An extended abstract of this paper appears in *Proceedings of the* 30th *Annual Symposium on the Theory of Computing*, ACM, 1998. This is the full version.

A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols

MIHIR BELLARE* RAN CANETTI† HUGO KRAWCZYK‡

March 13, 1998

Abstract

We present a general framework for constructing and analyzing authentication protocols in realistic models of communication networks. This framework provides a sound formalization for the authentication problem and suggests simple and attractive design principles for general authentication and key exchange protocols. The key element in our approach is a modular treatment of the authentication problem in cryptographic protocols; this applies to the definition of security, to the design of the protocols, and to their analysis. In particular, following this modular approach, we show how to systematically transform solutions that work in a model of idealized authenticated communications into solutions that are secure in the realistic setting of communication channels controlled by an active adversary.

Using these principles we construct and prove the security of simple and practical authentication and key-exchange protocols. In particular, we provide a security analysis of some well-known key exchange protocols (e.g. authenticated Diffie-Hellman key exchange), and of some of the techniques underlying the design of several authentication protocols that are currently being deployed on a large scale for the Internet Protocol and other applications.

*Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093. Email: mihir@cs.ucsd.edu. http://www-cse.ucsd.edu/users/mihir. Supported in part by NSF CAREER Award CCR-9624439 and a 1996 Packard Foundation Fellowship in Science and Engineering.

†IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, New York 10598. Email: canetti@watson.ibm.com.

‡Department of Electrical Engineering, Technion, Haifa 32000, Israel, and IBM T.J. Watson Research Center, New York, USA. Email: hugo@ee.technion.ac.il.

Contents

1 Introduction

A major task in cryptographic research is to define, and provide sound and efficient solutions for, the problem of secure communications. We need models and tools that enable the analysis of existing protocols as well as the development and analysis of new, improved solutions. Furthermore, in today's environment —we are seeing a rapidly growing demand for Internet security solutions— it is important to keep practical issues in mind from the start. This means that one cannot afford a significant sacrifice of simplicity and performance in order to achieve provability. Similarly, one cannot afford definitions which are an 'overkill' of the problem. It also requires a careful definition of the adversarial model in order to capture the real threats of today's global communications scenarios. Elements such as the concurrent execution of many instances of the same protocol by different parties, the inherent asynchrony of actual message-driven network protocols, and the scale of today's networks and applications must be considered.

1.1 The problems

The problem of "secure communications" is very broad, and encompasses a variety of sub-problems and goals. The focus of this paper is authentication in communications, with special emphasis on the related problem of key exchange.

Authenticated communications. In a nutshell, the authentication problem deals with curbing adversaries that actively control the communication links used by the legitimate parties. They may modify and delete messages in transit, and even inject false ones; they also control the *delays* of messages. (These adversaries may have additional capabilities, for instance corrupting parties.) Our goal is to guarantee authenticity of the communicated data; that is, make sure that the system behaves (as much as possible) as if the links were "physically authenticated" and no message was modified en-route or its origin misrepresented. Clearly, solving this problem is an essential step towards enabling secure communications in a realistic network environment.

The authentication problem has received less attention than secrecy and privacy issues, especially in the context of cryptographic protocols. Many of the fundamental works and techniques for the design of cryptographic protocols were developed under the idealized assumption of "authenticated links" between the communicating parties. Prime examples are the Diffie-Hellman key exchange protocol [DH] as well as much of the work on secure distributed function evaluation (eg, [Y, GMW, BGW, CCD]). This abstraction of the authentication problem is of great value when developing higher level protocols. However, implementing these "authenticated links" in realistic environments is far less trivial than one would have initially imagined. In particular, networks are often *asynchronous* in nature, and protocols are often *message-driven*. Here authentication cannot be achieved by simply applying signatures or message authentication codes to the transmitted data (although these primitives are useful tools in the design of complete solutions).

Several facets of the authentication problem have received a rigorous treatment, most notably entity authentication between two parties [BR1] and authenticated message transmission [Ra]. However there are basic and general questions that still remain untreated. The first part of our effort will go into properly formulating and then solving these. The tools that emerge are not only useful in their own right, but enable us to treat a more popular but notoriously difficult problem, namely authenticated key exchange, in effective ways.

Key exchange. An authenticated key exchange protocol enables two parties to end up with a shared secret key in a secure and authenticated manner. That is, no (computationally limited) adversary can impersonate any party during the protocol or learn any information about the value of the exchanged secret. Such protocols are an essential piece in the process of providing authenticated communication. On the other hand, they are themselves examples of problems in the domain of authenticated communications.

Authenticated key exchange has been extensively studied (we survey some of this work at the end of this section). Experience indicates that it is far more subtle and harder to achieve than may appear at first glance. A very large number of proposed protocols succumb to attacks not envisioned by their designers.

The first works to provide formal foundations in this area were those of Bellare and Rogaway [BR1, BR2]. They addressed the problems of entity authentication and key exchange in the two party, shared key model, and the problem of three party key distribution (Needham Schroeder [NS], or Kerberos [SNS] model). Generalizations are possible in order to accommodate other tasks and models of interest. But these formalizations are not always easy to work with. They also regard key-exchange as a stand-alone problem without putting it in a the wider context of authentication. Most importantly, each new key exchange protocol must be re-analyzed from scratch, and a proof provided that it meets one of the definitions.

1.2 Our approach and contributions

We present definitions and analysis techniques that provide a unified treatment of the different aspects of the authentication problem, thus leading us to to simplify the process of designing and analyzing authentication protocols. The key element is our modular treatment of the authentication problem. We start with solutions that work in a model of idealized authenticated communications and then transform these solutions, via automatic techniques (or "compilers"), into solutions that work in the realistic unauthenticated setting.

AUTHENTICATORS. The core component in our modular approach is the construction of "universal compilers" $\mathcal{C}$ that transform (or emulate) *any* protocol π in an ideally authenticated model into a protocol $\pi' = \mathcal{C}(\pi)$ that achieves essentially the same task as π (in a well defined sense) but withstands a much stronger and realistic adversary; not only can this adversary corrupt a subset of the parties, but it also has total control over the communication links connecting all the parties (corrupted or not). We call such a compiler $\mathcal{C}$ an authenticator. We formalize this notion and show several constructions of authenticators.

In particular, we show how the construction of authenticators is greatly simplified by reducing their design to that of much simpler protocols, called MT-authenticators, whose goal is limited to authenticate a simple exchange of messages between the parties. (Here MT stands for "message transmission".) These protocols have a very limited scope and are significantly easier to build and analyze than general authentication protocols. We show constructions of MT-authenticators based on standard cryptographic functions such as message authentication codes, digital signatures and public key encryption.

Following our methodology, a protocol intended to work in an unauthenticated network can be designed (or analyzed) in two independent stages: first design and prove the protocol in the authenticated model; then apply a particular authenticator. It is now guaranteed that the "compiled" protocol maintains, in the unauthenticated environment, the same behavior that the original one has in an ideally authenticated environment. The analysis of these two pieces is independent of each other and each piece can be "re-used" separately to get different protocols with different flavors of authentication. For example, one can use any of the basic authenticators proven in this paper in new authentication protocols without the need to re-prove the properties of these authenticators. In particular, one gets the security of the authenticated protocol solely based on the strength of the cryptographic functions underlying the design of the (MT-) authenticators. This modular approach has important practical consequences: it simplifies the design and analysis work, and helps building the protocol in ways that are easier to implement and maintain in practice. It also provides a "debugging tool" for the protocol designer and may help cleaning protocols from unnecessary elements. We exemplify these aspects through our construction and analysis of key exchange protocols as discussed below.

THE DEFINITIONAL APPROACH. The definition of authenticators involves a formal treatment of the authenticated and the unauthenticated models, as well as defining the notion of equivalence between protocols running in these different models. A basic ingredient in our formalization work is the notion of emulation of one protocol by another. This captures the idea of "equivalent behavior" of protocols running in different models of communication. (Below we illustrate this notion by sketching how it is used to define secure key-exchange protocols.) The notion draws from general definitions of secure multi-party protocols [MR, Bea, Ca1].

This definitional approach has several important properties which distinguish our work from previous treatment of the authentication problem: it makes the intuition behind the definitions clearer, it provides with uniform definitions for the different models of communication, and, very importantly, it enables our constructive methodology of transforming secure protocols from the idealized models to the realistic ones. In particular, this uniform treatment helps positioning key exchange in the broader context of the authentication problem.

KEY EXCHANGE. A main beneficiary of the above approach is the *key exchange* problem. We define this problem and its security requirements in a way that can fit both the authenticated and unauthenticated models. We do that by first defining an *ideal* key exchange process, where a fully honest and trusted party provides other parties, upon request, with shared secret keys. A secure key exchange protocol, in either the authenticated or unauthenticated model, is then defined as one where the result of the actions of the adversary can be efficiently *simulated* (or emulated, using our terminology) in the ideal model. In particular this means that "bad events" (such as secrecy compromise, replay of exchanges, impersonation, etc.) can happen in the actual protocols only if they could have happened in the ideal model. In order to achieve meaningful results, we carefully define the attacker in the ideal model to capture some of the unavoidable capabilities of a realistic attacker such as the ability to control the scheduling of key exchanges, and to corrupt all, or parts, of the information kept by the communicating parties.

This approach has as a prime advantage that it is applicable to different flavors of key exchange protocols, such as public key and shared key protocols, and can accommodate particular variants (e.g., identity protection, perfect forward secrecy, etc.) without requiring a complete re-definition of the problem.

Once the problem of key exchange is formally defined, we use the above modular authentication techniques to build and analyze specific key exchange protocols: we start with a key-exchange protocol that is proven secure in the simpler authenticated model, and then apply to it an authenticator that transforms it into a key exchange protocol secure in the unauthenticated model. We exemplify this process using the basic Diffie-Hellman protocol for idealized authenticated links [DH] and transforming it into several flavors of "authenticated Diffie-Hellman" through different MT-authenticators. As said before this reduces to only proving the basic Diffie-Hellman protocol in the much simpler authenticated model. Remarkably, we are able to provide in this way with analysis of several of the underlying key exchange mechanisms used in some well known practical key distribution protocols (see [DOW, Kra, ISO, HC]).

As another demonstration of the power of these tools, and as a significant result by itself, we prove validity of the following common approach to authenticating bulk communications: the parties first exchange a key using a particular key exchange protocol and then use this key to compute a MAC (message authentication code) function on the transmitted information. Using our methodology we show that any secure key exchange combined with a secure MAC function in this way represents a good authenticator.

RELATED WORK. As indicated above, the design of these protocols has a long history in which many protocols are proposed and broken. The works of Bird et al. [BGH+] and Diffie et al. [DOW], the first in the model of shared secrets and the second in the public key model, exposed several of the subtleties involved. This included issues like dealing with interleaving of different runs (or sessions) of a protocol and the importance of binding identities of sender and receiver to the exchanged information. These works also made clear the fact that no satisfactory definitions existed in the literature for this kind of protocols (in particular, they point out that previous extensive work in formalization of cryptographic primitives and protocols did not cover these basic authentication protocols).

The above-mentioned works of Bellare and Rogaway [BR1, BR2] provided formalizations for certain cases. They introduced the model of an adversary in charge of all communications, modeled sessions and session key reveal attacks, and suggested that the session key should be strongly secure, in the sense of semantic security. They also provided and proved secure some simple protocols. Various works address extending their framework to other settings and problems; for example, Shoup and Rubin to smart card settings [SR]; Lucks to consider dictionary attacks [Luc]; Blake-Wilson, Johnson and Menezes [BlMe, BJM] and Bellare, Petrank, Rackoff and Rogaway [BPRR] for the public key setting. See [MVV, Chapter 12] for general background and information on the subject of authenticated key exchange.

A different approach to the analysis of authentication and key exchange protocols is provided through the use of logic tools. The best known example is the BAN logic of [BAN]. This approach, however, abstracts out the cryptographic mechanisms and replace them with ideal primitives, thus limiting the significance of a successful analysis of a given protocol (on the other hand, these tools have proven useful to "debug" some weaknesses from certain authentication protocols).

Finally we note that a similar notion of an "authenticator" is used in [CHH]. There, however, both the setting and the solution are quite different: they design a *proactive* authenticator against a *mobile* adversary and in an ideally synchronized network.

ORGANIZATION. In Section 2 we present our communication models and formulate the central notion of an authenticator, i.e. a compiler for authenticated networks. We show how to build such an authenticator out of a much simpler protocol (an MT-authenticator) intended to authenticate a simple exchange of messages. In Section 3 we present two simple such MT-authenticators based on public key techniques. In Section 4 we define key exchange protocols and show how to use such protocols to construct authenticators which in turn form the basis for authentication of bulk data over typical communication networks. In Section 5 we show the application of the tools developed in previous sections to the construction of specific secure key exchange protocols. This is exemplified through several important variants of the authenticated Diffie-Hellman protocol. Proofs are omitted throughout, for lack of space. They appear in [BCK].

2 Authenticators

We introduce *authenticators*, i.e. compilers for unauthenticated networks, and show how to build them from simple protocols. Informally, an authenticator takes a protocol for (ideally) authenticated networks and turns it into a protocol that has similar input-output characteristics in an unauthenticated network. Here we formalize this notion and present a general methodology for building authenticators.

In existing unauthenticated networks (such as the Internet) an adversary may have control over the *delay* and *ordering* of messages. This control gives the adversary considerable powers, and in particular allows a host of attacks called *man-in-the-middle attacks.* In addition, protocols that do not depend on timings of messages are typically preferable in practice. In order to capture these concerns we allow our adversaries total control over the scheduling of messages. Formally, this modeling is reminiscent of the *asynchronous* model used in the theory of secure distributed computing (see [BCG] for instance). Yet, it is stressed that the notion of asynchrony there is different from the one here in several respects; in addition the motivations are different.

In Section 2.1 we present basic definitions that are central for our formalization and techniques. These include a formalization of (message-driven) protocols and of the adversarial model for authenticated and unauthenticated networks, as well as the definition of protocol emulation and of authenticators. In Section 2.2 we present a refinement of these definitions aimed at capturing the peculiarities of protocols that involve different sessions. In Section 2.3 we present a general approach for designing and analyzing authenticators. The approach regards an authenticator as a 'lower layer' communication protocol; this considerably facilitates designing and proving security of authenticators. In the rest of this work we concentrate on authenticators built using this approach.

2.1 Basic definitions

MESSAGE-DRIVEN PROTOCOLS. A message-driven protocol is an iterative process described as follows. The protocol is invoked by a party with some initial state that includes the protocol's input, random input, and the party's identity. Once invoked, the protocol waits for an activation. An activation can be caused by two types of events: the arrival of a message from the network, or an external request. (External requests model information coming from other processes run by the party). Upon activation, the protocol processes the incoming data together with its current internal state, generating a new internal state, as well as generating outgoing messages to the network and external requests to other protocols (or processes) run by the party. In addition, an output value is generated. We regard the output as cumulative. That is, initially the output is empty; in each activation the current output is appended to the previous one. Once the activation is completed, the protocol waits for the next activation. Formally, a protocol π is captured by a (probabilistic) function:

$$\pi(\text{current state, incoming message, external request}) =$$
$$(\text{new state, outgoing messages, outgoing requests, output})$$

THE AUTHENTICATED-LINKS MODEL (AM). There are n parties, $P_1 ... P_n$, each running a copy of a message-driven protocol π. The computation consists of a sequence of activations of π within different parties. The activations are controlled and scheduled by an adversary $\mathcal{A}$.[1] That is, initially and upon the completion of each activation $\mathcal{A}$ decides which party to activate next; $\mathcal{A}$ also decides which incoming message or external request the activated party is to receive. The outgoing messages, outgoing external requests and the output generated by the protocol become known to $\mathcal{A}$. The new internal state remains unknown to $\mathcal{A}$.

[1]All the adversaries in this paper are probabilistic polynomial time.

In the authenticated-links model, $\mathcal{A}$ is restricted to delivering messages faithfully. That is, we assume that each outgoing message carries the identities of the sender P_i and of the intended recipient P_j. When a message is sent by a party (ie, when the message appears in the list of outgoing messages in an activation of π within the party), the message is added to a set M of undelivered messages. Whenever $\mathcal{A}$ activates a party P_j on some incoming message m it must be that m is in the set M and that P_j is the intended recipient in m. Furthermore, m is now deleted from M.[2] We stress that $\mathcal{A}$ is not required to maintain the order of the messages, nor is it bound by any fairness requirement on the activation of parties, nor is it required to deliver all messages. There are also no limitations on the *external requests* that $\mathcal{A}$ issues.

In addition to activating parties, the adversary $\mathcal{A}$ can corrupt parties at wish. Upon corruption $\mathcal{A}$ learns the entire current state of the corrupted party P_i.[3] In addition, from this point on $\mathcal{A}$ can add to the set M any (fake) messages, as long as P_i is specified as the sender of these messages. The corrupted party appends a special symbol to its output, specifying that it is corrupted. From this point on the corrupted party is no longer activated (as its actions can be taken by the adversary itself), thus its output does not grow.[4] We refer to an adversary as described here as an AM-adversary.

The global output of running a protocol is the concatenation of the cumulative outputs of all the parties, together with the output of the adversary. The output of the adversary is a function, specific to each adversary, of the adversary view. The adversary view is all the information seen (and derived) by the adversary throughout the computation, together with its random input. Recall that the output of the parties includes registration of important events that occurred during the execution (such as corruption of parties). This provision ensures that these events are considered in the security requirement (Definition 1 below).

We use the following notation. Let $\text{ADV}_{\pi,\mathcal{A}}(\vec{x},\vec{r})$ denote the output of adversary $\mathcal{A}$ when interacting with parties running protocol π on input $\vec{x} = x_1 \dots x_n$ and random input $\vec{r} = r_0 \dots r_n$ as described above (r_0 for $\mathcal{A}$; x_i and r_i for party P_i, $i > 0$). Let $\text{AUTH}_{\pi,\mathcal{A}}(\vec{x},\vec{r})_i$ denote the cumulative output of party P_i after running protocol π on input $\vec{x}$ and random input $\vec{r}$, and with an AM-adversary $\mathcal{A}$. Let $\text{AUTH}_{\pi,\mathcal{A}}(\vec{x},\vec{r}) = \text{ADV}_{\pi,\mathcal{A}}(\vec{x},\vec{r}), \text{AUTH}_{\pi,\mathcal{A}}(\vec{x},\vec{r})_1 \dots \text{AUTH}_{\pi,\mathcal{A}}(\vec{x},\vec{r})_n$ .. Let $\text{AUTH}_{\pi,\mathcal{A}}(\vec{x})$ denote the random variable describing $\text{AUTH}_{\pi,\mathcal{A}}(\vec{x},\vec{r})$ when $\vec{r}$ is uniformly chosen.

THE UNAUTHENTICATED-LINKS MODEL (UM). Basically, the unauthenticated-links model of computation is similar to the authenticated-links one, with the exception that here the adversary $\mathcal{U}$, referred to as a UM-adversary, is not limited to deliver messages that are in M. Instead, it can activate parties with arbitrary incoming messages (even with fake messages that were never sent).

In addition, here 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 bootstrapping of the cryptographic authentication functions, e.g. by letting the parties choose private and public keys for some asymmetric cryptosystem and trustfully exchange the public keys.) Function I takes only a random input r, and outputs a vector $I(r) = I(r)_0 \dots . I(r)_n$. The component $I(r)_0$ is the public information and becomes known to all parties

[2]Here we assume that no message appears twice. Alternatively, one can add message-ID's to messages to make them unique.

[3]By limiting the adversary to learn only the current state of the corrupted party we allow protocols that *erase* data. An alternative, more conservative formalization (which we do not adopt) allows the adversary to learn all the past internal states of the party, thus making erasures pointless. This more conservative formalization captures the reluctance of parties to base their security on the good will of *other* parties to locally erase data. See [CFGN] for a discussion. In this work, the distinction between the two formalizations will become apparent in Section 5.

[4]We do not limit the number of parties that the adversary can corrupt. This reflects our goal of providing authenticity against adversaries that corrupt any number of parties.

and to the adversary. For $i > 0$, $I(r)_i$ becomes known only to P_i.[5]

We define $\textsc{unauth}_{\pi,\mathcal{U}}(\vec{x},\vec{r})$ and $\textsc{unauth}_{\pi,\mathcal{U}}(\vec{x})$ analogously to $\textsc{auth}_{\pi,\mathcal{A}}(\vec{x},\vec{r})$ and $\textsc{auth}_{\pi,\mathcal{A}}(\vec{x})$, where here the computation is carried out in the unauthenticated-links model. The initialization function I is part of the description of protocol π.

Emulation of protocols. We define what it means for a protocol π' in the unauthenticated-links model to emulate a protocol π in the authenticated-links model. We want to capture the idea that 'running π' in an unauthenticated network has the same effect as running π in an authenticated network'. This is done following a general approach used for defining secure multi-party protocols [MR, Bea, Ca1]. That is:

Definition 1 *Let π and π' be message-driven protocols for n parties. We say that π'* emulates π in unauthenticated networks *if for any* UM*-adversary $\mathcal{U}$ there exists an* AM*-adversary $\mathcal{A}$ such that for all input vectors $\vec{x}$,*

$$\textsc{auth}_{\pi,\mathcal{A}}(\vec{x}) \overset{c}{\approx} \textsc{unauth}_{\pi',\mathcal{U}}(\vec{x}) \tag{1}$$

where $\overset{c}{\approx}$ denotes 'computationally indistinguishable'.[6]

Requirement (1) incorporates many conditions. In particular, the combined distributions of the outputs of the parties, the adversary's output, and the identities of corrupted parties, should be indistinguishable on the two sides of (1). In general, this condition captures the required notion of "security equivalence" between the protocols in the sense that any consequences of the actions of the strong UM-adversary against executions of the protocol π' can be imitated or achieved by the weaker AM-adversary against the runs of protocol π without requiring the corruption of more (or different) parties.

An authenticator is a 'compiler' that takes for input protocols designed for authenticated networks, and turns them into 'equivalent' protocols for unauthenticated networks:

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 unauthenticated networks.*

In particular, authenticators translate secure (in some well defined sense) protocols in the authenticated-links model into secure protocols in the unauthenticated-links model. In the following sections we show constructions of authenticators and their applications (e.g., we show how to transform the basic Diffie-Hellman protocol that runs securely in the authenticated-links model into a secure key exchange protocol in the unauthenticated-links model).

---

[5]The initialization function I models a 'perfectly secure' initial exchange where the private information of all parties is chosen according to the protocol, and all parties learn the authentic public information pertaining to all other parties. Alternatively, the initial information exchange among the parties can be modeled via an explicit in-band communication phase where the adversary is restricted to be authenticated (ie to deliver only messages in M) and perhaps also synchronous. This alternative approach has the advantage that it captures possible weaknesses of methods for initial key exchange (eg, of protocols for certification authorities). In particular, this approach allows the adversary to corrupt parties during the initial phase. In this work we 'abstract out' the initial exchange phase and stick to the initialization function formalization.

Also note that The initialization function I allows for other types of initial exchanges, other than exchanging public keys of an asymmetric cryptosystem. For instance, I can contain shared secret keys between each pair of parties. Consequently, this same model can be used to analyze also shared-key based authentication protocols. In this work we concentrate on initialization functions where the parties do *not* share secret data.

[6]Two distribution ensembles are computationally indistinguishable if no polytime adversary can distinguish them with more than negligible probability. Here the asymptotics is over a security parameter that is implicit in the description.

2.2 Incorporating sessions

Definitions 1 and 2 treat each party as a monolythic entity: either a party is totally corrupted or it is totally secure. However, communication protocols often have additional structure that allows a party to have, at the same time, several "independent conversations" with other parties. (Two parties may even have several parallel "conversations" running between them.) These conversations, often called sessions, may be very different from each other, and in particular may have different security requirements. It is thus convenient to let each session have its own cryptographic protection mechanisms, and to ensure that sessions are "cryptographically independent" from each other as much as possible. That is, we want to make sure that compromising the security of one session will have as little effect as possible on the security of other session run by the same party.

To accomodate the session structure, we refine definitions 1 and 2 as follows. First we refine our formalization of message-driven protocols. One or more subroutines (or sub-protocols) can be defined within the code of a protocol. The syntax of each sub-protocol is similar to the syntax of a (message-driven) protocol described in Section 2.1. During a run of the protocol several invocations of these sub-protocols may occur. A protocol may classify some of these invocations as sessions.[7] Typically, the local state of a session is mostly independent of local state of other sessions. (Yet, often sessions do share some limited amount of state, e.g. the public and private keys of the party.) Each session is identified by a unique session ID. Notice that a message sent from a sender to an intended recipient is part of the sender's session and the recipient's session. We assume that such a message carries, in addition to the identities of the sender and receiver, also the ID corresponding to the sender's session and the ID corresponding to the recipient's session.

Upon activation of the protocol with an incoming message m, a special code called a session manager is executed. Typically, the session manager activates the session specified in m with incoming message m. (Alternatively, the session manager can decide to activate several sessions, to invoke a new session, or to discard the incoming message without invoking any session.) In addition, if an activated session generates outgoing requests $t_1 ... t_k$ that are addressed to other sessions $\rho_1 ... \rho_k$ within the party, then sessions $\rho_1 ... \rho_k$ are activated, one by one, each ρ_i with incoming request t_i. Finally, the session manager outputs all the outgoing messages from all the activated sessions.

Although all sessions are run by the same party, we would like to think of the different sessions as of independent modules to a large extent. In particular, we want to allow the adversary to compromise the security of only *some* of the sessions within some party. In this case we want the uncompromised sessions to proceed uninterrupted. In order to capture these additional requirements, the definitions of the authenticated-links and unauthenticated-links models are enriched as follows. Recall that a protocol may specify distinct sessions (at run-time). A well-defined part of the local state is associated with each session. In both models we allow the adversary an additional type of corruption, called a session corruption. In a session corruption the adversary learns only the internal state associated with a particular session. In addition, a special value is added to the party's output specifying that the particular session (identified by its session ID) has been corrupted. Once the adversary corrupts a session, it can add any (fake) messages to the set M of undelivered messages, as long as the specified origin of these messages is the corrupted session. The rest of Definitions 1 and 2 remains unchanged.

We maintain the convention that all corruption requests made by the adversary are registered in the parties' outputs, thus they become part of the global output of the computation. This

[7] Although the previous paragraph follows the common interpretation of a session as an object that involves two (conversing) parties, here we define a session as a syntactic object that is local to a party. This greatly simplifies the presentation. In fact, a conversation between two parties is now a *pair* of sessions — one session within each of the involved parties.

convention ensures that if protocol π' emulates π then π' must preserve the partitioning of protocol π into sub-protocols, and have similar session structure and session numbers.

2.3 MT-authenticators

We present the following general technique for designing authenticators. First design a 'lower layer' protocol λ that takes external requests for sending messages and sends these messages in an authenticated way. Next, given some protocol π (designed to work in the authenticated-links model), the authenticator will output a protocol π' that is identical to protocol π with the exception that messages are delivered through λ. That is, instead of sending messages to the network, λ is activated for delivery of these messages, and instead of receiving incoming messages from the network, the messages are taken from λ's output. We prove that this technique yields valid authenticators.

More precisely, consider the following simple protocol, called the message transmission (MT) protocol, and designed for authenticated networks. The protocol takes empty input. Upon activation within P_i on external request (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'.

Let λ be a protocol that emulates MT in unauthenticated networks. (We call such protocols MT-authenticators.) Then, define a compiler $\mathcal{C}_\lambda$ as follows. Given a protocol π, the generated protocol $\pi' = \mathcal{C}_\lambda(\pi)$, running within party P_i, first invokes λ. Next, for each message that π sends, π' activates λ with external request for sending that message to the specified recipient. Whenever π' is activated with some incoming message, it activates λ with this incoming message. When λ outputs 'P_i `received` m `from` P_j', protocol π is activated with incoming message m from P_j. We call authenticators that follow this design layered authenticators. This name is justified by the following theorem.

Theorem 3 *Let λ be an MT-authenticator (i.e. λ emulates MT in unauthenticated networks), and let $\mathcal{C}_\lambda$ be a compiler constructed based on λ as described above. Then $\mathcal{C}_\lambda$ is an authenticator.*

Proof: Let π be a protocol. We show that $\pi' = \mathcal{C}_\lambda(\pi)$ emulates π in unauthenticated networks. That is, let $\mathcal{U}_{\pi'}$ be a UM-adversary that works against π'. We construct an AM-adversary $\mathcal{A}_\pi$ such that for all inputs $\vec{x} = x_1 ... x_n$,

$$\text{AUTH}_{\pi,\mathcal{A}_\pi}(\vec{x}) \stackrel{s}{=} \text{UNAUTH}_{\pi',\mathcal{U}_{\pi'}}(\vec{x}) \tag{2}$$

where $\stackrel{s}{=}$ stands for 'having negligible statistical distance'.[8]

Adversary $\mathcal{A}_\pi$ runs $\mathcal{U}_{\pi'}$ on the following simulated interaction with a set of n parties $P'_1 ... P'_n$ running π' on input $\vec{x}$. (Intuitively, this is the "obvious" simulation, where $\mathcal{A}_\pi$ orchestrates an interaction between $\mathcal{U}_{\pi'}$ and λ, and at the same time uses the parties $P_1 ... P_n$ in the authenticated-links model to play the upper-layer protocol for λ.) First $\mathcal{A}_\pi$ invokes λ. Next, $\mathcal{A}_\pi$ proceeds according to the following rules:

1. Whenever $\mathcal{U}_{\pi'}$ activates P'_i with an external request, $\mathcal{A}_\pi$ activates (in its authenticated-links model) P_i with the same request. For each outgoing message that P_i generates in this activation, say for P_j, $\mathcal{A}_\pi$ activates protocol λ with external request for sending that message from P'_i to P'_j. Next $\mathcal{A}_\pi$ hands $\mathcal{U}_{\pi'}$ all the outgoing messages generated in the current imitated activation of P'_i, together with any outgoing requests and outputs that P_i may have generated.

[8] Interestingly, here we only assume that λ emulates MT with computational indistinguishability (see equation (1)), but end up with statistical closeness for $\mathcal{C}_\lambda$ (see equation (2)). This is an extra feature of using layered authenticators.

2. Whenever $\mathcal{U}_{\pi'}$ activates P'_i with an incoming message m, $\mathcal{A}_\pi$ first activates λ with this incoming message. Any outgoing messages and external requests generated by P'_i as the result of this activation by λ are handed by $\mathcal{A}_\pi$ to $\mathcal{U}_{\pi'}$.
3. If in any activation of λ party P'_i outputs 'P'_i `received` m `from` P'_j', then $\mathcal{A}_\pi$ activates, in the authenticated-links model, party P_i with incoming message m from party P_j. For each outgoing message that P_i generates as a result of this activation, $\mathcal{A}_\pi$ imitates an invocation within P'_i of λ for sending that message to the intended recipient. Next $\mathcal{A}_\pi$ hands $\mathcal{U}_{\pi'}$ all the outgoing messages generated in the current imitated activation of P'_i under λ, together with any outgoing requests and outputs that P_i may have generated.
4. Whenever $\mathcal{U}_{\pi'}$ corrupts party P'_i, $\mathcal{A}_\pi$ corrupts P_i in the authenticated-links model. $\mathcal{A}_\pi$ then hands $\mathcal{U}_{\pi'}$ the internal data of P_i together with the information regarding all copies of MT$'$ *within P'_i (i.e. all activations of P'_i by λ. If $\mathcal{U}_{\pi'}$ corrupts a session within some party P'_i then $\mathcal{A}_\pi$ corrupts the same session within P_i and hands the corresponding information back to $\mathcal{U}_{\pi'}$.*
5. $\mathcal{A}_\pi$ outputs whatever $\mathcal{U}_{\pi'}$ outputs.

We first need to show that the above description of the behavior of $\mathcal{A}_\pi$ is a legitimate behavior of an AM-adversary. The above steps are easy to verify as legal moves for $\mathcal{A}_\pi$, except for step 3. In the later case, it could be possible that the triple (P_j, P_i, m) is not currently in the set M of undelivered messages in the authenticated-links model, and P_j (and the originating session within P_j) is uncorrupted. It is easy to see that if this is not the case, namely, if we assume that step 3 can always be carried out, then the above construction satisfies (2). (In fact, under this condition the two sides of (2) are *identically distributed.*)[9] This holds since the simulated run of $\mathcal{U}_{\pi'}$, together with the activations of the parties in the authenticated-links model is an exact imitation of a run of $\mathcal{U}_{\pi'}$ in an unauthenticated network with parties running π'.

Thus, it remains to show that the probability that $\mathcal{A}_\pi$ cannot carry out step 3 is negligible. Assume that protocol λ within an *uncorrupted* party P'_i outputs 'P'_i `received` m `from` P'_j'. We show that, except with negligible probability, the following events occur in the authenticated-links model: (I). The triple (P_i, P_j, m) was added to M. (II). This triple was not yet deleted from M.

To see (I), notice first that if, in the simulated run of $\mathcal{U}_{\pi'}$, party P'_i outputs 'P'_i `received` m `from` P'_j', then P'_j has invoked λ for sending m to P'_i.[10] It now follows from the construction that P_j has sent m to P_i in the authenticated-links model. Thus the triple (P_i, P_j, m) was added to M.

To see (II), notice that, in the simulated run of $\mathcal{U}_{\pi'}$, P'_i has not previously output 'P'_i `received` m `from` P'_j'. (Otherwise it would be the case that P'_i outputs 'P'_i `received` m `from` P'_j' twice, and again one can construct from $\mathcal{U}_{\pi'}$ a UM-adversary $\mathcal{U}_\lambda$ that contradicts the assumption that λ emulates MT.) It now follows from the construction that P_j was not previously activated, in the authenticated-links model, with incoming message m from P_j. Thus the triple (P_i, P_j, m) was not deleted from M. □

[9] A bit more precisely, let $\mathcal{B}$ be the event that step 3 cannot be carried out. Then, each execution where event $\mathcal{B}$ does not occur has the same probability in the simulated and real interactions.

[10] Otherwise we construct from $\mathcal{U}_{\pi'}$ a UM-adversary $\mathcal{U}_\lambda$ that contradicts the assumption that λ emulates MT. $\mathcal{U}_\lambda$ will simply run $\mathcal{U}_{\pi'}$ (and complete $\mathcal{U}_{\pi'}$'s missing information by running a simulated copy of π on input $\vec{x}$). Now, in the global output of the parties running λ with $\mathcal{U}_\lambda$ the event that some party accepts a message that was never sent occurs with non-negligible probability. Yet, in an authenticated network, with *any* adversary, this event never occurs. We omit further details from this sketch.

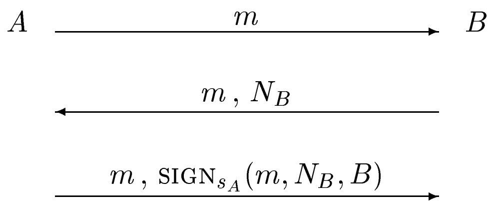

Figure 1: Signature based MT-authenticator

3 Two simple MT-authenticators

We present two simple MT-authenticators. These MT-authenticators adopt the following approach. Assume that the parties have set up keys of an asymmetric cryptosystem. (That is, each party has its private key as well as the public keys of all parties.) Now, authenticate each message *independently from all other messages.* That is, the MT-authenticator invokes an independent copy of some simple protocol for each message of the original protocol. One may think of such MT-authenticator as ones that consider each sending of a message as a session, and authenticate each session separately.

Applying Theorem 3, the two MT-authenticators can be made into full-fledged authenticators. In particular, they play a central role in our constructions of key-exchange protocols in Section 5.

3.1 A signature-based MT-authenticator

We construct an MT-authenticator, λ_{SIG}, based on public key signatures. The initialization function I first invokes, once for each party, the key generation algorithm of a signature scheme secure against chosen message attack with security parameter k. Let SIGN and VER denote the signing and verification algorithms. Let s_i and v_i denote the signing and verification keys associated with party P_i. The public information is all public keys: $I_0 = v_1 ... v_n$. P_i's private information is $I_i = s_i$.

Next, when activated, within party P_i and with external request to send message m to party P_j, protocol λ_{SIG} invokes a two-party protocol $\hat{\lambda}_{\text{SIG}}$ that proceeds as follows. (Since $\hat{\lambda}_{\text{SIG}}$ involves only two parties, we use A and B instead of P_i and P_j. Also, in the following description we alternate between instructions for the sender A and for the recipient B.) First, A sends '`message:`m' to B. (A also outputs 'A `sent message` m `to` B'.) Upon receipt of '`message:`m' from A, party B chooses a random value $N_B \in_{\text{R}} \{0,1\}^k$, and sends '`challenge:`$m, N_B$' to A. Upon receipt of '`challenge:`m, N_B' from B, party A sends '`signature:`$m, \text{SIGN}_{s_A}(m, N_B, B)$' to B. Upon receipt of '`signature:`$m, \text{SIGN}_{s_A}(m, N_B, B)$', party B accepts m (ie, it outputs 'B `received` m `from` A') if the signature is successfully verified. Pictorially, the exchange for a single message can be described as in Figure 1.

Proposition 4 *Assume that the signature scheme in use is secure against chosen message attacks. Then protocol λ_{SIG} emulates protocol* MT *in unauthenticated networks.*

Proof: Let $\mathcal{U}$ be a UM-adversary that interacts with λ_{SIG}. We construct an AM-adversary $\mathcal{A}$ such that $\text{AUTH}_{\text{MT},\mathcal{A}}() \stackrel{s}{=} \text{UNAUTH}_{\lambda_{\text{SIG}},\mathcal{U}}()$. (Since protocol MT ignores its input, we consider only the empty input.) Adversary $\mathcal{A}$ proceeds similarly to adversary $\mathcal{A}_\pi$ in the proof of Theorem 3. That is, $\mathcal{A}$ runs $\mathcal{U}$ on a simulated interaction with a set of parties running λ_{SIG}. First $\mathcal{A}$ chooses and distributes keys for the imitated parties, according to function I. Next, when $\mathcal{U}$ activates some imitated party A' for sending a message m to imitated party B', adversary $\mathcal{A}$ activates party A in the authenticated network to send m to B. In addition, $\mathcal{A}$ continues the interaction between $\mathcal{U}$ and the imitated parties running λ_{SIG}. When some imitated party B' outputs 'B' `received` $\hat{m}$ `from` A'', adversary $\mathcal{A}$ activates party B in the authenticated-links model with incoming message $\hat{m}$ from A. When $\mathcal{U}$ corrupts a party, $\mathcal{A}$ corrupts the same party in the authenticated network and hands the corresponding information (from the simulated run) to $\mathcal{U}$. Finally, $\mathcal{A}$ outputs whatever $\mathcal{U}$ outputs.

Let $\mathcal{B}$ denote the event that imitated party B' outputs 'B' `received` m `from` A'' where A' is uncorrupted and the message (m, A, B) is not currently in the set M of undelivered messages. In other words, $\mathcal{B}$ is the event where B' outputs 'B' `received` m `from` A'', and either A wasn't activated for sending m to B, or B has already had the same output before. In this event we say that $\mathcal{U}$ broke party A'. As in the proof of Theorem 3, it is straightforward to see that if $\mathcal{A}_\pi$ could continue with the simulation even when event $\mathcal{B}$ occurs, then the simulation run by $\mathcal{A}$ is perfect and $\text{AUTH}_{\text{MT},\mathcal{A}}() \stackrel{\text{d}}{=} \text{UNAUTH}_{\lambda_{\text{SIG}},\mathcal{U}}()$. (Here $\stackrel{\text{d}}{=}$ denotes 'equally distributed'.)

It remains to show that event $\mathcal{B}$ occurs only with negligible probability. Assume that event $\mathcal{B}$ occurs with probability ϵ. We construct a forger $\mathcal{F}$ that breaks the signature scheme with probability ϵ/n. Given a verification key v^* and access to a signing oracle, forger $\mathcal{F}$ runs $\mathcal{U}$ on the following simulated interaction with a set of parties running λ_{SIG}. First $\mathcal{F}$ chooses and distributes keys for the imitated parties according to function I, with the exception that the public verification key associated with some party P^*, chosen at random, is replaced with the input key v^*. Next, $\mathcal{F}$ continues the simulated interaction. If during the simulation P^* is required to sign a value l then $\mathcal{F}$ asks its signing oracle for a signature of l. If party P^* is corrupted then the simulation is aborted and $\mathcal{F}$ fails. If some party Q outputs 'Q `received` m `from` P^*', and P^* was not activated to sent m to Q (or if Q has already output this value before), then $\mathcal{F}$ announces success and outputs the signature in the last incoming message that Q received.

First note that $\mathcal{U}$'s view of the interaction with $\mathcal{F}$, conditioned on the event that $\mathcal{F}$ does not abort the simulation, is identically distributed to $\mathcal{U}$'s view of a real interaction with an unauthenticated network. (This is so since P^* is randomly chosen.) Let $\mathcal{B}^*$ be the event where $\mathcal{B}$ occurs in the simulated run of $\mathcal{U}$ within $\mathcal{F}$, and that the party broken by $\mathcal{U}$ is P^*. Since P^* is chosen at random, and since event $\mathcal{B}^*$ and an abortion never occur at the same run, we have that event $\mathcal{B}^*$ occurs with probability at least ϵ/n.

Assume that event $\mathcal{B}^*$ occurs. In this case, the signature that Q received in its last incoming message is a valid signature (w.r.t. verification key v^*) of a value (m, N_Q, Q). Moreover, P^* never generated this signature. (In case that P^* was not activated to send m to Q, it is clear that P^* never signed this message. In the case that Q outputs the same value twice, recall that all messages are different, thus P^* sent m only once. However, with probability $(1-2^{-k})$ the challenge N_Q that appears in the above signature is different than the challenge that P^* signed.) Thus, $\mathcal{F}$ never asked its oracle for this signature. Consequently, $\mathcal{F}$ has successfully broken the signature scheme. $\square$

REMARKS. 1. Protocol λ_{SIG} can be modified to replace the challenge N_B in the second flow with a counter that makes sure that no value appears twice. The proof remains unchanged. Yet, now λ_{SIG} has to 'maintain common state' among the different 'sessions' (i.e, among the activations pertaining to different π-messages). Maintaining state across sessions is problematic and inadvisable in actual systems.
2. Another possible modification of λ_{SIG} is to avoid the second flow altogether, and collapse the first and third flows to a single flow where A signs (m, B). To avoid replay of messages by the adversary, each party will maintain a database of all the messages received in the past and will accept only new messages. Yet, this protocol too requires the party to maintain (a lot of) state

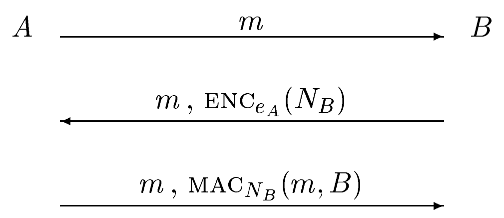

Figure 2: Encryption based MT-authenticator

across sessions, to the point of impracticality.
3. One does not need to send the message m in each of the flows of the protocol. A can send it in the first flow only or even in the last flow only (in the latter case the first flow can just carry a notice that there is a message "waiting for delivery"). The flow from B to A does not need to carry the message but needs some unique "identifier" (similar to a session-ID) that bounds the particular value of the challenge N_B to the message being transmitted.
4. All the three elements signed by A in the third flow are necessary. Omitting any one of these elements makes our proof fail and actual attacks against the security of the scheme are possible. This includes the less obvious case in which B's identity is omitted. An attack in this case was described by Diffie et al [DOW].

3.2 An encryption-based MT-authenticator

We construct a MT-authenticator, λ_{ENC}, based on public key encryption. In addition, λ_{ENC} uses a message authentication (MAC) scheme. This protocol is based on the authentication technique underlying the key exchange mechanisms in [Kra]. (It is also reminiscent of the [DDN] message authentication protocol.) Protocol λ_{SIG} proceeds as follows. The initialization function I first invokes, once for each party, the key generation algorithm of a public key encryption scheme semantically secure against chosen ciphertext attack in the sense of [RS], with security parameter k.[11] Let ENC and DEC denote the encrypting and decrypting algorithms, and let MAC denote the MAC scheme in use. Let e_i and d_i denote the signing and verification keys associated with party P_i. The public information is all the public keys: $I_0 = e_1 ... e_n$. P_i's private information is $I_i = d_i$.

Next, when activated within party A, and with external request to send message m to party B, protocol λ_{ENC} invokes a two-party protocol $\hat{\lambda}_{\text{ENC}}$ that proceeds as follows. First, A sends '`message:`m' to B. (A also outputs 'A `sent message` m `to` B'.) Upon receipt of '`message:`m' from A, party B chooses a random value $N_B \in_{\text{R}} \{0,1\}^k$, and sends '`challenge:`$m, \text{ENC}_{e_A}(N_B)$' to A. Upon receipt of '`challenge:`$m, \text{ENC}_{e_A}(N_B)$' from B, party A obtains N_B by decrypting the value in the last field in the incoming message, and sends '`mac:`$m, \text{MAC}_{N_B}(m, B)$' to B. Upon receipt of '`mac:`m, v', party B proceeds as follows. If $v = \text{MAC}_{N_B}(m, B)$ then B accepts m (ie, it outputs 'B `received` m `from` A'). Otherwise, B rejects this message and terminates this invocation of $\hat{\lambda}_{\text{ENC}}$. (Terminating the session in case of a bad MAC is not essential for the security; however, considerably simplifies the analysis.) Pictorially, the exchange for a single message can be described as in Figure 2.

[11] The adversary is given a ciphertext and is allowed to ask for decryption of any value other than the value given. It must eventually decide whether the given ciphertext is an encryption of a given value. Here we use a weak version of [RS] security (see remark after the proof of Proposition 5). The first encryption scheme secure in this sense appears in [DDN].

Proposition 5 *Assume that the encryption scheme in use is secure against chosen ciphertext attacks in the sense of [RS], and that the MAC scheme in use is secure. Then protocol* λ_{ENC} *emulates protocol* MT *in unauthenticated networks.*

Proof: Up till the definition of event $\mathcal{B}$, the proof is identical to the proof of Proposition 4. We pick up the proof from this point, showing that event $\mathcal{B}$ occurs only with negligible probability.

We proceed in two steps. First we use $\mathcal{U}$ to construct a forger $\mathcal{F}$ that breaks the MAC scheme, given an encryption of the secret key in use, and oracle access to a decryption box. Next we use $\mathcal{F}$ to construct a distinguisher $\mathcal{D}$ that breaks the semantic security of the encryption scheme against chosen ciphertext attacks.

Assume that event $\mathcal{B}$ occurs with probability ϵ, and let l be the total number of messages that $\mathcal{U}$ delivers in its run. (That is, $\mathcal{U}$ performs at most l activations of parties with incoming message `'message:...'`.) Then Forger $\mathcal{F}$ succeeds with probability ϵ/l, making at most l MAC queries. Distinguisher $\mathcal{D}$ breaks the semantic security of the encryption scheme with probability at least $\frac{\epsilon}{l} - \epsilon_{\text{MAC}}$, where ϵ_{MAC} is an upper bound on the probability to forge the MAC scheme with l queries. That is, if ϵ_{ENC} is an upper bound on the breaking probability of the encryption scheme then $\epsilon \leq l(\epsilon_{\text{MAC}} + \epsilon_{\text{ENC}})$.

More precisely, define an encryption-aided MAC forger, $\mathcal{F}$, as follows. Let e^*, d^* be a pair of encryption and decryption keys of the encryption scheme in use with security parameter k (ie, $G(1^k) = (e^*, d^*)$, where G is the key generation algorithm). Forger $\mathcal{F}$ takes for input the encryption key e^* and an encryption $f = \text{ENC}_{e^*}(N^*)$ where $N^* \in_{\text{R}} \{0,1\}^k$. In addition, $\mathcal{F}$ has access to two oracles: One is a decryption oracle D that decrypts messages (different than f) according to the decryption key d^*. (That is, $D(f) = \perp$. If $\hat{f} \neq f$ then $D(\hat{f}) = \text{DEC}_{d^*}(\hat{f})$.) The other oracle is a MAC oracle C with secret key N^*. That is, $C(m) = \text{MAC}_{N^*}(m)$. The goal of $\mathcal{F}$ is to output a pair $\hat{m}, \text{MAC}_{N^*}(\hat{m})$ where C wasn't queried on $\hat{m}$.

Construction of $\mathcal{F}$. Given $\mathcal{U}$, we construct an encryption-aided MAC forger. Forger $\mathcal{F}$ runs $\mathcal{U}$ on the following simulated interaction with a set of parties running λ_{ENC}. First $\mathcal{F}$ chooses and distributes keys for the imitated parties according to function I, with the exception that the public encryption key associated with some party P^*, chosen at random, is replaced with the input key e^*. Next, $\mathcal{F}$ continues the simulated interaction. Let m^* be a message chosen at random out of all messages m such that some party P was activated with `'message:`m`'` from P^*.[12] If during the simulation party P^* is corrupted then the simulation is aborted and $\mathcal{F}$ fails. If party P^* is required to decrypt a value $\text{ENC}_{e^*}(N)$ that is sent in response to receipt of a message different than m^* then $\mathcal{F}$ asks its decryption oracle for the decryption and proceeds with the simulation. When some party P is activated by $\mathcal{U}$ with incoming message m^* then $\mathcal{F}$ has P respond with `'challenge:`m^*, f`'` where f is $\mathcal{F}$'s input. (That is, P's encrypted challenge is N^*.) Next, if P^* is activated with incoming message `'challenge:`m, f`'` from some party Q and $m \neq m^*$ or $Q \neq P$, then $\mathcal{F}$ asks its MAC oracle for $\text{MAC}_{N^*}(m, Q)$, and proceeds with the simulation. If $m = m^*$ and $Q = P$ then the simulation is aborted and $\mathcal{F}$ fails. Finally, if $\mathcal{U}$ activates P with incoming message `'mac:`m^*, c`'` from party P^*, then $\mathcal{F}$ outputs m^*P, c and halts, hoping that $c = \text{MAC}_{N^*}(m^*P)$.

Analysis of $\mathcal{F}$. First note that $\mathcal{U}$'s view of the interaction with $\mathcal{F}$ (conditioned on the event that $\mathcal{F}$ does not abort the simulation) is distributed identically to $\mathcal{U}$'s view of a real interaction with an

[12] If the total number l^* of message that P^* sends and $\mathcal{U}$ delivers is known in advance then $\mathcal{F}$ simply choose at random $i \in_{\text{R}} \{1..l^*\}$ and lets m^* be the ith message that P^* sends and $\mathcal{U}$ delivers. Otherwise, $\mathcal{F}$ chooses m^* via the following process. Whenever some party P was activated with `'message:`m`'` from P^*, forger $\mathcal{F}$ decides to choose $m^* = m$ with an appropriate probability, making sure that by the end of the run all the candidate messages have equal probability to be chosen. We omit further details.

unauthenticated network. (This is so since P^* and m^* are uniformly distributed in their domains.) Let $\mathcal{B}^*$ be the event where $\mathcal{B}$ occurs in the simulated run of $\mathcal{U}$ within $\mathcal{F}$, that the party broken by $\mathcal{U}$ is P^*, and that the message with which P^* is broken is m^*. Since P^* and m^* are chosen at random, and since event $\mathcal{B}^*$ and an abortion never occur at the same run, we have that event $\mathcal{B}^*$ occurs with probability ϵ/l.

Assume that event $\mathcal{B}^*$ occurs. In this case, the MAC that P received in its last incoming message is a valid MAC (w.r.t. verification key N^*) of the value (m^*, P). Moreover, P^* never generated this MAC. (In case that P^* was never activated to send m^* to P, it is clear that P^* never generated this MAC. Otherwise, we have that P outputs the same value twice. In this case, recall that all messages are different, thus P^* sent m^* only once. However, with probability $1-2^{-k}$ the key N^* used in the above MAC is different than the key used in the MAC generated by P^*.) Thus, $\mathcal{F}$ never asked its MAC oracle for this MAC. Consequently, $\mathcal{F}$ has successfully broken the MAC scheme.

THE DISTINGUISHER $\mathcal{D}$. Next we construct a distinguisher $\mathcal{D}$ that, given an encryption-aided MAC forger $\mathcal{F}$, breaks the semantic security of the encryption scheme against chosen ciphertext attacks. That is, $\mathcal{D}$ takes for input a public key e of an asymmetric encryption scheme, an encryption $f = \textsc{enc}_e(N)$ where $N \in_{\mathrm{R}} \{0,1\}^k$, and a value N_1. In addition, $\mathcal{D}$ has access to a decryption oracle D that decrypts messages (different than f) according to the decryption key d^* that corresponds to e^*. (That is, $D(f) = \perp$. If $\hat{f} \neq f$ then $D(\hat{f}) = \textsc{dec}_{d^*}(\hat{f})$.) The goal of $\mathcal{D}$ is to distinguish between the case where $N_1 \in_{\mathrm{R}} \{0,1\}^k$ independently of N, and the case where $N = N_1$.

Distinguisher $\mathcal{D}$ runs forger $\mathcal{F}$ on input e, f. It relays $\mathcal{F}$'s decryption queries to D (ie, to $\mathcal{D}$'s decryption oracle). It answers $\mathcal{F}$'s MAC queries according to key N_1. That is, MAC query m is answered with $\textsc{mac}_{N_1}(m)$. Finally, if $\mathcal{F}$ outputs a successful forgery then $\mathcal{D}$ outputs '$N_1 = N$'. Otherwise $\mathcal{D}$ outputs 'N_1 is independent from N'.

Analysis of $\mathcal{D}$ is straightforward. Let ϵ_f denote the probability of successful forgery by $\mathcal{F}$, and let l be the number of MAC queries made by $\mathcal{F}$. Let ϵ_{MAC} be an upper bound on the probability of breaking the MAC scheme with up to l queries. Then, if $N = N_1$ then $\mathcal{D}$ outputs '$N_1 = N$' with probability ϵ_f. If N_1 is independent from N then $\mathcal{D}$ outputs '$N_1 = N$' with probability at most ϵ_{MAC}, otherwise $\mathcal{F}$ can be used to break the security of the MAC algorithm in use. $\square$

REMARKS. 1. Proposition 5 needs only a considerably weaker property than full-fledged security against chosen ciphertext attacks of the encryption scheme. That is, forger $\mathcal{F}$ (in the proof of Proposition 5), as well as distinguisher $\mathcal{D}$, do not need a full-fledged decryption oracle. Instead, they only need to know, given $\textsc{enc}_{e^*}(N^*)$ and a value v, the appropriate MAC (ie, $\textsc{mac}_{N^*}(v)$).
2. An alternative MT-authenticator, based on encryption schemes that are non-malleable against chosen ciphertext attacks, is described in [DDN]. Their construction does not use MAC schemes; instead it uses the 'full power' of the decryption oracle.

4 Key exchange protocols: Definitions and usage

In Section 3 we presented authenticators that treated each message separately. Such authenticators are not suited for authentication of a large number of messages exchanged between two parties since they involve a costly public-key operation (ie, encryption, decryption, signature or verification) *per message.* When transmitting many messages between two parties the following approach is usually taken: first have the parties engage in a key exchange protocol, where they obtain a common key known only to the two parties. Next, authenticate each message between the parties using a standard symmetric key message authentication (MAC) algorithm under the exchanged key. MAC performance is usually several orders of magnitude faster than the public key related operations. (In addition, some extra measure is needed in order to avoid replay of messages. This can be done through a shared state between the parties.)

Typically, several invocations of a key exchange protocol may occur between each two parties during the lifetime of the system. It is natural to consider each invocation of the key exchange protocol (together with the invocation of the code pertaining to the incoming and outgoing messages that are authenticated with the obtained key) as a separate session. In this and the following section we concentrate on this meaning of the term session. As usual, we would like to consider each session as an independent entity, as much as possible (see Section 2.2).

Below we define key-exchange protocols (Section 4.1), and show that the above approach to constructing authenticators is indeed secure (Section 4.2).

4.1 Key-exchange protocols: a definition

We present a definition of a secure key exchange (KE) protocol. We use the following standard paradigm for defining secure protocols: First we specify an ideal KE process. This ideal process captures our intuitive notion of 'the best we can expect' from a KE protocol. Next we say that a KE protocol (either in the authenticated or in the unauthenticated-links model) is secure if it emulates to the ideal KE process, as formalized in Definition 1.

Using this definitional approach we obtain, 'in one blow', definitions for secure KE protocols both in the authenticated and the unauthenticated-links models. It may seem that KE protocols in the authenticated-links model are of theoretical interest only. Yet, it will follow as an immediate corollary that if π is a secure KE protocol in the authenticated-links model, and $\mathcal{C}$ is an authenticator, then $\mathcal{C}(\pi)$ is a secure KE protocol in the unauthenticated-links model. This points to a very attractive design principle for secure KE protocols. See Section 5 for further details.

Although each exchange of a key involves only two parties, we model a KE protocol as an on-going multi-party process that involves all parties in the system. This captures the realistic (and often subtle) threats against a key exchange protocol where an attacker can use a corrupted party in order to attack an exchange between two other uncorrupted parties. In addition, we allow pairs of parties to have multiple keys exchanged between them. Each exchange of a key induces a separate session within each participating party.[13] We want the different sessions to be as independent of each other as possible. In particular, the compromise of a key exchanged and used in one session should not lead to the compromise of keys exchanged in other sessions. These and other characteristics (such as the ability of an attacker to destroy messages sent from one party to another) are captured in the ideal KE process, described below.

THE IDEAL KE PROCESS. There are n parties $P_1 ... P_n$, and an ideal KE adversary $\mathcal{S}$. We also imagine participation of a trusted party T. The computation consists of a series of activations of parties, made by $\mathcal{S}$. There are four types of activations:

I. Invoke P_i to establish a new key with P_j. The effect is that the value 'P_i `established key` (κ, s) `with` P_j' is added to P_i's output, where κ is a key chosen according to some predefined distribution.[14] The value s is a session ID that consists of a tuple (P_i, P_j, c, I), where c is the numeral of the session among all sessions established by P_i with P_j, and I is a symbol specifying that P_i is

[13] Typically, all the sessions are invocations of a single 'exchange subroutine'. (This subroutine is invoked whenever a party is prompted to exchange a key with another party.) Yet, we do not exclude protocols that invoke different subroutines for different sessions.

[14] The distribution of the established key is a parameter of the system. For instance, it may be that $\kappa \in_{\mathrm{R}} \{0,1\}^k$ where k is a security parameter.

an initiator in this exchange. The adversary learns only the value s, and does *not* learn the value κ. (We envision that the value κ is handed to P_i by the trusted party.) In addition, the value s is added to a global set I of incomplete sessions.

If an uncorrupted party establishes a key with a corrupted party (see item **IV** below), then we let the adversary choose the value of the key. This provision reflects the fact that we do not have security requirements from keys exchanges with corrupted parties. In addition it will be important for proving validity of known KE protocols (see Section 5).

II. Invoke P_j to establish key of session s with P_i. This activation is allowed only if the value s is currently in the set I and $s = (P_i, P_j, c, \text{I})$ for some P_i and c. The effect is that the value 'P_j `established key` $(\kappa, \hat{s})$ `with` P_i' is added to P_j's output, where κ is the same value that appears in the corresponding output of P_i, and $\hat{s} = (P_i, P_j, c, \text{R})$. Here the symbol R specifies that P_j is a responder in this exchange.[15] (Also here, we envision that the value κ is handed to P_j by the trusted party.) In addition, the value s is deleted from the set I of incomplete sessions.

III. Corrupt session s. This activation is of course valid only if s is a session ID that was in I at some point (or is currently in I). The effect is that the adversary learns the key κ that corresponds to s. In addition, the value '`Session` s `is corrupted`' is appended to the output of the initiator. If s is no longer in I then the value '`Session` $\hat{s}$ `is corrupted`' is appended also to the output of the responder. We stress that s is not deleted from I (in case that s is currently in I).

IV. Corrupt party P_i. The effect is that all the keys known to P_i become known to the adversary. In addition a value 'P_i `is corrupted`' is appended to P_i's output.

We allow the adversary to keep activating corrupted parties (in order to allow corrupted parties to exchange keys with uncorrupted ones.) Yet, we stick to the convention that the output of a corrupted party does not grow (ie, the last value in a corrupted party's output is the corruption notice).

As in Section 2.1, the global output of the ideal KE process is the concatenation of the cumulative outputs of all the parties, together with the output of the adversary. (The output of the adversary is a function of the information seen by the adversary throughout the computation, together with its random input.)

We use the following notation. Let $\text{ADV}_{\mathcal{S}}(r_{\mathcal{S}}, r_T)$ denote the output of ideal KE adversary $\mathcal{S}$ of a run on random input $r_{\mathcal{S}}$ and when the keys are chosen (by the trusted party T) using random input r_T. Let $\text{IDEAL}_{\mathcal{S}}(r_{\mathcal{S}}, r_T)_i$ denote the cumulative output of party P_i after an interaction with ideal KE adversary $\mathcal{S}$ and random inputs $r_{\mathcal{S}}, r_T$. Let $\text{IDEAL}_{\mathcal{S}}(r_{\mathcal{S}}, r_T) =$

$$\text{ADV}_{\mathcal{S}}(r_{\mathcal{S}}, r_T), \text{IDEAL}_{\mathcal{S}}(r_{\mathcal{S}}, r_T)_1 \ldots \text{IDEAL}_{\mathcal{S}}(r_{\mathcal{S}}, r_T)_n \ .$$

Let $\text{IDEAL}_{\mathcal{S}}()$ denote the random variable describing $\text{IDEAL}_{\mathcal{S}}(r_{\mathcal{S}}, r_T)$ when $r_{\mathcal{S}}, r_T$ are uniformly chosen.

Let $\text{AUTH}_{\pi,\mathcal{A}}()$ and $\text{UNAUTH}_{\pi,\mathcal{U}}()$ be defined as in Section 2. As there, we use the convention that whenever a party is corrupted, or a session within a party is corrupted, or a request to exchange a key is issued, an appropriate note is appended to the party's output.[16]

[15] We adopt the convention that in the ideal model the session IDs of the initiating and responding parties are identical, except for the I/R field. In particular, the initiator of the exchange always appears first.

[16] This recording of "important events" is crucial for our notion of emulation. In particular, it forces a secure KE protocol to actually exchange keys if the adversary delivers all messages faithfully; it also makes sure that a secure KE protocol maintains a session structure similar to the ideal KE process (ie, each exchanged key is associated with a separate session).

Definition 6 *Let π be a message-driven protocol for n parties. We say that π is a* secure KE protocol in unauthenticated networks *if for any* UM*-adversary $\mathcal{U}$ there exists an ideal KE adversary $\mathcal{S}$ such that*

$$\text{UNAUTH}_{\pi,\mathcal{U}}() \stackrel{c}{\approx} \text{IDEAL}_{\mathcal{S}}().$$

We say that π is a secure KE protocol in authenticated networks *if for any* AM*-adversary $\mathcal{A}$ there exists an ideal KE adversary $\mathcal{S}$ such that*

$$\text{AUTH}_{\pi,\mathcal{A}}() \stackrel{c}{\approx} \text{IDEAL}_{\mathcal{S}}().$$

REMARK. The ideal KE process does not take any input. This allows Definition 6 to consider only the empty input.

4.2 From key-exchange protocols to authenticators

We show an MT-authenticator that exchanges a single key (ie, maintains a single session) for the entire communication between each pair of parties. This can be generalized straightforwardly to the case of multiple keys (ie, multiple sessions) between each pair of parties.

Let KE be a key exchange protocol in the unauthenticated-links model. Then the MT-authenticator λ_{KE} proceeds as follows. First each party A invokes a copy of KE with each other party. Next, when invoked to send a message m to party B, party A increments a local counter (associated with party B), and sends $m, i, MAC_{K_{AB}}(m, i)$ to party B, where i is the current reading of the counter and K_{AB} is the obtained key for authenticating messages from A to B.[17] Upon activation with incoming message (m, j, v), from B, verify that $v = MAC_{K_{BA}}(m, j)$ and that no message with counter j or higher was received from B. If both verifications succeed then output 'A `received` m `from` B'.

Theorem 7 *Assume that* KE *is a key exchange protocol in the unauthenticated-links model, and that* MAC *is a secure MAC scheme. Then protocol λ_{KE} described above is a secure* MT*-authenticator.*

Proof (sketch): Let $\mathcal{U}$ be a UM-adversary that interacts with λ_{KE}. We construct an AM-adversary $\mathcal{A}$ such that

$$\text{AUTH}_{\text{MT},\mathcal{A}}() \stackrel{s}{=} \text{UNAUTH}_{\lambda_{\text{KE}},\mathcal{U}}() \ .$$

(Since protocol MT ignores its input, we consider only the empty input.) Adversary $\mathcal{A}$ proceeds similarly to adversary $\mathcal{A}_\pi$ in the proof of Theorem 4. As there, let $\mathcal{B}$ denote the event that some simulated party B' outputs 'B' `received` m `from` A'', and either A' wasn't activated for sending m to B', or B' has already had the same output before. As there, it is straightforward to see that if $\mathcal{A}_\pi$ could continue with the simulation even when event $\mathcal{B}$ occurs, then the simulation run by $\mathcal{A}$ is perfect and $\text{AUTH}_{\text{MT},\mathcal{A}}() \stackrel{d}{=} \text{UNAUTH}_{\lambda_{\text{KE}},\mathcal{U}}()$. It remains to show that event $\mathcal{B}$ occurs with negligible probability.

Assume event $\mathcal{B}$ occurs often. Then construct a forger $\mathcal{F}$ that forges the MAC scheme. Given a MAC-oracle, $\mathcal{F}$ performs a simulation of $\mathcal{U}$, where $\mathcal{F}$ simply runs the programs for all parties, with the following exception. $\mathcal{F}$ chooses an (ordered) pair of parties, A, B at random. First $\mathcal{F}$ runs a simulated execution of KE between A, B similarly to all other pairs of parties. Next, in all

[17] An immediate optimization is to let party A invoke the key-exchange protocol with party B only when it needs to send the first message to B.

Also, one can use the above protocol to have one key for the communication from A to B and a second key for the communication from B to A. Alternatively, one can have the same key for the communications in both ways. Both variants are secure.

the subsequent communication from A to B, $\mathcal{F}$ replaces the MACs that A sends B with MACs computed according to $\mathcal{F}$'s oracle. If at some point $\mathcal{U}$ delivers a MACed message (m, j) from A to B and A did not send this message then $\mathcal{F}$ outputs the message (m, j) and its MAC as forgery.

ANALYSIS OF $\mathcal{F}$. Informally, we want to say that: **(i).** If protocol KE is replaced by an ideal KE process then forger $\mathcal{F}$ succeeds. (This is so since in the ideal process the key established between A, B is unknown to $\mathcal{F}$. Thus $\mathcal{F}$'s view of the simulated interaction is distributed identically to its view of the idealized interaction.) **(ii).** Running KE is "equivalent" to running the ideal KE process. Therefore, $\mathcal{F}$ succeeds here too. Formalizing this informal argument, we proceed as follows.

(I). Given adversary $\mathcal{U}$, construct an adversary $\mathcal{U}_{\text{KE}}$ that works against protocol KE. $\mathcal{U}_{\text{KE}}$ will choose public and private keys for a set of simulated (tagged) parties $P'_1...P'_n$, and will run a full simulated interaction between $\mathcal{U}$ and $P'_1...P'_n$, with the following exception. $\mathcal{U}_{\text{KE}}$ will choose two parties A', B' at random; whenever $\mathcal{U}$ activates A' or B' then $\mathcal{U}_{\text{KE}}$ activates A, B in its actual environment, and reports the party's actions back to $\mathcal{U}$. If $\mathcal{U}$ asks A to send some MACed message then $\mathcal{U}_{\text{KE}}$ MACs this message according to random key that $\mathcal{U}_{\text{KE}}$ chooses independently. If $\mathcal{U}$ wants to fully corrupt A or B then $\mathcal{U}_{\text{KE}}$ fails. (We do not care about such failures since they do not occur when event $\mathcal{B}^*$ does. $\mathcal{B}^*$ is the event that $\mathcal{B}$ occurs w.r.t the pair A, B. Finally $\mathcal{U}_{\text{KE}}$ outputs whatever $\mathcal{U}$ does.

(II). Since KE is a secure KE protocol then there exists a simulator $\mathcal{S}$ that successfully imitates $\mathcal{U}_{\text{KE}}$ in the ideal KE model.

(III). Use $\mathcal{S}$ to construct another adversary, $\mathcal{U}_\lambda$, that works against λ_{KE}. $\mathcal{U}_\lambda$ is identical to $\mathcal{U}$ except that when $\mathcal{U}$ activates A or B then $\mathcal{U}_\lambda$ ignores $\mathcal{U}$ and instead follows the instructions of $\mathcal{S}$. (If event $\mathcal{B}^*$ is to occur, then the only instruction that $\mathcal{S}$ will generate is to give A, B a shared secret key.) We do not care what $\mathcal{U}_\lambda$ outputs, since we only want to simulate the interaction until the point where event $\mathcal{B}^*$ occurs, in case it does occur.

(IV). Assume that $\mathcal{F}$ runs $\mathcal{U}_\lambda$ instead of running $\mathcal{U}$. Then $\mathcal{F}$ succeeds whenever $\mathcal{B}^*$ occurs. This is so since now the view of $\mathcal{U}_\lambda$ when running within $\mathcal{F}$ is is identically distributed to its view of a real run of $\mathcal{U}_\lambda$.

(V). We now claim that if $\mathcal{F}$ does not succeed when running $\mathcal{U}$, then it is possible to distinguish between the output of protocol KE run with $\mathcal{U}_{\text{KE}}$ and the output of the ideal KE process with $\mathcal{S}$ (in contradiction to the assumption that the two runs are computationally indistinguishable). This is done as follows. Given an input string (that describes the output of an execution), distinguisher $\mathcal{D}$ runs $\mathcal{F}$ with a simulated MAC oracle (to which $\mathcal{D}$ chose the secret key), and with the exception that $\mathcal{F}$ now interacts with an adversary $\hat{\mathcal{U}}$ defined as follows: $\hat{\mathcal{U}}$ runs $\mathcal{U}$ except that the invocation of KE between A, B is performed according to the input string. If $\mathcal{F}$ outputs a successful forgery then $\mathcal{D}$ decides that the input string comes from $\mathcal{S}$. Otherwise the input string comes from $\mathcal{U}_{\text{KE}}$.

Analyzing $\mathcal{D}$, we show that if $\mathcal{D}$'s input comes from $\mathcal{S}$ then $\hat{\mathcal{U}}$ behaves like $\mathcal{U}_\lambda$, and if $\mathcal{D}$'s input comes from $\mathcal{U}_{\text{KE}}$ then $\hat{\mathcal{U}}$ behaves like $\mathcal{U}$. □

REMARK. Theorem 7 is stated for the case where only a single session (ie, a single KE invocation) exists between each two parties. It can be straightforwardly generalized to the case where multiple sessions co-exist between two parties. (New sessions are invoked by external requests from other (higher-layer) protocols) The difference in the proof is that instead of choosing two parties A, B, one will choose a specific session between A, B.

5 Key exchange protocols: Constructions

In this section we present some constructions of KE protocols. Here we finally put our new machinery in use. In fact, we use our machinery with the following 'twist'. In Section 4 we considered KE protocols as a tool for obtaining authenticators. Here we use authenticators (and, in particular, the authenticators constructed in Section 3) to construct KE protocols. That is, we first perform the much easier task of designing a KE protocol KE in the *authenticated-links* model. Next, we use any authenticator $\mathcal{C}$ to obtain a full-fledged authenticator $\mathcal{C}(\mathrm{KE})$ in the unauthenticated-links model.

An important observation is that using this method with the layered authenticators of Section 3 we are able to obtain KE protocols that underlie important practical key distribution protocols. This exposes a design principle that implicitly underlies those KE protocols, namely, the "separability" of the authentication mechanisms from the underlying basic key exchange mechanism.[18]

Substantiating this design principle, we note the following easy (but central) corollary:

Corollary 8 *Let* KE *be a secure KE protocol in authenticated networks, and let* $\mathcal{C}$ *be an authenticator. Then* $\mathcal{C}(\mathrm{KE})$ *is a secure KE protocol in unauthenticated networks.* □

We consider two well-known KE protocols in the authenticated-links model: the Diffie-Hellman protocol [DH], denoted DH, and an encryption-based protocol, denoted EB.

ENCRYPTION-BASED KE. First the parties choose private and public keys of any semantically secure encryption scheme with security parameter k. (We stress that here, in the authenticated-links model, simple semantic security against passive eavesdroppers is sufficient.) Let e_P, v_P denote party P's encryption and decryption keys. Next, upon activation for exchanging a key with party B, party A invokes the following two-party protocol $\hat{\mathrm{EB}}$. A chooses a key $\kappa \in_{\mathrm{R}} \{0,1\}^k$ sends $\mathrm{ENC}_{e_{\mathrm{B}}}(\kappa)$ to B, and lets κ be the established key. It is crucial that A immediately *erase* the random coins used to encrypt κ.[19] Upon receipt of $\mathrm{ENC}_{e_{\mathrm{B}}}(\kappa)$, party B lets κ be the established key.[20]

Proposition 9 *Protocol* EB *is a secure KE protocol in authenticated networks.*

Proof: Let $\mathcal{A}$ be an AM-adversary that interacts with EB. We construct an ideal KE adversary $\mathcal{S}$ such that $\mathrm{AUTH}_{\mathrm{EB},\mathcal{A}}() \overset{c}{\approx} \mathrm{IDEAL}_{\mathcal{S}}()$. Adversary $\mathcal{S}$ chooses private and public keys for a set of simulated parties $P'_1 ... P'_n$ and runs a simulated interaction between $\mathcal{A}$ and $P'_1 ... P'_n$. Whenever a simulated party A' sends an encrypted key to simulated party B', adversary $\mathcal{S}$ invokes party A, in the ideal model, to establish a new key with party B. Whenever simulated party B' accepts an encrypted key from A', $\mathcal{S}$ invokes B to establish a key of the corresponding session with A. Since $\mathcal{A}$ is an AM-adversary we are guaranteed that $\mathcal{S}$'s operations are always legal. When $\mathcal{A}$ corrupts either a session or a party, $\mathcal{S}$ performs the same corruption in the ideal model and reports the newly learned information back to $\mathcal{A}$, together with the relevant information from the simulated execution. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

[18] We stress that not all proposed key exchange protocols in the literature follow this modular approach (see [MVV]). However, our work demonstrates a clear advantage of "separable" protocols where the design and analysis of the protocol are made possible, and greatly simplified, by this property.

[19] If the parties do not trust each other to properly erase their random inputs then standard semantically secure encryption does not suffice. In some cases non-committing (ie, adaptively secure) encryption will suffice [CFGN]. See details there.

[20] Typically in practice, the exchange is bi-directional. That is, both A and B send encrypted values to each other, and the established key is some function (say, the bitwise exclusive or) of the two values. This is done so that parties do not have to trust each other to properly choose their keys. For simplicity we concentrate here on the single-sided protocol.

Demonstrating that $\text{AUTH}_{\text{EB},\mathcal{A}}() \stackrel{c}{\approx} \text{IDEAL}_{\mathcal{S}}()$ is done via a reduction to the semantic security of the encryption scheme, by defining the appropriate $l+1$ hybrid distributions where l is the number of keys exchanged. We omit further details. □

DIFFIE-HELLMAN KE. It is assumed that a large safe prime p and an element g of multiplicative order $(p-1)/2 \bmod p$ are known to all parties. (p is safe if $(p-1)/2$ is a prime). Upon activation for exchanging a key with party B, party A invokes the following two-party protocol $\widehat{\text{DH}}$. (Confusingly, this protocol is often called "unauthenticated DH".) A chooses an element $x \in_{\text{R}} \mathbb{Z}_p^*$ and sends $g^x (\bmod\ p)$ to B. Upon receipt of g^x from A, party B chooses $y \in_{\text{R}} \mathbb{Z}_p^*$, sends $g^y (\bmod\ p)$ to A and lets the established key be $(g^x)^y (\bmod\ p)$. Upon receipt of g^y from B, party A lets the established key be $(g^y)^x (\bmod\ p)$. It is crucial that both A and B *erase* the secret exponents x, y once the keys are established, otherwise the protocol is not known to be secure against adversaries that choose the corrupted parties in an adaptive way.

Proposition 10 *Protocol* DH *is a secure KE protocol in authenticated networks.*

Proof: Let $\mathcal{A}$ be an AM-adversary that interacts with DH. Construction of the ideal KE adversary $\mathcal{S}$ is identical to the proof of Proposition 9, with the exception that here the simulated parties run protocol DH. Demonstrating that $\text{AUTH}_{\text{DH},\mathcal{A}}() \stackrel{c}{\approx} \text{IDEAL}_{\mathcal{S}}()$ is done via a reduction to the DH Indistinguishability assumption, in the same way as in the proof of Proposition 9. (The DH Indistinguishability assumption states that $g^x, g^y, g^{xy} \stackrel{c}{\approx} g^x, g^y, g^z$, where $x, y, z \in_{\text{R}} \mathbb{Z}_p^*$ and p, g are known. For discussion and more details on this assumption see, for instance, [Ca2, NR].) □

Let $\mathcal{C}_{\text{SIG}}$ and $\mathcal{C}_{\text{ENC}}$ be the layered authenticators based on MT-authenticators λ_{SIG} and λ_{ENC}, respectively (see Section 3). Then, it follows from Corollary 8 that both $\mathcal{C}_{\text{SIG}}(\text{DH})$ and $\mathcal{C}_{\text{ENC}}(\text{DH})$ are secure KE protocols in unauthenticated networks. Protocol $\mathcal{C}_{\text{SIG}}(\text{DH})$ is very similar to a signature-based KE protocol used in an ISO standard [ISO, MVV]. $\mathcal{C}_{\text{ENC}}(\text{DH})$ is very similar to the SKEME KE protocol [Kra].

REMARKS. 1. Naive application of $\mathcal{C}_{\text{SIG}}$ and $\mathcal{C}_{\text{ENC}}$ to protocolDH results in a 6-move protocol. Yet it is straightforward to see that the 6 moves can be 'collapsed' to 3 moves by 'piggybacking' messages of one protocol on the other. This is done as follows. Let λ be either one of λ_{SIG} or λ_{ENC}, let λ_{I} be the invocation of λ that authenticates the initiator's message, and let λ_{R} be the invocation of λ that authenticates the responder's message. Then, in the first message of the combined protocol the initiator sends the first message of λ_{I} together with the second message of λ_{R}. (The first message of λ_{R} is not sent. It is implicit in the next step.) In the second step of the combined protocol the responder sends the second message of λ_{I} together with the third message of λ_{R}. In the third step of the combined protocol the initiator sends the third message of λ_{I}.

2. In a full version of this work we will extend the analysis of the DH exchange to deal with issues like perfect-forward-secrecy and anonymity.

References

[Bea] D. BEAVER, "Foundations of Secure Interactive Computing", *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[BCK] M. BELLARE, R. CANETTI AND H. KRAWCZYK, "A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols", Available at http://www-cse.ucsd.edu/users/mihir and at the Theory of Cryptography Library, http://theory.lcs.mit.edu/~tcryptol, March 1998.

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

[BR1] M. Bellare and P. Rogaway, "Entity authentication and key distribution", *Advances in Cryptology – Crypto* 93 *Proceedings*, Lecture Notes in Computer Science Vol. 773, D. Stinson ed., Springer-Verlag, 1993.

[BR2] M. Bellare and P. Rogaway, "Provably secure session key distribution– the three party case," *Proceedings of the* 27th *Annual Symposium on the Theory of Computing*, ACM, 1995.

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

[BJM] 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.

[BCG] M. Ben-Or, R. Canetti and O. Goldreich, "Asynchronous Secure Computations", *Proceedings of the* 25th *Annual Symposium on the Theory of Computing*, ACM, 1993.

[BGW] M. Ben-Or, S. Goldwasser, and A. Wigderson, Completeness theorems for non-cryptographic fault-tolerant distributed computations, *Proceedings of the* 20th *Annual Symposium on the Theory of Computing*, ACM, 1988.

[BGH+] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung, "Systematic design of two-party authentication protocols," *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[BAN] 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.

[Ca1] R. Canetti, "Modular Composition of Secure Multiparty Protocols", Available at the Theory of Cryptography Library, http://theory.lcs.mit.edu/ tcryptol, 1998.

[Ca2] R. Canetti, "Towards realizing random oracles: Hash functions that hide all partial information", *Advances in Cryptology – Crypto* 97 *Proceedings*, Lecture Notes in Computer Science Vol. 1294, B. Kaliski ed., Springer-Verlag, 1997.

[CHH] R. Canetti, S. Halevi and A. Herzberg, "How to Maintain Authenticated Communication", *16th PODC,* 15-25, 1997.

[CFGN] R. Canetti, U. Feige, O. Goldreich and M. Naor, "Adaptively Secure Computation", *Proceedings of the* 28th *Annual Symposium on the Theory of Computing*, ACM, 1996. Fuller version MIT-LCS-TR #682, 1996.

[CCD] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols", *Proceedings of the* 20th *Annual Symposium on the Theory of Computing*, ACM, 1988.

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

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

[DDN] D. Dolev, C. Dwork and M. Naor, "Non-malleable cryptography", TR CS95-27, Weizmann Institute. Preliminary version in *Proceedings of the* 23rd *Annual Symposium on the Theory of Computing*, ACM, 1991.

[GMW] O. Goldreich, S. Micali and A. Wigderson, "How to play any mental game, or A completeness theorem for protocols with honest majority," *Proceedings of the* 19th *Annual Symposium on the Theory of Computing*, ACM, 1987.

[GMR] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing*, Vol. 17, No. 2, April 1988, pp. 281–308.

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

[HC] D. Harkins and D. Carrel, ed., "The Internet Key Exchange (IKE)", *Internet Draft*, `draft-ietf-ipsec-isakmp-oakley-06.txt`, February 1998.

[Kra] 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.

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

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

[MR] S. Micali and P. Rogaway, "Secure Computation", Manuscript, 1992. Preliminary version in *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[NR] M. Naor and O. Reingold, "Efficient cryptographic primitives based on the decisional Diffie-Hellman assumption", *Proceedings of the* 38th *Symposium on Foundations of Computer Science*, IEEE, 1997.

[NY] M. Naor and M. Yung, "Public key cryptosystems provably secure against chosen ciphertext attacks", *Proceedings of the* 22nd *Annual Symposium on the Theory of Computing*, ACM, 1990.

[NS] 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.

[Ra] C. Rackoff, "Some definitions, protocols and proofs about secure authentication," *IBM CASCON* 92.

[SR] V. Shoup and A. Rubin. Session key distribution using smart cards. *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[RS] C. Rackoff and D. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack", *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[SNS] J. Steiner, C. Newman and J. Schiller, "Kerberos: an authentication service for open network systems," *Proceedings of the USENIX Winter Conference*, 1988, pp. 191–202.

[Y] A. Yao, Protocols for Secure Computation, In *Proc.* 23*th Annual Symp. on Foundations of Computer Science*, pages 160–164. IEEE, 1982.