetric) authentication techniques to obtain key-exchange protocols that are secure in the realistic UM model. Namely, applying any MT-authenticator (see Sections 2.3 and 3.4) to the messages of the AM-protocol results in a secure KE protocol in the UM. The next Theorem states that this methodology does work for our purposes.

Theorem 6 *Let π be a SK-secure key-exchange protocol in the* AM *with* PFS *(resp., without* PFS*) and let λ be an* MT*-authenticator. Then $\pi' = C_\lambda(\pi)$ is a SK-secure key-exchange protocol in the* UM *with* PFS *(resp., without* PFS*).*

We remark that the following proof is somewhat more general, and proves that *any* authenticator (not only MT-authenticators) is sufficient for proving the theorem.

Proof: We start by noting that the theorem's statement does not follow directly from the results of [2] (specifically from Theorem 3 in that paper) since there the guarantee for secure transformation between models is proven for the basic UM and AM. Here we need to extend the proof to capture the additional test-session queries that we allow the KE-adversary against the KE protocol. Also worth noting is that our UM and AM are richer than the ones in [2] (e.g. they include session expiration and session-state reveals), however it is easy to see that the proof of Theorem 3 in [2] will work for these adversary activities as well.

Based on these facts we proceed to prove that if protocol π satisfies SK-security (Definition 4) in the AM then protocol $\pi' = C_\lambda(\pi)$ satisfies that definition in the UM. We note that the proof is the same for the cases of SK-security with or without PFS. Consider a protocol π that satisfies Definition 4 in the AM, and let $\mathcal{U}$ be a KE-adversary against π' in the UM. We first observe that π' satisfies Requirement 1 of Definition 4 in the UM with respect to $\mathcal{U}$ (otherwise the global output of running π' in the UM with $\mathcal{U}$ is easily distinguishable from the global output of running π in the AM with *any* AM KE-adversary, in contradiction to the fact that C_λ is an authenticator).

Next we concentrate on demonstrating that π' satisfies Requirement 2 of Definition 4 in the UM. Specifically, given a KE-adversary $\mathcal{U}$ that guesses the bit b in the game of Definition 4 in the UM with probability $1/2 + \epsilon$, we construct a KE-adversary $\mathcal{A}$ that guesses the bit b in the game of

Definition 4 in the AM with probability $1/2 + \epsilon'$, where ϵ' is polynomial in ϵ and in the security parameter.

The construction of $\mathcal{A}$ proceeds in few steps, as follows:

1. Given $\mathcal{U}$, we first construct a regular UM-adversary $\mathcal{U}'$ against π' (i.e., $\mathcal{U}'$ is not allowed to make test-session queries). Adversary $\mathcal{U}'$ runs adversary $\mathcal{U}$ and follows its instructions, with the following exception: When $\mathcal{U}$ chooses a test-session s, $\mathcal{U}'$ queries session s and chooses $b \stackrel{R}{\leftarrow} \{0,1\}$. If $b = 0$ then $\mathcal{U}'$ hands the key of session s to $\mathcal{U}$. If $b = 1$ then $\mathcal{U}'$ hands $\mathcal{U}$ a value drawn from the distribution of session keys. Next, $\mathcal{U}'$ returns to following the instructions of $\mathcal{U}$. When $\mathcal{U}$ halts, $\mathcal{U}'$ outputs the transcript of its interaction with $\mathcal{U}$ and halts.

2. Since $\pi' = C_\lambda(\pi)$, we have that there exists an adversary, $\mathcal{A}'$ in the AM whose output is indistinguishable from the output of $\mathcal{U}'$.

3. Given adversary $\mathcal{A}'$, we construct the KE-adversary $\mathcal{A}$ promised above. Recall that $\mathcal{A}$ interacts in the AM with the game of Definition 4. $\mathcal{A}$ starts by choosing a session s at random out of the sessions initiated by $\mathcal{A}'$. Next, $\mathcal{A}$ follows the instructions of $\mathcal{A}'$; when the chosen session s is established, $\mathcal{A}$ announces s to be its test session. In addition, if $\mathcal{A}'$ queries session s (and session s is not yet exposed) then $\mathcal{A}$ feeds the obtained value for the key of session s to $\mathcal{A}'$. Next $\mathcal{A}$ returns to following the instructions of $\mathcal{A}'$. When $\mathcal{A}'$ halts, $\mathcal{A}$ inspects the output of $\mathcal{A}'$. Recall that the output of $\mathcal{A}'$ mimics the output of $\mathcal{U}'$, which in turn describes a transcript of an execution of $\mathcal{U}$. If in that transcript of $\mathcal{U}$ the test session is session s then $\mathcal{A}$ outputs the bit b' that $\mathcal{U}$ outputs in that transcript. Otherwise, $\mathcal{A}$ outputs a randomly chosen bit.

We analyze the success probability of $\mathcal{A}$ under the assumption that the output of $\mathcal{A}'$ and $\mathcal{U}'$ are identically distributed. Accounting for the fact that the two outputs are only computationally indistinguishable is done in standard ways.

Let ℓ be an upper bound on the number of sessions invoked by $\mathcal{U}$, the advantage (i.e., the probability of success over $1/2$) of $\mathcal{A}$ is $1/\ell$ times its advantage conditioned on the event that the test session chosen by $\mathcal{U}$ (in the output of $\mathcal{A}'$) equals s. For the rest of the analysis we assume that the test session chosen by $\mathcal{U}$ (in the output of $\mathcal{A}'$) equals s.

Let p_b denote the probability that $\mathcal{U}$ outputs 1 when interacting with the game of Definition 4 in the UM, when the value of the "real or random" bit is b. We have that $|p_{\text{real}} - p_{\text{random}}| \geq \epsilon$. Also, when run within $\mathcal{U}'$, $\mathcal{U}$ outputs 1 with probability $(p_{\text{real}} + p_{\text{random}})/2$. Consider the following cases:

1. Assume that $\mathcal{A}$ is given the "real" key of the test session s. In this case, the view of $\mathcal{U}$ (within the output of $\mathcal{A}'$ that is run inside $\mathcal{A}$) is distributed identically to its view when $\mathcal{U}'$ interacts with π' in the UM. In this case $\mathcal{U}$ (and thus also $\mathcal{A}$) outputs 1 with probability $(p_{\text{real}} + p_{\text{random}})/2$.

2. Assume that $\mathcal{A}$ is given the "random" value for the key of the test session s. In this case, the view of $\mathcal{U}$ (within the output of $\mathcal{A}'$ that is run inside $\mathcal{A}$) is distributed identically to its view when interacting in the game of Definition 4 in the UM, conditioned on the event that it is given a "random" value for the key of the test session. In this case $\mathcal{U}$ (and thus also $\mathcal{A}$) outputs 1 with probability p_{random}.

It follows that, when $\mathcal{A}'$ perfectly simulates $\mathcal{U}'$, the advantage of $\mathcal{A}$ is $\epsilon/2\ell$. □

5.1 Two-move Diffie-Hellman in the AM

We demonstrate that under the Decisional Diffie-Hellman (DDH) assumption (see below) the 'classic' two-move Diffie-Hellman key-exchange protocol designed to work against eavesdroppers-only is SK-secure in the AM. We denote this protocol by 2DH and describe it in Figure 2 (here and in the sequel all exponentiations are modulo the defined prime p).

Using Theorem 6 we can apply any authenticator to this protocol to obtain a secure Diffie-Hellman exchange against realistic UM attackers. For illustration, a particular instance of such a SK-secure protocol in the UM, using digital signatures for authentication, is shown in the next section. Other flavors of authenticated DH protocols can be derived in a similar way by using other authenticators (e.g. based on public key encryption or on pre-shared keys [2]); see Section 5.4.

Protocol 2DH

Common information: Primes p, q, $q/p{-}1$, and g of order q in Z_p^*.

Step 1: The initiator, P_i, on input (P_i, P_j, s), chooses $x \stackrel{R}{\leftarrow} Z_q$ and sends $(P_i, s, \alpha = g^x)$ to P_j.

Step 2: Upon receipt of (P_i, s, α) the responder, P_j, chooses $y \stackrel{R}{\leftarrow} Z_q$, sends $(P_j, s, \beta = g^y)$ to P_i, *erases* y, and outputs the session key $\gamma = \alpha^y$ under session-id s.

Step 3: Upon receipt of (P_j, s, β), party P_i computes $\gamma' = \beta^x$, *erases* x, and outputs the session key γ' under session-id s.

Figure 2: The two-move Diffie-Hellman protocol in the AM

The Decisional Diffie-Hellman (DDH) assumption is as follows.

Assumption 7 *Let k be a security parameter. Let p, q be primes, where q is of length k bits and $q/p{-}1$, and g be of order q in Z_p^*. Then the probability distributions of quintuples $Q_0 = \{\langle p, g, g^x, g^y, g^{xy}\rangle : x, y \stackrel{R}{\leftarrow} Z_q\}$ and $Q_1 = \{\langle p, g, g^x, g^y, g^z\rangle : x, y, z \stackrel{R}{\leftarrow} Z_q\}$ are computationally indistinguishable.*

Theorem 8 *Assuming the Decisional Diffie-Hellman (DDH) assumption, protocol* 2DH *is SK-secure in the* AM.

Proof: To see that the first requirement of Definition 4 is satisfied, note that if both P_i and P_j are uncorrupted during the exchange of the key and both complete the protocol (i.e. the three steps of the protocol are completed by P_i and P_j) then they both establish the same key (which is $\gamma = \gamma' = g^{xy} \bmod p$). Note that the session identifier s uniquely binds the values of g^x and g^y to these particular matching sessions and differentiates them from other exponentials that the parties may exchange in other (possibly simultaneous) sessions.

We show that the second requirement of Definition 4 is also satisfied by protocol 2DH. Assume to the contrary that there is a KE-adversary $\mathcal{A}$ in the AM against protocol 2DH that has a non-negligible advantage in guessing correctly whether the response to a test-query is real or random. Out of this attacker $\mathcal{A}$, we construct an algorithm $\mathcal{D}$ that distinguishes between the distributions Q_0 and Q_1 with non-negligible probability; thus reaching a contradiction with Assumption 7. The input to $\mathcal{D}$ is denoted by $(p, g, \alpha^*, \beta^*, \gamma^*)$ and is chosen from Q_0 or Q_1 each with probability 1/2.

Let ℓ be an upper bound on the number of sessions invoked by $\mathcal{A}$ in any interaction. Algorithm $\mathcal{D}$ uses adversary $\mathcal{A}$ as a subroutine and is described in Figure 3.

Distinguisher $\mathcal{D}$

Proceed as follows, on input $(p, g, \alpha^*, \beta^*, \gamma^*)$:

1. Choose $r \stackrel{R}{\leftarrow} \{1 \ldots \ell\}$.
2. Invoke $\mathcal{A}$, on a simulated interaction in the AM with parties $P_1, ..., P_n$ running 2DH. Hand $\mathcal{A}$ the values p, g as the public parameters for the protocol execution.
3. Whenever $\mathcal{A}$ activates a party to establish a new session (except for the r-th session) or to receive a message, follow the instructions of 2DH on behalf of that party. When a session is expired at a player erase the corresponding session key from that player's memory. When a party is corrupted or a session (other than the r-th session) is exposed, hand $\mathcal{A}$ all the information corresponding to that party or session as in a real interaction.
4. When the r-th session, say (P_i, P_j, s), is invoked within P_i to exchange a key with P_j, let P_i send the message (P_i, s, α^*) to P_j.
5. When P_j is invoked to receive (P_i, s, α^*), let P_j send the message (P_j, s, β^*) to P_i.
6. If session (P_i, P_j, s) is chosen by $\mathcal{A}$ as the test-session, then provide $\mathcal{A}$ with γ^* as the answer to this query.
7. If the r-th session (P_i, P_j, s) is ever exposed, or if a session different than the r-th session is chosen as the test-session, or if $\mathcal{A}$ halts without choosing a test-session then $\mathcal{D}$ outputs $b' \stackrel{R}{\leftarrow} \{0, 1\}$ and halts.
8. If $\mathcal{A}$ halts and outputs a bit b', then $\mathcal{D}$ halts and outputs b' too.

Figure 3: Building a distinguisher for DDH

First note that the run of $\mathcal{A}$ by $\mathcal{D}$ (up to the point where $\mathcal{A}$ stops or $\mathcal{D}$ aborts $\mathcal{A}$'s run) is identical to a normal run of $\mathcal{A}$ against protocol 2DH.

Consider the case in which the test session s chosen by $\mathcal{A}$ coincides with the session chosen at random by $\mathcal{D}$ (i.e., the r-th session as chosen in Step 1). In this case, the response to the test-query by $\mathcal{A}$ is γ^*. Thus, if the input to $\mathcal{D}$ came from Q_0 then the response was the actual value of the key exchanged between P_i and P_j during the test-session s (since, by construction, the session key exchanged in Steps 4 and 5 of Figure 3 is $\gamma^* = g^{xy}$). On the other hand, if the input to $\mathcal{D}$ came from Q_1 then the response to the test query was a random exponentiation, i.e. a random value from the distribution of keys generated by the protocol. In addition, the input to $\mathcal{D}$ was chosen with probability $1/2$ from Q_0 and with probability $1/2$ from Q_1 and then the distribution of responses provided by $\mathcal{D}$ to the test query of $\mathcal{A}$ is the same as specified by Definition 4. In this case, the probability that $\mathcal{A}$ guesses correctly whether the test value was "real" or "random" is $1/2 + \varepsilon$ for non-negligible ε. By the above argument this is equivalent to guessing whether the input to the distinguisher $\mathcal{D}$ came from Q_0 or Q_1, respectively. Thus, by outputting the same bit as $\mathcal{A}$ we get that the distinguisher $\mathcal{D}$ guesses correctly the input distribution Q_0 or Q_1 with the same probability $1/2 + \varepsilon$ as $\mathcal{A}$ did.

Now consider the case in which the r-th session is not chosen as a test-session. In this case $\mathcal{D}$ always ends outputting a random bit, and thus its probability to guess correctly the input distribution is 1/2.

Since the first case (in which the test-session and the r-th session coincide) happens with probability $1/\ell$ while the other case happens with probability $1 - 1/\ell$ we get that the overall probability of $\mathcal{D}$ to guess correctly is $1/2 + \varepsilon/\ell$, and thus $\mathcal{D}$ succeeds in distinguishing Q_0 from Q_1 with non-negligible advantage.

□

5.2 SK-secure Diffie-Hellman Protocol in the UM

Here we apply the signature-based authenticator of [2] to the protocol 2DH from Figure 2 to obtain a Diffie-Hellman key-exchange that is SK-secure in the UM. We present the resultant protocol in Figure 4 (it is very similar to a protocol specified in [29]). Its SK-security follows from Theorems 6 and 8.

Protocol SIG-DH

Initial information: Primes p, q, $q/p{-}1$, and g of order q in Z_p^*. Each player has a private key for a signature algorithm SIG, and all have the public verification keys of the other players.

Step 1: The initiator, P_i, on input (P_i, P_j, s), chooses $x \stackrel{R}{\leftarrow} Z_q$ and sends $(P_i, s, \alpha = g^x)$ to P_j.

Step 2: Upon receipt of (P_i, s, α) the responder, P_j, chooses $y \stackrel{R}{\leftarrow} Z_q$, and sends to P_i the message $(P_j, s, \beta = g^y)$ together with its signature $\text{SIG}_j(P_j, s, \beta, \alpha, P_i)$; it also computes the session key $\gamma = \alpha^y$ and *erases* y.

Step 3: Upon receipt of (P_j, s, β) and P_j's signature, party P_i verifies the signature and the correctness of the values included in the signature (such as players identities, session id, the value of exponentials, etc.). If the verification succeeds then P_i sends to P_j the message $(P_i, s, \text{SIG}_i(P_i, s, \alpha, \beta, P_j))$, computes $\gamma' = \beta^x$, *erases* x, and outputs the session key γ' under session-id s.

Step 4: Upon receipt of the triple (P_i, s, sig), P_j verifies P_i's signature sig and the values it includes. If the check succeeds it outputs the session key γ under session-id s.

Figure 4: Diffie-Hellman protocol in the UM: authentication via signatures.

Remarks on protocol SIG-DH. The protocol is the result of applying the signature-based authenticator of [2] to each of the flows in the 2-pass Diffie-Hellman protocol 2DH of Figure 2, and joining (piggy-backing) the common flows. The authenticators use the values α and β (the DH exponentials) as the challenges required by these authenticators. This assumes (as specified in protocol 2DH) that these exponentials are chosen afresh for each new exchange. We remark that this dual use of α and β as exponentials and as challenges is done to simplify the protocol but separate challenges could be sent by the parties and included under the signature. It is worth noting that the identity of the destination party included under the signatures is part of the specification of the signature-based authenticator of [2] and is fundamental for the security of protocol SIG-DH (without them the protocol is insecure; see [19]).

The description of SIG-DH in Figure 4 assumes, as formalized in our model, that the value s of the session-id is provided to the parties. In practice, one usually generates the session identifier s as a pair (s_1, s_2) where s_1 is a value chosen by P_i and different (with very high probability) from all other such values chosen by P_i in his other sessions with P_j. Similarly, s_2 is chosen by P_j with an analogous uniqueness property. These values s_1, s_2 can be exchanged by the parties as a prologue to the above protocol (this may be the case of protocols that implement such a prologue to exchange some other system information and to negotiate exchange parameters; see for example [28]). Alternatively, s_1 can be included by P_i in the first message of SIG-DH, and s_2 be included by P_j in the second message. In any case, it is important for the security of the protocol that these values be included under the parties' signatures.

5.3 A public-key encryption-based protocol without PFS

The protocol described in this section is based on public key encryption schemes secure against chosen ciphertext attacks. We show that this protocol satisfies Definition 5, i.e. SK-security *without* PFS, in the AM. That is, the protocol does not provide forward secrecy of the session keys (an attacker who breaks into a party may compromise all the keys exchanged by this party in the past even if these keys are erased from that party's memory). Formally, we consider that session keys never expire. The protocol can be made into a SK-secure without PFS protocol in the UM by using any authenticator (Theorem 6).

Let (G, E, D) be a key-generation, encryption and decryption algorithms, respectively, of a public-key encryption scheme secure against chosen ciphertext attacks (CCA). (See, for instance, [20, 4, 17]). Let k be the security parameter. Assume that each party P_i has invoked $G(k)$ to get a pair (e_i, d_i) of encryption and decryption keys, and all parties have the public encryption key e_i of the other parties. In addition, let $\{f_\kappa\}_{\kappa\in\{0,1\}^k}$ be a pseudorandom function family (as in [22]). The protocol, denoted ENC, is described in Figure 5.

Protocol ENC

Proceed as follows, given security parameter k.

Step 1: The initiator, P_i, on input (P_i, P_j, s), chooses $\kappa \stackrel{R}{\leftarrow} \{0,1\}^k$ and sends $(P_i, s, E_{e_j}(\kappa))$ to P_j. Next, P_i outputs the session key $\sigma = f_\kappa(P_i, P_j, s)$ under session-id s.

Step 2: Upon receipt of (P_i, s, c) the responder, P_j, computes $\kappa' = D_{d_j}(c)$. If the decryption algorithm does not reject the ciphertext, then P_j outputs the session key $\sigma' = f_{\kappa'}(P_i, P_j, s)$ under session-id s.

Figure 5: A KE protocol based on CCA-secure encryption.

Remark. For ensuring the security of protocol ENC we need to assume that the decryption operation $\kappa' = D_{d_j}(c)$ (including the validity check for the ciphertext c) and the computation of the pseudorandom value $f_{\kappa'}(P_i, P_j, s)$ in Step 2 are performed such that neither the long-term decryption key d_j or the temporary value κ' appear as part of the state of session s. Namely, we need to assume that these operations are done in a separate secure module and only the value $\sigma' = f_{\kappa'}(P_i, P_j, s)$ is returned to the session state. The assumption that long-term private keys are not part of the session state is a fundamental requirement in a model as ours that differentiates session-state corruptions from total corruptions (see Section 2.2). The need to hide κ' from the session state is a specific requirement of the ENC protocol and it is illustrated by the following attack. Say that κ' is returned to the session state, then an attacker could compromise an unexposed session

(P_i, P_j, s) as follows. It corrupts party $P_l, l \neq i, j$, and initiates an ENC session (P_l, P_j, s') between the corrupted P_l and P_j in which P_l sends to P_j the same ciphertext c sent in session s from P_i to P_j. Once P_j decrypts c and stores the temporary value κ' in the state of session s', the attacker performs a session-state reveal and learns κ'. Now it can also compute the value of the session key corresponding to the unexposed session s. Thus, this attack (and the proof of Theorem 9) show the care needed in specifying and implementing the ENC protocol if we require resistance to session-state reveals. Whether this is a realistic risk or not may depend on particular applications and scenarios. In any case, if a separate module for the above operations cannot be assumed then the protocol becomes insecure in our model but is still secure in a model where session states can only be revealed via total corruptions (i.e., a weakened model where session-state reveals are not considered as a separate attacker action).

Theorem 9 *If the encryption (G, E, D) is CCA-secure and the family $\{f_\kappa\}_{\kappa \in \{0,1\}^k}$ is pseudorandom, then protocol* ENC *is SK-secure without* PFS *in the authenticated links model (*AM*).*

Proof: It is easy to see that the first condition of Definition 4 is satisfied by protocol ENC (that is, uncorrupted parties that complete matching sessions output the same session-key). The core of the proof is in proving the second condition of Definition 4 in the case where keys are not expired.

We start by defining a "game" which captures the chosen-ciphertext security of the encryption function E in combination with the pseudorandom family $\{f_\kappa\}_{\kappa \in \{0,1\}^k}$. We will then show that an attacker that breaks the SK-security of protocol ENC can also win in this game and then break the encryption function E. The game is defined in Figure 6.

The encryption game

The parties to the game are $\mathcal{G}$ and $\mathcal{B}$ (for good and bad). $\mathcal{G}$ possesses a pair of public and private keys, e and d (generated via the key generation algorithm G). $\mathcal{B}$ knows e but not d.
The game proceeds in phases:

Phase 0: $\mathcal{G}$ provides $\mathcal{B}$ with a challenge ciphertext $c^* = E_e(\kappa_0)$ for $\kappa_0 \stackrel{R}{\leftarrow} \{0,1\}^k$.

Phase 1: $\mathcal{B}$ sends a pair (c, t) to $\mathcal{G}$ who responds with $f_\kappa(t)$ where $\kappa = D_d(c)$. This is repeated a polynomial (in k) number of times with each pair being chosen adaptively by $\mathcal{B}$ (i.e., after seeing $\mathcal{G}$'s response to previous pairs).

Phase 2: $\mathcal{B}$ sends a test string t^* to $\mathcal{G}$. Then $\mathcal{G}$ chooses a random bit $b \stackrel{R}{\leftarrow} \{0,1\}$. If $b = 0$ then $\mathcal{G}$ responds with $f_{\kappa_0}(t^*)$ where κ_0 is the value encrypted by $\mathcal{G}$ in phase 0. If $b = 1$ then $\mathcal{G}$ responds with a random string r of the same length as $f_{\kappa_0}(t^*)$.

Phase 3: Same as Phase 1.

Phase 4: $\mathcal{B}$ outputs a bit b'.

And the winner is... $\mathcal{B}$ if and only if $b = b'$.

Figure 6: A game that captures the CCA-security of the encryption function E

We state the following Lemma (the proof uses standard arguments and, in particular, is similar to the proof of the encryption-based authenticator from [2]).

Lemma 10 *Assume that the encryption scheme (G, E, D) is CCA-secure and the family $\{f_\kappa\}_{\kappa\in\{0,1\}^k}$ is pseudorandom. Then if the pair (c^*, t^*) is not queried by $\mathcal{B}$ during Phases 1 and 3 the probability that $\mathcal{B}$ wins in the above game is no more than 1/2 plus a negligible fraction.*

We note that $\mathcal{B}$ is not allowed (in the lemma formulation) to query the pair (c^*, t^*) but it is allowed to include, separately, the values c^* and t^* in other pairs.

We now proceed to show that if there is an AM KE-attacker $\mathcal{A}$ that breaks the SK-security of protocol ENC in the sense that it can distinguish between real and random values of a test session while not being allowed to corrupt the partners to this session, then there is an efficient algorithm $\mathcal{B}$ that wins in the above game with non-negligible probability over 1/2.

We build such $\mathcal{B}$. Let $\mathcal{G}$ be the party against which $\mathcal{B}$ plays the game. $\mathcal{G}$ holds a private decryption key d and public encryption key e. The game starts with $\mathcal{G}$ sending a challenge ciphertext c^* to $\mathcal{B}$. Then, $\mathcal{B}$ proceeds to Phase 1 of the game doing the following. It builds a virtual scenario for the run of protocol ENC, and activates the attacker $\mathcal{A}$ against this virtual run. Among all n parties in this run, $\mathcal{B}$ chooses one at random, call it P_j^*. For all other virtual players $\mathcal{B}$ chooses private keys (using the key generation algorithm G) and provides $\mathcal{A}$ with the corresponding public keys. $\mathcal{B}$ does not choose a private key for P_j^*; instead it provides $\mathcal{A}$ with e (the public key of $\mathcal{G}$) as the public key of P_j^*. Also, $\mathcal{B}$ chooses a random session among the sessions where P_j^* is the responder. We denote this session as s^*, and its initiator as P_i^* (i.e. the chosen session is (P_i, P_j, s^*)).

All operations scheduled by $\mathcal{A}$ are performed by $\mathcal{B}$ on behalf of the virtual players in the following way. All session establishments are executed by $\mathcal{B}$ according to the protocol except for the establishment of session s^*. When $\mathcal{A}$ schedules the establishment of session s^* between P_i^* and P_j^*, $\mathcal{B}$ sends the message (P_i^*, s^*, c^*) to P_j^* on behalf of P_i^*. Here c^* is the challenge ciphertext provided to $\mathcal{B}$ by $\mathcal{G}$ in Phase 0.

All exposure of session keys performed by $\mathcal{A}$, via session or party corruptions, that do not involve P_j^* as the responder are answered by $\mathcal{B}$ using his knowledge of private keys. When $\mathcal{A}$ corrupts a party other than P_j^* and P_i^*, then $\mathcal{B}$ also provides $\mathcal{A}$ with the private key of that party. If a session $s \neq s^*$ between a player P and P_j^* in which the latter acts as responder is exposed by $\mathcal{A}$, then $\mathcal{B}$ provides the value of that key to $\mathcal{A}$ in the following way. If P was uncorrupted at the time that s was established then $\mathcal{B}$ was the one to choose the key κ encrypted by P and then it knows it. If P was corrupted then all $\mathcal{B}$ knows is the message (P, s, c) sent from P to P_j^* as step 1 in the protocol. In this case $\mathcal{B}$ presents to $\mathcal{G}$ (as part of Phase 1) the pair (c, t) where $t = (P, P_j^*, s)$. The value σ returned by $\mathcal{G}$ is the value that $\mathcal{B}$ provides to $\mathcal{A}$ as the queried session key (note that by our assumption in the Remark preceding the Theorem the only information exposed in a session-key query or in session-state reveal is the value of the session key so no other information needs to be returned by $\mathcal{B}$ to $\mathcal{A}$).

If at any point $\mathcal{A}$ queries or reveals session s^*, corrupts P_i^* or P_j^*, or chooses a test session different than s^*, $\mathcal{B}$ proceeds as follows. It aborts the run of $\mathcal{A}$ and moves to Phase 2 sending an arbitrary value t^* to $\mathcal{G}$. After getting $\mathcal{G}$'s response it moves directly to Phase 4 outputting a *random* bit b'.

If $\mathcal{A}$ decides to be tested on session s^* then $\mathcal{B}$ moves to Phase 2 and sends to $\mathcal{G}$ the value $t^* = (P_i^*, P_j^*, s^*)$. The response from $\mathcal{G}$ is passed by $\mathcal{B}$ to $\mathcal{A}$ as the value of the key for session s^*.

$\mathcal{B}$ enters Phase 3. It keeps running $\mathcal{A}$ in the same way as described for Phase 1 above (note that in this case $\mathcal{A}$ is not allowed to expose s^*). When $\mathcal{A}$ outputs a bit b' and stops, then $\mathcal{B}$ moves to Phase 4 and outputs the same bit b' as $\mathcal{A}$ did.

We first note that the above behavior of $\mathcal{B}$ in the game is a legal one, namely, that it never asks the pair (c^*, t^*) from $\mathcal{G}$ in phases 1 and 3. This is easy to see since all pairs queried by $\mathcal{B}$ during these phases contain a value t different than t^*. Indeed all the values t queried by $\mathcal{B}$ have the form of a triple (P, Q, s) where P, Q are player identities and s a session identifier. Thus, due to the uniqueness of session-id's the value $t^* = (P_i^*, P_j^*, s^*)$ occurs only with relation to session s^* which is never queried by $\mathcal{B}$ from $\mathcal{G}$ in phases 1 and 3.

Now we prove that $\mathcal{B}$ wins the game against $\mathcal{G}$ with non-negligible advantage. First, note that in the cases where $\mathcal{B}$ aborts the run of $\mathcal{A}$ before completion it outputs a random bit b' so its chances to win in this case is exactly 1/2. In the case where $\mathcal{A}$ ends with output b' the chances of $\mathcal{B}$ to win are exactly the same as those of $\mathcal{A}$ to guess *correctly* whether the test value was real or random. This probability is, by assumption, non-negligible over 1/2. The later case happens whenever the tested session chosen by $\mathcal{A}$ is the same s^* chosen (randomly) by $\mathcal{B}$. Since this event happens with non-negligible probability ($1/\ell$ where ℓ is an upper bound on the number of sessions established in the protocol run) then the overall advantage of $\mathcal{B}$ is non-negligible. □

Remarks on Protocol ENC.

1. The derivation of the session key via a pseudorandom function applied to the session and parties' identifiers is of fundamental importance for the security of the protocol. Had the session key be just κ then the protocol would be insecure (even in the AM!). In this case the attacker sees that P_i sends the ciphertext $E_{e_j}(\kappa)$ to P_j. Then party P_l, that we assume is controlled by the attacker, sends the same ciphertext to P_j. Now, P_j has established the same session-key with two different parties. This a serious security flaw[7] that breaks SK-security: the attacker can now query P_j for the key exchanged with P_l and in this way to learn the key that P_j exchanged with P_i.

2. The actual security of protocol ENC can be improved by specifying that sessions do expire at the initiator (and the corresponding keys removed from its memory). This preserves SK-security and adds considerably to the practical security of the protocol. For example, consider an application where the initiators are mobile devices, vulnerable to the stealing of the private key, communicating with a well-protected gateway. In this case, if we let keys to expire at the initiator, then finding the decryption key of such a mobile device is of no help to the attacker in recovering past (expired) session keys. The attacker must break the gateway to obtain these keys.

3. Another stronger version of this protocol is obtained by letting each party send the other a key as in ENC and deriving the shared session key in a way that requires knowledge of both encrypted keys. In this case, the protocol still does not provide PFS but (if keys are erased from memory when the session is expired) the only way to recover a past key is to find the private keys of *both* initiator and responder. This is the basis to the DH-less mode of SKEME [32]. However, note that our definition of SK-security (with PFS) would reject such a protocol as secure. A weakened version of the definition that is satisfied by the protocol is obtained if one requires that *at most* one of the partners to the modified session can be corrupted (and only after the key is expired at that party).

[7][19] describes an attack in which a dishonest customer exploits a key-exchange weakness to defraud a bank and a honest customer; the same attack can be mount here with P_j acting as the bank, and P_i and P_l acting as the honest and cheating customers, respectively. See [19] for the details.

5.4 Protocols based on shared keys

In order to further illustrate the usability of our methodology we show how to apply it to key-exchange protocols that assume that the two peers initially share a secret key and use this key to authenticate the exchange of new key material. This "key refreshment" functionality is very important in network security protocols (e.g. [32, 28]). We show examples of SK-secure protocols with and without PFS.

These examples use the following MAC-based authenticator that assumes a shared key κ_{ij} between a pair of parties P_i, P_j. Let f denote a secure MAC function, and κ_{ij} be a random key for f chosen under security parameter k. The authenticator is defined as follows: when P_i wants to send a message to a recipient P_j, the latter sends a challenge $r \stackrel{R}{\leftarrow} \{0,1\}^{2k}$ to P_i, and P_i sends the message m together with the authentication tag $f_{\kappa_{ij}}(P_j, r, m)$. The security of this authenticator can be proven in a way similar to the proof of the signature-based authenticator from [2].

Applying this authenticator to the basic two-move Diffie-Hellman protocol 2DH in the AM one obtains (see Section 5.1) an SK-secure Diffie-Hellman protocol (with PFS) in the UM. We omit a detailed description of the resultant protocol and just point out that it is similar to protocol SIG-DH from Section 5.2 where the digital signatures are replaced with the application (by P_i and P_j) of the MAC function keyed under the shared key κ_{ij}.

Protocol REKEY

Initial information: Each pair of players (P_i, P_j) share a secret pseudorandom function $f_{\kappa_{ij}}$.

Protocol REKEY in the AM:

Step 1: The initiator, P_i, on input (P_i, P_j, s), chooses $r_i \stackrel{R}{\leftarrow} \{0,1\}^{2k}$ and sends (P_i, s, r_i) to P_j.

Step 2: Upon receipt of (P_i, s, r_i), the responder P_j chooses $r_j \stackrel{R}{\leftarrow} \{0,1\}^{2k}$ and sends (P_j, s, r_j) to P_i. Then, P_j outputs session key $f_{\kappa_{ij}}(r_i, r_j)$

Step 3: Upon receipt of (P_j, s, r_j), player P_i outputs session key $f_{\kappa_{ij}}(r_i, r_j)$.

Protocol REKEY in the UM:

Step 0: Both players derive two keys from κ_{ij}: $\kappa_1 = f_{\kappa_{ij}}(1)$ and $\kappa_2 = f_{\kappa_{ij}}(2)$.

Step 1: The initiator, P_i, on input (P_i, P_j, s), chooses $r_i \stackrel{R}{\leftarrow} \{0,1\}^{2k}$ and sends (P_i, s, r_i) to P_j.

Step 2: Upon receipt of (P_i, s, r_i), the responder P_j chooses $r_j \stackrel{R}{\leftarrow} \{0,1\}^{2k}$, computes $t_j = f_{\kappa_1}(P_i, r_i, s, r_j)$ and sends (P_j, s, r_j, t_j) to P_i.

Step 3: Upon receipt of (P_j, s, r_j, t_j), player P_i verifies the authentication tag t_j and if successful it computes $t_i = f_{\kappa_1}(P_j, r_j, s, r_i)$, sends (P_i, s, t_i) to P_j, and outputs session key $f_{\kappa_2}(r_i, r_j)$.

Step 4: Upon receipt of (P_i, s, t_i), player P_j verifies the authentication tag t_i and if successful it outputs session key $f_{\kappa_2}(r_i, r_j)$.

Figure 7: Key-refresh protocol based on a shared secret. The protocol in the UM is the result of applying the MAC-based authenticator to the protocol in the AM

We proceed to show yet another example of the application of our modular methodology for designing and proving KE protocols. In this case we show a simple and efficient protocol to derive a fresh session key between players P_i and P_j based on the common ("master") secret key κ_{ij} and without the use of Diffie-Hellman (the protocol does not provide PFS). In Figure 7 we present the protocol in the AM and the protocol in the UM where the latter is derived from the former by applying the above based MAC-based authenticator to each of the protocol's messages and joining (piggy-backing) the common flows. We note that in this case we use a pseudorandom family f rather than a mere MAC for the implementation of the authenticator.

The SK-security (without PFS) of the AM version of REKEY follows directly from the properties of pseudorandom functions. The SK-security (without PFS) of the second protocol in the UM is the result of applying the MAC-based authenticator to the first protocol. Note that we are using f_{κ_2} with the functionality of a pseudorandom function as in the AM protocol, and f_{κ_1} with the functionality of a MAC for the implementation of the MAC-based authenticator. As in the case of protocol SIG-DH, also here we are re-using the strings r_i and r_j both for key derivation and as challenges for the authenticator. We remark that the REKEY protocol in the UM is similar to the AKEP2 protocol from [7].

We end this section by remarking that another interesting use of our results is for analyzing the password-based KE protocols from [27]. It is shown there how to build a *password-based authenticator* which is then used to authenticate a Diffie-Hellman exchange.

6 Applications to Secure Channels

It is common practice to protect end-to-end communications by letting the end parties exchange a secret session key and then use this key to authenticate and encrypt the transmitted data under symmetric cryptographic functions. In order for a key-exchange protocol to be considered secure it needs to guarantee that the above strategy for securing data works correctly, namely, that by using a shared key provided by the KE protocol one achieves sound authentication and secrecy. As it is customary, we will refer to a link between a pair of parties that achieves these properties as a secure channel. While secure channels may have different meanings in different contexts, here we restrict our treatment to the above setting of securing communications using symmetric cryptography with a key derived from a key-exchange protocol[8]. *We prove that an SK-secure key-exchange protocol, appropriately combined with secure MAC and symmetric encryption functions, suffices for realizing such secure channels.*

6.1 A Template Protocol: Network Channels

We start by formalizing a "template protocol" that captures a generic session-oriented KE-based protocol for secure channels between pairs of parties in a multi-party setting with parties $P_1, \ldots, P_n$. This template protocol, called NetChan, simply captures the mechanism by which two parties first share a secret key and then use this key for securing information they exchange. In the template protocol this exchange of information and the security functions applied to it are represented through abstract 'send' and 'receive' functions. Later we will see specific implementations of this template protocol where the generic 'send' and 'receive' primitives are instantiated with actual functions (e.g., for providing authentication and/or encryption). We will also define what it means for such an implementation to be "secure".

Definition of NetChan. A (session-based) network channels protocol, $\mathsf{NetChan}(\pi, \mathsf{snd}, \mathsf{rcv})$, is defined on the basis of a KE protocol π, and two generic functions snd and rcv. (A more general

[8] A somewhat different formalization of secure channels appears in [14] (see Appendix).

treatment can be obtained by considering these functions as interactive protocols but we leave this more general approach beyond the scope of the present paper.) Both snd and rcv are probabilistic functions that take as arguments a session-key (we denote this key as a subscript to the function) and a message m. The functions may also depend on additional state information such as a session-id and partner identifiers, a message identifier, etc. The output of snd is a single value m', while the output of rcv is a pair (v, ok) where ok is a bit and v an arbitrary value. (The bit ok will be used to return a verification value, e.g. the result of verifying an authentication tag.) On the basis of such functions we define $\mathsf{NetChan}(\pi, \mathsf{snd}, \mathsf{rcv})$ in Figure 8.

Protocol $\mathsf{NetChan}(\pi, \mathsf{snd}, \mathsf{rcv})$

$\mathsf{NetChan}(\pi, \mathsf{snd}, \mathsf{rcv})$ is initialized with the same initialization function I of the KE protocol π. It can then be invoked within a party P_i under the following activations:

1. $\mathsf{establish\text{-}session}(P_i, P_j, s, role)$: this triggers a KE-session under π within P_i with partner P_j, session-id s and $role \in \{\mathsf{initiator}, \mathsf{responder}\}$. If the KE-session completes P_i records in its local output "`established session` s `with` P_j" and stores the generated session key.
2. $\mathsf{expire\text{-}session}(P_i, P_j, s)$: P_i marks session (P_i, P_j, s) (if it exists at P_i) as expired and the session key is erased. P_i records in its local output "`session` s `with` P_j `is expired`".
3. $\mathsf{send}(P_i, P_j, s, m)$: P_i checks that session (P_i, P_j, s) has been completed and not expired, if so it computes $m' = \mathsf{snd}_\kappa(m)$, using the corresponding session key κ, sends (P_i, s, m') to P_j, and records "`sent message` m `to` P_j `within session` s" in the local output.
4. On incoming message (P_j, s, m'), P_i checks that the session (P_i, P_j, s) has been completed and not expired, if so it computes $(m, \mathsf{ok}) = \mathsf{rcv}_\kappa(m')$ under the corresponding session key κ. If $\mathsf{ok} = 1$ then P_i records "`received message` m `from` P_j `within session` s." If $\mathsf{ok} = 0$ then no further action is taken.

Figure 8: A generic network channels protocol

We emphasize that the above definition of NetChan applies to either AM or UM adversarial models; indeed, the attacker against NetChan is allowed to initiate and schedule any of the protocol activations and has all the capabilities of an attacker in the corresponding model, including attacks against the key-exchange protocol π (such as party corruptions, session-state reveal, and session-key queries). Also for NetChan we keep our KE convention that session identifiers are checked for uniqueness. Note that NetChan has no local outputs labeled 'secret' (in particular, the session key is not part of the local output of NetChan as these keys are used internally and never passed to another protocol). Thus, the external functionality of NetChan is as in a simple (session-based) message exchange protocol.

6.2 Network Authentication

On the basis of the above formalism, we treat the case of network channels that provide authentication of information over adversary-controlled channels. Namely, we are interested in a NetChan protocol that runs in the unauthenticated-links model UM and yet provides authenticity of transmitted messages. This implementation of NetChan (which we call NetAut) will be aimed at capturing the practice by which communicating parties use a key-exchange protocol to establish a shared session key, and use that key to authenticate (via a message authentication function, MAC) the information exchanged during that session. Namely, if P_i and P_j share a matching session s and P_i wants to send a message m to P_j during that session then P_i transmits m together with $\mathrm{MAC}_\kappa(m)$ where κ is the corresponding session key.

Protocol NetAut . Let π be a KE protocol and let f be a MAC function. Protocol $\mathsf{NetAut}(\pi, f)$ is a network channels protocol $\mathsf{NetChan}(\pi, \mathsf{snd}, \mathsf{rcv})$ as defined in Figure 8, where functions snd and rcv are defined as:

- On input m, $\mathsf{snd}_\kappa(m)$ produces output $m' = (m, f_\kappa(m))$.
- On input m', $\mathsf{rcv}_\kappa(m')$ outputs (v, ok) as follows. If m' is of the form (m,t) then $\mathsf{ok} = 1$ if and only if (i) m is different than all previously seen messages in the session, and (ii) (m,t) passes the verification function of f under key κ. If $\mathsf{ok} = 1$ then set $v = m$, otherwise $\mathsf{ok} = 0$ and $v = null$.

Note that we require the receiver of a message to check for uniqueness of the incoming message. This check is needed to avoid "re-play" or duplication of delivered messages by an attacker. Equivalently, one can think of m as the concatenation of the message with a unique per-message identifier which is computed by the sender and checked for uniqueness at the receiver (e.g., based on a shared counter between the parties). For simplicity and generality, in the above specification of protocol NetAut we abstract out the specific message differentiation mechanism in use.

Our goal is to show that if the key-exchange protocol π is SK-secure and the MAC function f is secure (against chosen-message attacks) then the resultant network channels protocol $\mathsf{NetAut}(\pi, f)$ provides authenticated transmission of information. This requirement can be formulated under the property that "any message recorded by P_i as received from P_j has been necessarily recorded as sent by P_j, except if the pertinent session is exposed". We will actually strengthen this requirement and ask that a network channels protocol provides authentication if it *emulates* (i.e. imitates) the transmission of messages in the *ideally* authenticated-links model AM. Formally, we do so using the notion of protocol emulation and the formalization (see Sections 2.3 and 3.4) of the message transmission protocol (MT) in the AM as done in [2]. Recall that MT is a simple protocol that specifies the functionality of transmitting individual messages in the AM. Here we extend the basic definition of MT to a *session-based* message transmission protocol called SMT. By proving that the network channels protocol NetAut emulates SMT in the UM we get the assurance that transmitting messages over unauthenticated-links using NetAut is as secure as transmitting them in the presence of an attacker that is not allowed to change, duplicate or inject messages over the communication links.

Protocol SMT. We extend protocol MT from [2] to fit our session-based setting in which transmitted messages are grouped into different sessions. We call the extended protocol a session-based message transmission protocol (SMT), and define it in Figure 9. (Note the structural similarity between SMT and NetChan – the differences are that no actual key-exchange is run in SMT, and the functions snd and rcv are instantiated to simple "identity functions".)

Secure network authentication protocols. Since protocol SMT represents a perfectly authenticated exchange of messages, we use it as the specification protocol to define what is meant for an implementation of protocol NetChan to be a secure network authentication protocol (for the definition of the notion of "emulation" used in the following definition see Section 3.4):

Definition 11 *Protocol* $\mathsf{NetChan}(\pi, \mathsf{snd}, \mathsf{rcv})$ *is called a* secure network authentication protocol *if it emulates protocol* SMT *in the* UM.

Protocol SMT

Protocol SMT can be invoked within a party P_i under the following activations:

1. establish-session(P_i, P_j, s): in this case P_i records in its local output "`established session` s `with` P_j".
2. expire-session(P_i, P_j, s): in this case P_i records in its local output "`session` s `with` P_j `is expired`".
3. send(P_i, P_j, s, m): in this case P_i checks that session (P_i, P_j, s) has been established and not expired, if so it sends message m to P_j together with the session-id s (i.e., the values m and s are sent over the ideally-authenticated link between P_i and P_j); P_i records in its local output "`sent message` m `to` P_j `within session` s".
4. On incoming message (m, s) received over its link from P_j, P_i checks that session (P_i, P_j, s) is established and not expired, if so it records in the local output "`received message` m `from` P_j `within session` s".

Figure 9: Protocol SMT in the AM: The specification protocol for authenticated session-based message transmission.

Theorem 12 *If π is a SK-secure key-exchange protocol in the* UM *and f is a MAC function secure against chosen message attacks, then protocol* NetAut(π, f) *is a secure network authentication protocol.*

Proof: In order to show that NetAut(π, f) is a secure network authentication protocol we need to prove that NetAut(π, f) emulates SMT in the UM. Namely, given an attacker $\mathcal{U}$ against NetAut(π, f) in the UM we need to build an AM-attacker, $\mathcal{A}$, against SMT that produces a protocol and adversary output that is indistinguishable from the output produced by the interaction of $\mathcal{U}$ with NetAut(π, f). We define $\mathcal{A}$ to simulate $\mathcal{U}$ as follows. $\mathcal{A}$ builds a virtual "unauthenticated" scenario in which it simulates $\mathcal{U}$ where to each party in SMT corresponds a virtual party in the UM world of NetAut(π, f). We denote by μ the SMT protocol run in the AM by $\mathcal{A}$, and by μ' the simulated virtual protocol NetAut(π, f) in the UM. Also, we denote by $P_1, \ldots, P_n$ the parties running the SMT-protocol μ, and by $P'_1, \ldots, P'_n$ the corresponding virtual parties running μ'.

All the activations by $\mathcal{U}$ (such as invoking KE-sessions, issuing "send" activations, corrupting parties and sessions, etc.) are carried out in the virtual protocol μ' through $\mathcal{A}$. In particular, the action of virtual parties are carried out by $\mathcal{A}$ on their behalf; this includes the running of KE-session within μ' parties and the transmission of messages. The description of $\mathcal{A}$ is presented in Figure 10.

The following facts about the behavior of $\mathcal{A}$ as defined in Figure 10 are easy to inspect.

1. $\mathcal{A}$ is a legal attacker against SMT in AM (in particular, only previously recorded `sent` messages are delivered, except if the sender is corrupted or the session is exposed).
2. The actions of $\mathcal{U}$ are perfectly simulated by $\mathcal{A}$ (i.e., carried identically by $\mathcal{A}$) against μ'.
3. All `sent`, `established` and `expired` events recorded in μ' are equally recorded in μ.
4. `received` events in μ can differ from those in μ' only in the following case (see Step 5 in Figure 10): party P'_i recorded "`received message` m `from` P'_j `within session` s" in μ'

Adversary $\mathcal{A}$

$\mathcal{A}$ proceeds as follows when interacting with $P_1...P_n$ running SMT in the AM.

1. $\mathcal{A}$ initiates a copy of $\mathcal{U}$, interacting with parties $P'_1...P'_n$ running $\mathsf{NetAut}(\pi, f)$ in the UM. In particular, it evaluates the initialization function I of $\mathsf{NetAut}(\pi, f)$ on random input and hands $\mathcal{U}$ the public output of I.
2. Whenever $\mathcal{U}$ activates P'_i with an activation $\mathsf{establish\text{-}session}(P'_i, P'_j, s, role)$ or any activation of P'_i related to the run of a KE-session of protocol π, $\mathcal{A}$ performs the resultant actions within P'_i and hands out the resultant messages to $\mathcal{U}$ for delivery.
3. Whenever $\mathcal{U}$ issues P'_i with one of the $\mathsf{NetAut}(\pi, f)$ activations $\mathsf{send}(P'_i, P'_j, s, m)$, $\mathsf{expire\text{-}session}(P'_i, P'_j, s)$, or with incoming message (P'_j, s, m, t), $\mathcal{A}$ performs the resultant actions of the players in μ' according to the $\mathsf{NetAut}(\pi, f)$ specifications. Every message generated by the parties is transferred by $\mathcal{A}$ to $\mathcal{U}$ for delivery.
4. Whenever P'_i records one of the events "`established session` s `with` P'_j", "`session` s `with` P'_j `is expired`", or "`sent message` m `to` P'_j `within session` s", then in μ attacker $\mathcal{A}$ issues to P_i the activation $\mathsf{establish\text{-}session}(P_i, P_j, s)$, $\mathsf{expire\text{-}session}(P_i, P_j, s)$, $\mathsf{send}(P_i, P_j, s, m)$, respectively.
5. Whenever P'_i records "`received message` m `from` P'_j `within session` s", $\mathcal{A}$ does:
   (a) If P_j recorded "`sent message` m `to` P_i `within session` s" then $\mathcal{A}$ activates P_i with incoming message (m, s) from P_j.
   (b) Else, if P_j is corrupted or session s within P_j is locally exposed, then $\mathcal{A}$ activates P_j with $\mathsf{send}(P_j, P_i, s, m)$ (note that this sent event is not recorded at P_j by the convention that locally exposed sessions do not produce output) and then activates P_i with incoming message (m, s) from P_j.
6. Whenever $\mathcal{U}$ corrupts P'_i, $\mathcal{A}$ hands $\mathcal{U}$ the internal data of the simulated P'_i, and corrupts P_i in the run of μ. Whenever $\mathcal{U}$ issues an exposure action against a session (P'_i, P'_j, s) within P'_i, $\mathcal{A}$ hands $\mathcal{U}$ the corresponding information from the session within P'_i and issues the exposure against the session (P_i, P_j, s) within P_i in μ.
7. When $\mathcal{U}$ halts, $\mathcal{A}$ outputs whatever $\mathcal{U}$ outputs and halts.

Figure 10: Emulation of SMT: the AM-adversary $\mathcal{A}$.

but P'_j did not record the corresponding `sent` event, and neither session s is exposed nor P'_j is corrupted. **We call this case a** forgery-event.

The above facts show that the simulation of $\mathcal{U}$ by $\mathcal{A}$ against the $\mathsf{NetAut}(\pi, f)$ protocol is perfect (i.e. identical to a real run of $\mathcal{U}$) as long as a forgery-event as defined above does not happen. (In the case of a forgery-event the simulation of $\mathcal{U}$ by $\mathcal{A}$ fails since in μ' party P'_i records the received message m while in μ the corresponding party P_i will not record it.) In Lemma 13 below we show that this forgery-event happens with negligible probability (i.e., there is a negligible probability that in an unexposed session of μ', a party P'_i accepts a message from P'_j that the latter did not send). Therefore, we have that the statistical distance between $\mathrm{AUTH}_{\mathrm{SMT},\mathcal{A}}(k)$ and $\mathrm{UNAUTH}_{\mathsf{NetAut}(\pi,f),\mathcal{U}}(k)$ is negligible. Consequently, $\mathsf{NetAut}(\pi, f)$ emulates SMT in the UM and thus it is a secure network authentication protocol. □

Lemma 13 *If π is a SK-secure key-exchange protocol and f is a MAC function secure against chosen message attacks, then for any attacker $\mathcal{U}$ running against* NetAut *the probability of a* "forgery-event" *(as defined in the proof of Theorem 12) during an unexposed session is negligible.*

Proof: We prove the lemma by contradiction: if for a given attacker $\mathcal{U}$ against $\mathsf{NetAut}(\pi, f)$, a forgery-event happens in an unexposed session with non-negligible probability then we build a forger for the MAC function f that succeeds with non-negligible probability. For convenience, we denote the above assumed non-negligible probability of a forgery-event by ε (more precisely, this value is a function of the security parameter).

Building a forger F. The forger F has an oracle to f that uses an unknown random key; F can request from the oracle the value of f on any message under the oracles key and can also request the verification of pairs (m, t) in which case the oracle verifies whether t is the correct value of $f(m)$ under the oracle's key (the latter are called "verification queries"). The goal of F is to produce a MAC forgery, i.e. the value of f on a message under the key of the oracle, without requesting the box to compute this value. F starts by building a virtual NetAut world and activates $\mathcal{U}$ against it (similarly as $\mathcal{A}$ did in the proof of Theorem 12 but without any "SMT parties"). In addition, F chooses a session at random (from all sessions completed during the run of the protocol), say (P_i, P_j, s_0). We will use the identifier s_0 to refer to the chosen session or its matching session. In the cases where $\mathcal{U}$ delivers a message under the session-key of session s_0, F does not use the actual session-key as exchanged in the simulated protocol but instead it requests the oracle to f to provide that value of f (i.e F is effectively using the oracle key as the s_0 session-key). Similarly, F uses the oracle to verify whether messages sent under session s_0 possess the correct value of f (this is needed in cases where $\mathcal{U}$ injects or changes the authentication tags). If during simulation session s_0 is exposed by $\mathcal{U}$, then F aborts its computation (i.e. it fails to forge). If at any point one of the partners to session s_0, say P_i, accepts a message as correctly MACed while the other party did not record the corresponding sent event (in particular, F did not request the MAC of this message from its oracle) then F outputs the message and its MAC as sent to P_i as a forgery against the oracle to the MAC function f. (Note that by the uniqueness property of sent and received messages in NetAut the message on which F outputs this forgery was never queried from the MAC oracle.)

Thus, if in the run of $\mathcal{U}$ by F a forgery-event happens under session s_0 then F succeeds in producing a forgery against the MAC. We want to show that this happens with non-negligible probability.

Recall that we are assuming (by way of contradiction) that in a *regular* run by $\mathcal{U}$ a forgery-event happens with non-negligible probability ε. Thus, if one chooses a session s_0 at random, then in a run of $\mathcal{U}$ a forgery-event will happen in session s_0 with non-negligible probability too (i.e., ε divided by an upper bound on the number of sessions in the protocol). However, the run of $\mathcal{U}$ by the forger *F is not a regular run:* the key used to MAC messages in session s_0 is not the real session key exchanged by the parties but an independent random value. Still we claim that if in a run of $\mathcal{U}$ we replace the session-key in a randomly chosen session s_0 with a random value then the probability of a forgery-event in that session does not change significantly, i.e., it remains non-negligible (and thus F has a non-negligible probability to break the security of the MAC function f).

In order to prove this claim we introduce the following notation. If s is a session completed by some party under a run of $\mathcal{U}$, then we denote by *forgery*(s) the event that a forgery-event happens during session s. We know, by assumption, that if s is chosen at random among all sessions under a regular run of $\mathcal{U}$ then the probability of event *forgery*(s) is non-negligible. We want to prove that this is the case even when $\mathcal{U}$ is run by F. (In this case, this probability, that we denote by $Prob_F(\mathit{forgery}(s))$, is taken over runs of $\mathcal{U}$ in which the real session-key for s is replaced with a random value.) The remainder of the proof is devoted to proving this claim.

The plan for this proof is as follows. Based on $\mathcal{U}$, we build a KE-adversary $\mathcal{U}_{\text{KE}}$ against the KE protocol π. Then we show that if in the modified run of $\mathcal{U}$ as produced by forger F the probability of a forgery-event changes substantially (relative to its probability in a regular run of $\mathcal{U}$), then $\mathcal{U}_{\text{KE}}$ breaks the SK-security of π (i.e., it can choose a test-session in which to distinguish the real value of the session-key from a random independent value.)

Attacker $\mathcal{U}_{\text{KE}}$ runs π with n parties $P_1, \ldots, P_n$ by essentially simulating the actions of $\mathcal{U}$ against a NetAut protocol with protocol π and MAC function f. For this, $\mathcal{U}_{\text{KE}}$ runs $\mathcal{U}$ against a virtual copy of $\mathsf{NetAut}(\pi, f)$, denoted μ', with n players $P'_1, \ldots, P'_n$. Each action decided by $\mathcal{U}$ that concerns the KE protocol part of μ' (such as session establishment, party corruptions, session exposure, etc.) is applied by $\mathcal{U}_{\text{KE}}$ against the real run of π (i.e. against parties $P_1, \ldots, P_n$). Whenever $\mathcal{U}$ orders an action involving the computation of a MAC value by party P'_i using a completed and unexpired session-key (P'_i, P'_j, s), $\mathcal{U}_{\text{KE}}$ checks if it has already learned the value of that key (via a previous session exposure). If not, $\mathcal{U}_{\text{KE}}$ issues a session-key query against (P_i, P_j, s). With the value of the learned session-key, $\mathcal{U}_{\text{KE}}$ computes the required value of f and hands it to $\mathcal{U}$.

There is one exception, however, to the above behavior of $\mathcal{U}_{\text{KE}}$. Among the sessions completed in the run of π, attacker $\mathcal{U}_{\text{KE}}$ chooses one at random as its test-session (e.g., $\mathcal{U}_{\text{KE}}$ chooses at the beginning of its run a number $j \in \{1, \ldots, \ell\}$ where ℓ is an upper bound on the number of sessions created by $\mathcal{U}$ during its run, and then $\mathcal{U}_{\text{KE}}$ chooses the j-th completed session as its test session). If $\mathcal{U}$ ends its run before the test-session is chosen, or if this session happens to be exposed at time of completion (i.e., either a partner to the session is corrupted before completion or $\mathcal{U}$ issued a session-state reveal against this session) then $\mathcal{U}_{\text{KE}}$ stops its run without issuing a test-session query. Otherwise, once the chosen test-session is completed, $\mathcal{U}_{\text{KE}}$ issues a test-session query. We denote the test-session as s_0, and the response to the test query as v (as usual a bit $b \xleftarrow{R} \{\text{REAL}, \text{RANDOM}\}$ is chosen and v is set to the real value of the session-key if $b = \text{REAL}$ and to a random independent value otherwise[9].) Whenever $\mathcal{U}$ evaluates f involving the key of session s_0, $\mathcal{U}_{\text{KE}}$ uses v as the value of the key for f. If at any point $\mathcal{U}$ produces a forgery-event in session s_0 (i.e. $\mathcal{U}$ is able to MAC under key v a message not MACed by $\mathcal{U}_{\text{KE}}$) then $\mathcal{U}_{\text{KE}}$ stops and outputs $b' = \text{REAL}$. If at some point $\mathcal{U}$ stops its run, or if $\mathcal{U}$ orders the exposure of session s_0, then $\mathcal{U}_{\text{KE}}$ stops and outputs $b' \xleftarrow{R} \{\text{REAL}, \text{RANDOM}\}$.

Recall that we want to prove that $Prob_F(forgery(s_0))$ is non-negligible. This is equivalent to proving that the conditional probability

$$\alpha = Prob_{\mathcal{U}_{\text{KE}}}(forgery(s_0) : b = \text{RANDOM})$$

(now taken over the distribution of runs by $\mathcal{U}_{\text{KE}}$ against protocol π) is non-negligible. In order to show this we start by noting that the conditional probability

$$\beta = Prob_{\mathcal{U}_{\text{KE}}}(forgery(s_0) : b = \text{REAL})$$

represents the probability that a forgery-event happens in a regular run of $\mathcal{U}$ (i.e. with all real session-keys used for MAC-ing information) in a randomly selected session s_0. As said earlier, this probability is ε/ℓ (the values ε and ℓ are defined above), and then non-negligible. We end the proof by proving that $\alpha \approx \beta$ (i.e. they differ by only a negligible amount) and thus α is non-negligible. This proof is obtained via the analysis of the probability, denoted $Prob(b' = \text{REAL})$, that $\mathcal{U}_{\text{KE}}$ ends its run with output $b' = \text{REAL}$ (we consider this probability only over runs in which $\mathcal{U}_{\text{KE}}$ issues a

[9]For clarity, we denote bits by REAL and RANDOM rather than 0, 1.

test-session query). We have that $Prob(b' = \text{REAL})$ equals

$$Prob(b' = \text{REAL} : \textit{forgery}(s_0))Prob(\textit{forgery}(s_0)) + Prob(b' = \text{REAL} : \neg\textit{forgery}(s_0))Prob(\neg\textit{forgery}(s_0))$$

By the definition of $\mathcal{U}_{\text{KE}}$, $Prob(b' = \text{REAL} : \textit{forgery}(s_0))$ is always 1 regardless of whether b is REAL or RANDOM. Similarly, $Prob(b' = \text{REAL} : \neg\textit{forgery}(s_0))$ is always $1/2$ regardless of the value of b.

Now consider the case $b = \text{REAL}$; we have that

$$\begin{aligned}
&Prob(b' = \text{REAL} : b = \text{REAL}) = \\
&= 1 \cdot Prob(\textit{forgery}(s_0) : b = \text{REAL}) + 1/2 \cdot Prob(\neg\textit{forgery}(s_0) : b = \text{REAL}) = \\
&= 1 \cdot \beta + 1/2 \cdot (1 - \beta) = 1/2 + \beta/2.
\end{aligned}$$

Similarly, for $b = \text{RANDOM}$ we can obtain

$$Prob(b' = \text{REAL} : b = \text{RANDOM}) = 1/2 + \alpha/2.$$

Since π is a SK-secure KE protocol we know that the difference between $Prob(b' = \text{REAL} : b = \text{REAL})$ and $Prob(b' = \text{REAL} : b = \text{RANDOM})$ is negligible, or otherwise $\mathcal{U}_{\text{KE}}$ would break the security of π. But then we have that $\alpha \approx \beta$ as we had to prove. □

Thus we have completed the proof of Theorem 12 showing that SK-security is a sufficient condition to guarantee the secure composition of key-exchange protocol with a network authentication application. One important aspect of the above proof is that it makes clear the need for allowing the attacker against the key-exchange protocol ($\mathcal{U}_{\text{KE}}$ in our case) to keep running even after the value of the test-session is provided to him (see the remark after Definition 4); indeed, without that capability the theorem is not true.

6.3 Network Encryption

In this section we treat the problem of *secrecy* of communications, and introduce a definition of secrecy in the context of general network channels protocols as defined in Section 6.1. This notion of secrecy is used in the next subsection to formulate our definition of secure channels and to analyze a specific implementation of such channels using SK-secure KE protocols.

Secure network encryption protocols. We start by defining what is meant for a network channels protocol NetChan to be a "secure network encryption protocol". We want to capture the secrecy property that the attacker does not learn information on messages that are exchanged during unexposed sessions (see the "explanation" paragraph following the definition). We follow the indistinguishability approach used to define semantic security of encryption (also used in our definition of SK-security). For this we augment the capabilities of AM and UM attackers that interact with a network channels protocol to include the following action.

> We let the attacker $\mathcal{A}$, running against NetChan, to choose, at some arbitrary point during the interaction, a (single) *test-session* (P_i, P_j, s) among the sessions that are completed, unexpired and unexposed at the time. Also, $\mathcal{A}$ gets to choose a pair of equal-length messages m_0, m_1. Next, a bit $b \xleftarrow{R} \{0,1\}$ is chosen (but <u>not</u> provided to $\mathcal{A}$) and P_i is activated with $\mathsf{send}(P_i, P_j, s, m_b)$. This activation follows the specification of a regular send-activation in the protocol except that when P_i records the sent event in its local output it does *not* write down the value of m_b. Later, if P_j is activated by $\mathcal{A}$ under session s with some incoming message μ and the output of $\mathsf{rcv}_\kappa(\mu)$ (where κ is

the session-key of the test session s) is the pair $(m_b, \mathsf{ok} = 1)$ then the receive-activation is recorded by the recipient but the value m_b is *not* written to the local output. The attacker $\mathcal{A}$ is allowed all the regular adversarial actions except that it is not permitted to expose the test-session (P_i, P_j, s). (However, as in the case of SK-security, the attacker is allowed to corrupt P_i as soon as the test-session expires, and to corrupt P_j as soon as the matching session expires.) At the end of its run, $\mathcal{A}$ outputs a bit b' (as its guess for b).

Definition 14 *We say that a network channels protocol is a* secure network encryption protocol in the UM *if the probability of any* UM*-attacker $\mathcal{A}$ as described above to guess correctly b (i.e., to output $b' = b$) is no more than 1/2 plus a negligible fraction in the security parameter.*

Security of a network encryption protocol is defined in the AM in the same form provided the attacker is a AM-adversary with the above added capability.

Explanation. We clarify the rationale of the above definition. In this definition we want to capture the secrecy property of a network channels protocol, namely, the infeasibility of the attacker to learn information on messages transmitted (usually in encrypted form) between the parties. However, note that in our formalism of network channels the attacker gets to learn the sent and received messages by watching the local output of the parties (recall that whatever is written on the local output becomes immediately available to $\mathcal{A}$); moreover, the attacker even gets to choose the messages in sent-activations. So, how can we say that the attacker does not learn the exchanged messages? For this, we introduce the test messages m_0 and m_1 that the attacker gets to choose but not to learn which one was sent. In particular, in order to hide this information from the attacker we specify that the send and receive activations corresponding to the test message do not record the value of the specific sent or received message. Thus, for a protocol to be secure by our definition it needs to make infeasible for the attacker to guess correctly (i.e., with non-negligible advantage) the sent test-message even though this attacker has access to *all other* messages (in cleartext form) that were sent and received during the protocol.

6.3.1 Discussion

One importat aspect of the above definition is the way we specify the receive-activations (at P_j) in which the test-message m_b is not written to the local output. In order to highlight this issue let's consider first an alternative definition of security of network encryption protocols. Namely, a definition similar to the above definition with the difference that the only receive-activation in which m_b is not written to the local output is an activation where the incoming message is *identical* to the message, call it m^*, handed to $\mathcal{A}$ by P_i as the result of the test send-activation. In the sequel we refer to this variant of the definition as the "strict definition" (of secure network encryption protocols). The reason that we have not adopted this strict definition is that we consider it over-restrictive: for example, this definition call insecure any network encryption protocol that specifies that the message delivered to P_j is different from the exact output produced by P_i. In particular, it would invalidate any protocol that allows for some changes to the transported messages to happen in transit, even though such protocols are common in practice and secure. For example, the AH protocol from [31] allows some well defined parts of the message header, such as number of hops, to be changed in-route by intermediate routers. Other protocols allow for arbitrary or random padding of messages just to comply with some standard length boundary; a change in-route of such padding would not be checked by the receiver nor should such change impact the security of the protocol.

Thus, a better (and more realistic) approach is to permit such possible (inocous) changes to the transported message, and only care about the correctness of the value of the message accepted and recorded by the receiver, namely, the output of the rcv function. This is why, under our definition, we consider *any* incoming message that "decodes" (under rcv) to the test message m_b as related to the test send-activation and then its decoded value is not disclosed to the attacker.

One consequence of this definitional decision is that while Definition 14 does not explicitly mention the need to ensure the uniqueness of messages or the use of message identifiers, it actually requires from a secure network encryption protocol to be careful about the way it guarantees the uniqueness of transmitted messages. To illustrate this point consider the following strategy for attacker $\mathcal{A}$. After activating P_i with the test send-activation with messages m_0 and m_1, $\mathcal{A}$ activates P_i with another send-activation with m_0 as the input message. Now, $\mathcal{A}$ delivers to P_j the message resultant from the later send-activation. If P_j does not write the decoded message to its local output then $\mathcal{A}$ learns that $m_b = m_0$, if P_j does write the message then $\mathcal{A}$ learns that $m_b = m_1$. Thus a secure network encryption protocol must make this attack impossible for $\mathcal{A}$ and, in particular, it must ensure the uniqueness of sent messages. This can be achieved by the use of unique message identifiers that become part of the sent messages. We exemplify this mechanism in our realization of secure channels in Section 6.4.

Another remark concerning the "strict definition" discussed above is that it naturally corresponds to the way security of encryption functions against chosen ciphertext attacks (CCA) is usually defined. (That is, the definition of CCA security allows an attacker to query a decryption oracle with any input ciphertext except for the one in which the attacker is being tested.) While this correspondence can be seen as an advantage of the "strict definition" it actually points to an important issue here: CCA-security is not a necessary notion when formalizing security of network channels. Indeed, the CCA formulation actually carries the same drawbacks as discussed before for the strict definition. A further illustration of these issues can be found in the remark after the proof of Theorem 16.

On the correctness requirement. Notice that Definition 14 does not make any "correctness" requirements from the encryption protocol. That is, it is not required that the recipient will output the same message as recorded by the sender. While this is a natural requirement for a network encryption protocol (we want decrypted messages to correspond to the plaintext originally encrypted) we omit it from our definition since our use of network encryption (for defining and realizing secure channels – see Definition 15) appears only in conjunction with a network authentication protocol, and the latter already guarantees this correctness property. If one is interested in a stand-alone use of the notion of a network encryption protocol then adding this correctness requirement to the above definition is straightforward.

6.4 Secure Channels

We are now ready to define what is meant by a "secure channels" protocol.

Definition 15 *A network channels protocol in the* UM *is called a* secure network channels protocol *if it is a secure network encryption protocol and also a secure network authentication protocol.*

We proceed to show that the network channels protocol, denoted NetSec and defined below, that applies encryption to transmitted messages and applies a secure MAC function to the resultant ciphertext is a secure channels protocol. In the description of this protocol we assume *explicit* message identifiers that are part of the sent messages and make all these messages necessarily different. Specifically, the input to a send activation is a pair $m = (m\text{-}id, \bar{m})$ where m is chosen by the attacker at will but $m\text{-}id$ is an identifier that is *independent* from the message (can think of it as a message counter) and is *different* from all message identifiers used in other send activations in the same session. In actual implementations of the protocol this unique $m\text{-}id$ value needs to be chosen by the sender and checked for uniqueness at the receiver; here we represent it as part of the input to the send activation in order to be consistent with our general formalism of network channels from Figure 8 and to avoid the specification of a particular message-id mechanism. This uniqueness of message identifiers is assumed only for uncorrupted sessions.

The secure channels protocol NetSec. Let π be a KE protocol, f a MAC function, $\textsc{Enc}$ a symmetric encryption function, and F a family of pseudorandom functions. We denote by $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ the network channels protocol $\mathsf{NetChan}(\pi, \mathsf{snd}, \mathsf{rcv})$, as defined in Figure 8, that uses the snd and rcv functions defined as:

- On input $m = (m\text{-}id, \bar{m})$, $\mathsf{snd}_\kappa(m)$ produces output $m' = (m\text{-}id, c, t)$ where $c = \textsc{Enc}_{\kappa_e}(\bar{m})$ and $t = f_{\kappa_a}(m\text{-}id, c)$. The keys κ_e and κ_a are computed as $F_\kappa(0)$ and $F_\kappa(1)$, respectively[10].
- On input m', $\mathsf{rcv}_\kappa(m')$ outputs (m, ok) as follows. If m' is of the form $(m\text{-}id, c, t)$ then $\mathsf{ok} = 1$ if and only if (i) $m\text{-}id$ is different than all previously seen message identifiers in the session, and (ii) $(m\text{-}id, c, t)$ passes the verification function of f under key κ_a. If $\mathsf{ok} = 1$ then set $\bar{m} = \textsc{Enc}^{-1}_{\kappa_e}(c)$ and $m = (m\text{-}id, \bar{m})$, otherwise $\mathsf{ok} = 0$ and $m = null$. The keys κ_e and κ_a are defined as above.

That is, function snd applies an encryption on the message and a MAC to the ciphertext where these functions use "computationally independent" keys derived from the session κ via a pseudorandom function. The function rcv does the decryption but only after verifying that the authentication of the ciphertext is correct.

Note 1: We stress again our assumption that message identifiers are different for each sent message in a session (and checked for uniqueness at the recipient). In particular, this means that an implementation of the message id mechanism needs to make sure that the two parties of the session (while uncorrupted) choose different identifiers for each new message. This can achieved, for example, if each party chooses the values $m\text{-}id$ from disjoint sets (e.g., P_i sets the first bit of its identifiers to 0 and P_j to 1). Actual protocols can also specify the use of "directional" keys, i.e., the keys used for the snd function from P_i to P_j are different (and computationally independent) from the keys used from P_j to P_i; in this case message identifiers need only be unique per direction. Clearly, these multiple keys can be derived from the session key κ using a pseudorandom function.

Note 2: Message identifiers are not protected for secrecy. Since they are chosen independently of the sent message this does not compromise the secrecy of the message. In particular, when analyzing the above protocol as a network encryption protocol, we assume that the test messages m_0 and m_1 from Definition 14 have the same message identifier so its exposure provides no information to the attacker about which message m_b was actually sent.

In the following theorem we use the notion of a symmetric encryption function that is secure against chosen-plaintext attacks. For a formalization of this notion see [3] (see also the CPA game in Figure 11).

[10] For simplicity we assume the encryption and authentication functions to use uniformly distributed keys of the same length; other cases can be handled via standard key derivation methods (e.g., truncating the output of F, iterating F to produce longer outputs, etc.).

Theorem 16 *If π is a SK-secure key-exchange protocol in* UM*, f is a MAC function secure against chosen-message attacks,* Enc *a symmetric encryption function secure against chosen-plaintext attacks, and F a secure family of pseudorandom functions, then* $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ *is a secure channels protocol in the* UM.

Proof: We can assume, for simplicity, that keys κ_e and κ_a in $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ are direct outputs of protocol π (and then indistinguishable from uniformly and independently chosen keys). Accounting for the fact that we actually derive them from a single session key κ via a pseudorandom function can be done using standard arguments.

The proof that $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ is a secure network authentication protocol follows from Theorem 12 with one modification: here we are not applying the MAC function directly to the plaintext but on the ciphertext. Since by property of the encryption function we have that a ciphertext decrypts to a unique plaintext under key κ_e then the authentication of the ciphertext implies the authentication of the plaintext message. (Formally, one can consider a modification of protocol NetAut in Theorem 12 where the function snd is defined to first encrypt the message and then authenticate the ciphertext under the MAC function; the output of snd is the concatenation of the computed ciphertext and MAC tag. Similarly, rcv first checks the MAC on the ciphertext, and if successful it decrypts the ciphertext and outputs the plaintext message.)

The rest of the proof is devoted to proving that $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ is a secure network encryption protocol. The plan for the proof and many of the details are similar to the proof of the network authentication theorem (Theorem 12, and more specifically of Lemma 13). We thus sketch the most important aspects of the current proof but omit the details that are easy to complete following the network authentication case. Our goal here is to prove the theorem by way of contradiction, namely, given an attacker $\mathcal{A}$ that breaks the security of $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ as a network encryption protocol then we can build an attacker $\mathcal{B}$ that breaks the security of the symmetric encryption function Enc against chosen-plaintext attacks.

The CPA symmetric encryption game

The game is played by an attacker $\mathcal{B}$ with access to an encryption oracle E. On input m, the oracle returns the encryption of m under function Enc using a secret key κ not provided to $\mathcal{B}$. The game proceeds in phases:

Phase 1: $\mathcal{B}$ queries E with any messages of its choice. At any point $\mathcal{B}$ may choose to move to phase 2.

Phase 2: $\mathcal{B}$ chooses two equal-length messages m_0 and m_1; a bit b is chosen at random and the value $c^* = E(m_b)$ is returned to $\mathcal{B}$. (The value of b is not provided to $\mathcal{B}$.)

Phase 3: Same as Phase 1.

Phase 4: $\mathcal{B}$ outputs a bit b'.

And the winner is... $\mathcal{B}$ if and only if $b = b'$.

Figure 11: CPA-security of the symmetric encryption function Enc

In order to capture the CPA-security of Enc (i.e., its security against chosen-plaintext attacks) we consider the game described in Figure 11. By the assumption that Enc is semantically secure against chosen-plaintext attacks we have that no polynomial-time attacker $\mathcal{B}$ can win this game with non-negligible advantage (where "advantage" means the winning probability minus 1/2). However, we show next how to construct such attacker $\mathcal{B}$ given an attacker $\mathcal{A}$ that breaks the security of $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ as a network encryption protocol, i.e. an attacker $\mathcal{A}$ that wins the test of Definition 14 against $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ with non-negligible advantage. This proves that such an attacker $\mathcal{A}$ does not exist and then $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ is a secure network encryption protocol. (A precise quantified relation between the success probability of $\mathcal{B}$ and $\mathcal{A}$ can be easily derived from the proof arguments below.)

Building $\mathcal{B}$ given $\mathcal{A}$. Phase 1. $\mathcal{B}$ starts by building a virtual $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ world (including the choice of initial information for the parties) and activates $\mathcal{A}$ against it. In addition, $\mathcal{B}$ chooses a session at random (from all sessions completed during the virtual run of $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$) under $\mathcal{A}$, say (P_i, P_j, s_0). We will use the identifier s_0 to refer to the chosen session or its matching session. All actions by $\mathcal{A}$ (activations or corruptions) that do not involve session s_0 are carried by $\mathcal{B}$ using the specification of $\mathsf{NetSec}(\pi, f, \textsc{Enc}, F)$ and based on the full knowledge that $\mathcal{B}$ has of the information held by the parties in the protocol. If at any point session s_0 is exposed by $\mathcal{A}$ (this may happen as long as the session is unexpired at P_i or P_j) then $\mathcal{B}$ outputs a random bit b' and stops. When $\mathcal{A}$ activates the establishment of the KE session s_0 between P_i and P_j, $\mathcal{B}$ activates P_i and P_j with the normal operations for session-key establishment as in protocol π. When $\mathcal{A}$ activates P_i or P_j with a send-activation under session s_0 and input message $m = (m\text{-}id, \bar{m})$, $\mathcal{B}$ does *not* use the actual key shared in session s_0 to compute the outgoing message m'. Instead, $\mathcal{B}$ computes $m' = (m\text{-}id, c, t)$ where $c = E(\bar{m})$ (i.e., $\mathcal{B}$ uses the encryption oracle for the encryption of messages under session s_0) and $t = f_{\kappa_a}(m\text{-}id, c)$ where κ_a is a key that $\mathcal{B}$ chooses independently and at random for use as the MAC key during session s_0.
Receive-activations under session s_0 are handled by $\mathcal{B}$ (during Phase 1) as follows. Say P_j is activated with incoming message $m' = (m\text{-}id, c, t)$ under session s_0, then $\mathcal{B}$ checks $m\text{-}id$ for validity and if valid it uses its knowledge of κ_a to verify the authentication tag t. If any of these verifications fail then P_j sets $\mathsf{ok} = 0$ and $m = null$. If the verification is successful, in particular the triple $(m\text{-}id, c, t)$ passes the verification of f_{κ_a}, then (except for a negligible probability of forgery against the MAC) the pair $(m\text{-}id, c)$ was included in the output of a previous send-activation under session s_0 in which case $\mathcal{B}$ already knows the plaintext encrypted under ciphertext c and can record the reception of the message in P_j's local input.
If at any point $\mathcal{A}$ chooses a test session (according to Definition 14) different than s_0 then B outputs a random bit b' and stops. If s_0 is chosen by $\mathcal{A}$ as the test session and messages m_0 and m_1 are provided by $\mathcal{A}$ then $\mathcal{B}$ moves to phase 2.
Phase 2. Since we assume message-identifiers that are independent from the message then we have that the test messages m_0 and m_1 chosen by $\mathcal{A}$ have the same message identifier, which we denote by $m\text{-}id^*$. Namely, $m_0 = (m\text{-}id^*, \bar{m}_0)$ and $m_1 = (m\text{-}id^*, \bar{m}_1)$. Now, $\mathcal{B}$ uses the messages $\bar{m}_0$ and $\bar{m}_1$ as its own test messages to oracle E in the CPA game. Let c^* be the value returned to $\mathcal{B}$ as the oracle response to this test. $\mathcal{B}$ then hands to $\mathcal{A}$ the triple $(m\text{-}id^*, c^*, t^*)$ where $t^* = f_{\kappa_a}(m\text{-}id^*, c^*)$. As specified in Definition 14, P_i does not record the actual value of $\bar{m}_b$ (which $\mathcal{B}$ does not know anyway). Now, $\mathcal{B}$ moves to phase 3.
Phase 3. The actions of $\mathcal{B}$ in Phase 3 are similar to Phase 1 except that now $\mathcal{A}$ may activate P_j with incoming message containing $(m\text{-}id^*, c^*)$ for which $\mathcal{B}$ does not know its decryption. In this case, $\mathcal{B}$ first checks the validity of the authentication tag in the incoming message. If it fails then no action is needed. If it is successful then P_j records the reception of the message in its local output but without specifying the decrypted message since this message is m_b which, by specification of

Definition 14, P_j does not write to its local output (thus, $\mathcal{B}$ does not need to know m_b).
Phase 4. Whenever $\mathcal{A}$ stops its run with output b', $\mathcal{B}$ moves to phase 4 and stops with the same output b' as $\mathcal{A}$.

Analysis of $\mathcal{B}$. It is easy to verify that $\mathcal{B}$ as defined above is a legal attacker in the CPA game of Figure 11. We need to show that $\mathcal{B}$ has non-negligible advantage in winning that game. Proving this is similar to the analysis of the success probability of forger F in the proof of Lemma 13. First, we claim that the event in which the session s_0 chosen at random by $\mathcal{B}$ is also the test session chosen by $\mathcal{A}$ has non-negligible probability to occur (simply because there are only polynomially many sessions). Second, we note that if s_0 is chosen by $\mathcal{A}$ as the test session and $\mathcal{B}$ carries the actions of $\mathcal{A}$ related to session s_0 using the actual session key exchanged by P_i and P_j in that session then the advantage of $\mathcal{B}$ to guess b correctly is the same as for $\mathcal{A}$ to guess correctly (since in this case the simulation of $\mathcal{A}$ by $\mathcal{B}$ is perfect) and then non-negligible. So, the main argument is to show that replacing the actual session keys (for authentication and for encryption) from session s_0 with the random independent key κ_a chosen by $\mathcal{B}$ for the MAC and the random independet key used by oracle E for its encryptions does not significantly change the odds of $\mathcal{A}$ to win. This fact follows from the SK-security of the KE protocol π and its proof is similar to the proof of the analogous fact in Lemma 13 (with $\mathcal{B}$ and $\mathcal{A}$ taking the roles of F and $\mathcal{U}$, respectively).

□

On the (non) necessity of CCA-security. The above Theorem shows that security against chosen-plaintext attacks (CPA) is all we need to require from the function ENC in order to implement secure channels. This is an important property since most symmetric encryption functions and modes used in practice are CPA-secure but not secure against chosen-ciphertext attacks (CCA). Also worth noting is that even the combination of the MAC function f on top of ENC does not necessarily result in a CCA-secure function (namely, the function snd defined under NetSec(π, f, ENC, F) is not necessarily CCA-secure when considered as an encryption function with keys κ_e and κ_a). To see this consider a MAC function with the property that flipping the least significant bit of an authentication tag does not change the validity of the tag. In this case the resultant composed function snd is not CCA-secure while it suffices (by virtue of the above theorem) for implementing secure channels.

This example also helps to emphasize the over-restrictive character of the "strict definition" of a secure network encryption protocol as discussed in Section 6.3.1. Indeed, it is easy to see that in order for protocol NetSec to satisfy this strict definition one has to make sure that the snd function in protocol NetSec is CCA-secure. In particular, the above example shows that the assumption that f is a secure MAC function is not enough to prove the network encryption security of NetSec(π, f, ENC, F) under the strict definition. Such a definition would require a stronger notion of a MAC where in addition to the regular unforgeability requirements one requires that the attacker cannot change a given valid pair (m, t) (where m is a message and t a valid authentication tag) into another valid pair (m, t') with $t \neq t'$. When inspecting the NetSec protocol one can easily see that this extra requirement from the MAC function is not a real security necessity but just the artificial result of the unnecessarily restrictive nature of the strict definition.

The order of encryption and authentication. Recent results in [33] show that if the encryption function is assumed to be secure against chosen-plaintext attacks (as in the above Theorem) then the ordering of first applying the encryption function and then the authentication function (as in NetSec(π, f, ENC, F)) is instrumental for guaranteeing secure channels. It is shown in [33] that other common orderings of the functions (in which authentication is applied directly to the plaintext) cannot ensure secure channels even if the key-exchange protocol in use is (ideally) secure.

References

[1] D. Beaver, "Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority", J. Cryptology (1991) 4: 75-122.

[2] M. Bellare, R. Canetti and H. Krawczyk, "A modular approach to the design and analysis of authentication and key-exchange protocols", *30th STOC*, 1998.

[3] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation", *Proceedings of the 38th Symposium on Foundations of Computer Science*, IEEE, 1997.

[4] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations Among Notions of Security for Public-Key Encryption Schemes", *Advances in Cryptology - CRYPTO'98 Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk, ed., Springer-Verlag, 1998, pp. 26–45.

[5] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm", *Advances in Cryptology - ASIACRYPT'00 Proceedings*, Lecture Notes in Computer Science Vol. xxxx, T. Okamoto, ed., Springer-Verlag, 2000.

[6] M. Bellare, E. Petrank, C. Rackoff and P. Rogaway, "Authenticated key exchange in the public key model," manuscript 1995–96.

[7] M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Advances in Cryptology, - CRYPTO'93*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed, Springer-Verlag, 1994, pp. 232-249.

[8] M. Bellare and P. Rogaway, "Provably secure session key distribution– the three party case," *Annual Symposium on the Theory of Computing (STOC)*, 1995.

[9] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung, "Systematic design of two-party authentication protocols," *IEEE Journal on Selected Areas in Communications* (special issue on Secure Communications), 11(5):679–693, June 1993. (Preliminary version: Crypto'91.)

[10] S. Blake-Wilson, D. Johnson and A. Menezes, "Key exchange protocols and their security analysis," *Proceedings of the sixth IMA International Conference on Cryptography and Coding*, 1997.

[11] S. Blake-Wilson and A. Menezes, "Entity authentication and key transport protocols employing asymmetric techniques", *Security Protocols Workshop*, 1997.

[12] M. Burrows, M. Abadi and R. Needham, "A logic for authentication," DEC Systems Research Center Technical Report 39, February 1990. Earlier versions in *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, 1988, and *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, 1989.

[13] R. Canetti, "Security and Composition of Multiparty Cryptographic Protocols", *Journal of Cryptology*, Vol. 13, No. 1, 2000.

[14] R. Canetti, "A unified framework for analyzing security of Protocols", manuscript, 2000. Available at http://eprint.iacr.org/2000/067.

[15] R. Canetti, S. Halevi and A. Herzberg, "How to Maintain Authenticated Communication", *Journal of Cryptology*, Winter 2000. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC)*, ACM, 1997, pp. 15-25.

[16] R. Canetti and H. Krawczyk, "Proving secure composition of key-exchange protocols with any application", in preparation.

[17] R. Cramer and V. Shoup, "A practical public-key cryptosystem provably secure against adaptive chosen ciphertext attack", *Advances in Cryptology - CRYPTO'98 Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk, ed., Springer-Verlag, 1998.

[18] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Info. Theory* IT-22, November 1976, pp. 644–654.

[19] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, 2, 1992, pp. 107–125.

[20] D. Dolev, C. Dwork and M. Naor, Non-malleable cryptography, SICOMP, to appear. Preliminary version in *23rd STOC*, 1991.

[21] O. Goldreich, *"Foundations of Cryptography (Fragments of a book)"*, Weizmann Inst. of Science, 1995. (Available at `http://philby.ucsd.edu/cryptolib.html`)

[22] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM*, Vol. 33, No. 4, 210–217, (1986).

[23] S. Goldwasser, and L. Levin, "Fair Computation of General Functions in Presence of Immoral Majority", *CRYPTO '90, LNCS 537*, Springer-Verlag, 1990.

[24] S. Goldwasser and S. Micali, Probabilistic encryption, *JCSS*, Vol. 28, No 2, April 1984, pp. 270-299.

[25] S. Goldwasser, S. Micali and C. Rackoff, "The Knowledge Complexity of Interactive Proof Systems", *SIAM Journal on Comput.*, Vol. 18, No. 1, 1989, pp. 186-208.

[26] C.G. Günther, "An identity-based key-exchange protocol", *Advances in Cryptology - EUROCRYPT'89*, Lecture Notes in Computer Science Vol. 434, Springer-Verlag, 1990, pp. 29-37.

[27] S. Halevi, and H. Krawczyk, "Public-Key Cryptography and Password Protocols", *ACM Transactions on Information and System Security*, Vol. 2, No. 3, August 1999, pp. 230–268.

[28] D. Harkins and D. Carrel, ed., "The Internet Key Exchange (IKE)", *RFC 2409*, November 1998.

[29] ISO/IEC IS 9798-3, "Entity authentication mechanisms — Part 3: Entity authentication using asymmetric techniques", 1993.

[30] J. Katz and M. Yung, "Complete characterization of security notions for probabilistic private-key encryption", *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 2000.

[31] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", *Request for Comments 2401*, Nov. 1998.

[32] H. Krawczyk, "SKEME: A Versatile Secure Key Exchange Mechanism for Internet,", *Proceedings of the 1996 Internet Society Symposium on Network and Distributed System Security,* Feb. 1996, pp. 114-127.

[33] H. Krawczyk, "The order of encryption and authentication for protecting communications (Or: how secure is SSL?)", to appear Crypto 2001.

[34] C. H. Lim and P.J. Lee, "A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroup", *Advances in Cryptology – CRYPTO 97 Proceedings*, Lecture Notes in Computer Science, Springer-Verlag Vol. 1294, B. Kaliski, ed, 1997, pp. 249–263.

[35] P. Lincoln, J. Mitchell, M. Mitchell, A. Schedrov, "A Probabilistic Poly-time Framework for Protocol Analysis", *5th ACM Conf. on Computer and System Security*, 1998.

[36] S. Lucks, "Open key exchange: How to defeat dictionary attacks without encrypting public keys," *Proceedings of the 1997 Security Protocols Workshop*, 1997.

[37] A. Menezes, P. Van Oorschot and S. Vanstone, "Handbook of Applied Cryptography," CRC Press, 1996.

[38] S. Micali and P. Rogaway, "Secure Computation", unpublished manuscript, 1992. Preliminary version in *CRYPTO 91.*

[39] R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, Vol. 21, No. 12, December 1978, pp. 993–999.

[40] B. Pfitzmann, M. Schunter and M. Waidner, "Secure Reactive Systems", IBM Research Report RZ 3206 (#93252), IBM Research, Zurich, May 2000.

[41] B. Pfitzmann and M. Waidner, "A General Framework for Formal Notions of 'Secure' System", Hildesheimer Informatik-Berichte 11/94 Institut fr Informatik, Universitt Hildesheim, April 1994.

[42] B. Pfitzmann and M. Waidner, "A model for asynchronous reactive systems and its application to secure message transmission", IEEE Security and Privacy Symposium, May 2001. Earlier version available in IBM Research Report RZ 3304 (#93350), IBM Research, Zurich, December 2000.

[43] V. Shoup, "On Formal Models for Secure Key Exchange", Theory of Cryptography Library, 1999. Available at: http://philby.ucsd.edu/cryptolib/1999/99-12.html.

[44] V. Shoup and A. Rubin. "Session key distribution using smart cards", *Advances in Cryptology – EUROCRYPT 96 Proceedings*, Lecture Notes in Computer Science Vol. 1070, Springer-Verlag, U. Maurer, ed, 1995.

A More on Related Work

We provide some more details on several definitional works on KE that are closely related to the present work.

On the work of Bellare and Rogaway [7, 8]. The first complexity-based formalization of secure KE protocols (i.e., the first definitions that take into consideration the computational limitations of an adversary and allow for an analysis that considers non-idealized cryptographic primitives) was presented by Bellare and Rogaway in [7, 8], in the context of shared long-term keys. These works postulate an adversary in charge of all communications, and explicitly model concurrent sessions by creating a model where the adversary is surrounded by "oracles" that represent sessions within parties. In such a model, querying an oracle represents delivery of a message or the corruption of a session. Their method of defining security is based on the method used for defining semantic security of encryption functions [24]: the adversary should be unable to distinguish, with non-negligible probability, between the key of a chosen session and an independent random value. They prove the security of specific authentication and key exchange protocols under these definitions. Various works extend the [7, 8] framework to other settings and problems; for example, Shoup and Rubin to smart card settings [44]; Lucks to consider dictionary attacks [36]; Blake-Wilson, Johnson and Menezes [10, 11] for the public key setting.

The original formalization of [7, 8] was later demonstrated to have a security flaw, by Rackoff (personal communication, 1995). In an unpublished work, Bellare, Petrank, Rackoff and Rogaway [6] proposed a fix for this flaw. Our definition of security (Section 4) follows essentially that fixed version of the [7, 8] definition, but cast in the protocol and adversary framework used here. Next, we sketch the Rackoff attack which is instructive for pointing out to the subtleties involved in the formalization of security for key-exchange protocols.

In the definitions of [7, 8], the adversary points to an unexposed session of its choice, and receives a value k_b, where k_0 is the real session key of this session, k_1 is an independently chosen random value, and b is a randomly chosen bit that is unknown to the adversary. The security requirement is that the adversary is unable to predict b with non-negligible advantage over one half. The original version of these definitions requires that the adversary outputs its guess for b immediately after it obtains the test value. Rackoff has noticed that this requirement is not strong enough: Consider your favorite secure key-exchange protocol π. Now, add to the specifications of the protocol the following instruction for the party that completes first the session establishment according to protocol π: if at any point this party receives a message with the value $\mathrm{MAC}_\kappa(0)$, where MAC is a secure message authentication function and κ the established session-key, then the party publicizes (say via a further message in the protocol) the value of κ. However, the protocol never instructs any party to carry out such an instruction. As a result the protocol can be shown to pass the weakened definition. On the other hand, it is clear that such a protocol cannot be composed securely with an authentication application that uses the session key for MAC-ing information (since such an application could produce the value $\mathrm{MAC}_\kappa(0)$ that can be used to expose κ).

The fix to the definition, proposed by [6], is to let the adversary to continue interacting with the protocol even after the test value is received and before the guess is made. We stress that, although no attacks against the fixed definition were known, up till now it was never demonstrated that this definition (or any other) is "sufficiently strong" for guaranteeing the security of the common applications that use key exchange.

On the work of Bellare, Canetti and Krawczyk [2]. A somewhat different approach to defining secure KE protocols is taken in the work of Bellare, Canetti and Krawczyk [2]. First they specify an adversarial model (called the unauthenticated-links model (UM)) that represents the capabilities of the adversary in real-life networks. (As in [7, 8], this model also postulates independent sessions and adversarially controlled communication. However it is different in that it directly represents a communication network and accounts in a natural way to the fact that other protocols can be running in the same system.) In this model they formalize the notion of authenticators, i.e., "compilers" that transform protocols that assume ideally-authenticated communication into "equivalent" protocols in the UM. (Authenticators are also formalized, in a different context, in [15].) In our work we borrow from [2] the above protocol and adversarial models, and demonstrate the usefulness of the authenticators notion for designing and analyzing protocols.

In addition to the above basic models, [2] also treat the issue of security of KE protocols. For this they formulate an "ideal KE process" that is meant to capture the expected properties of a KE protocol, and require that a secure KE protocol will "emulate" the ideal process. Their notion of emulation is influenced by general definitions of security of multi-party protocols [23, 38, 1, 13]. They also consider the use of KE for maintaining authenticated communication. In particular, they claim that the standard method of combining a KE protocol with a shared-key message authentication code (MACs) results in a secure authenticator. However, while the basic approach of the [2] definition of KE is intuitive and attractive, their actual definition of secure KE protocols has several subtle shortcomings. One consequence is that, contrary to their claims, their definition of KE seems insufficient to prove the security of the above-mentioned application to constructing authenticators (via KE and MAC). Another consequence is that their definition seems to be somewhat "over-restrictive", in the sense that it rules out KE protocols that seem "intuitively secure" and even provide secure composition with applications. In particular, Propositions 9 and 10 from [2] are incorrect.

On the work of Shoup [43]. Shoup's definitions are based on the simulatability approach of [2] with some significant modifications. Three levels of security are presented: *Static security* (i.e., security against adversaries that corrupt parties only at the onset of the computation), *adaptive security* (where the adversary obtains only the long-term information of a newly corrupted party) and *strongly adaptive security* where the adversary obtains all the private information of corrupted parties. (Oddly, strongly adaptive security does not imply adaptive security.) In addition, two definitions based on the indistinguishability approach of Bellare and Rogaway [7] are presented. The first is aimed at capturing security without perfect forward secrecy (PFS), and is shown to be equivalent to the static variant of the simulation-based definition. The second is aimed at capturing security with PFS, and is claimed to be equivalent to the adaptive variant of the simulation-based definition. Sufficiency of the definitions to constructing secure-channel protocols is informally argued, but is not proved nor rigorously claimed.

While the first variant of the indistinguishability-based definition is roughly equivalent to the non-PFS variant presented here (modulo the general differences mentioned below), the second variant is strictly weaker than our PFS formulation of SK-security. Specifically, the definition in [43] accepts as secure protocols that do not erase sensitive ephemeral data (e.g. protocol DHKE-1 in [43]), while the definition here treats these protocols as insecure.

There are several other technical and methodological differences between the two works that we mention next. (a) A major methodological difference is our use of the authenticated-links model and authenticators as a simplifying analysis tool. While our formalization of security does not mandate the use of this methodology we carefully build our definitions to accommodate the use of this tool. (b) Shoup allows the adversary a more general attack than session-key query, namely

an *application attack* that reveals an arbitrary function of the key. Our modeling does not define this explicit attack as it is subsumed by the session-key query capability and, in particular, since it is not necessary for guaranteeing secure channels. (c) Here we consider an additional adversarial behavior that is not treated in [43]. Specifically, we protect against adversaries that obtain the internal state of corrupted sessions (even without fully corrupting the corresponding parties) by requiring that such exposure will not compromise other protocol sessions run by the same parties. This protection is not guaranteed by some protocols suggested in [43] (e.g., protocol DHKE).
(d) The treatment of the interaction with the certificate authority (CA) in the case of public-key based authentication. In [43] the interaction with the CA is an integral part of every KE protocol, whereas here this interaction with the CA is treated as a separate protocol. We make this choice for further modularity and ease of proof. Yet, as we already remarked in Section 2.2, the CA protocol needs to be taken into consideration with any full specification and analysis of actual KE protocols. (e) The treatment of the session-id's. In [43] the session-id's are artificially given to the parties by the model which results, in our view, in a more cumbersome formalization of the security conditions. In contrast, here we adopt a more natural approach where the session-id's are generated by the calling protocol and security is guaranteed only when these session-id's satisfy some minimal (and easy to implement) conditions. In particular, this formalism can be satisfied by letting the parties jointly generate the session-id (as is common in practice).

On the works of Pfitzmann, Schunter and Waidner [41, 40, 42] and Canetti [14]. These works provide general frameworks for studying the security of cryptographic protocols in several models of computation, and prove some composition theorems with respect to protocols that satisfy their respective definitions of security. The proposed frameworks are natural candidates for defining and studying secure key-exchange protocols and their application to providing secure channels. In particular, Canetti [14] defines secure key exchange protocols, as well as protocols for providing "secure sessions" within his framework, and uses his general composition theorem in order to obtain similar results as the ones provided here (i.e., that the standard use of KE protocols for securing communication sessions result in a good "secure sessions" protocol). The [14] definition of secure KE protocols implies the definition here. However, it is somewhat over-restrictive, as it implies the definitions of both [2] and [43]. (In particular, we do not know how to show that Protocol ENC from Section 5.3 satisfies this definition.) In [16] we investigate a relaxed version of the [14] definition of key exchange, that is *equivalent* to the definition here and at the same time enjoys the general composability properties provided by the [14] framework.