On Formal Models for Secure Key Exchange*
(version 4)

Victor Shoup
IBM Zurich Research Lab, Säumerstr. 4, 8803 Rüschlikon, Switzerland
`sho@zurich.ibm.com`

November 15, 1999

Abstract

A new formal security model for session key exchange protocols is proposed, and several efficient protocols are analyzed in this model. Our new model is in the style of multi-party simulatability: it specifies the service and security guarantees that a key exchange protocol should provide to higher-level protocols as a simple, natural, and intuitive interface to which a high-level protocol designer can program. The relationship between this new model and previously proposed models is explored, and in particular, several flaws and shortcomings in previously proposed models are discussed. The model also deals with anonymous users—that is, users who do not have public keys, but perhaps have passwords that can be used to authenticate themselves within a secure session.

*First version (IBM Research Report RZ 3120), April 1999; version 2 (October 21, 1999) is a substantial revision; versions 3 (October 29, 1999) and 4 are minor revisions.

Contents

1 Introduction

In this paper, we investigate formal models of security for session key exchange protocols. A session key protocol allows two users to exchange a secret key. The most important—and perhaps the only significant—application of a session key exchange protocol is to implement a secure session protocol, which in effect provides a secure (private, authenticated), bi-directional channel between the two users. A user may establish session keys with many other users, possibly in a concurrent fashion. The main security goals, intuitively speaking, are that session keys should be random and independent of one another, and that a user really establishes a key with the user he "thinks" he is, and not with some other user.

There are two basic settings in which key exchange protocols are usually considered. In both settings, there is a trusted third party (TTP). The only difference is whether the TTP is "on line," i.e., involved in every key exchange, or "off line," is only needed to register users of the system, but does not participate in the key exchange protocol itself. In the on-line TTP setting, one uses symmetric key cryptography; Kerberos [SNS88] is an example of a session key exchange protocol in the on-line setting; the TTP is usually called a *key distribution center* in this setting. In the off-line TTP setting, one uses public key cryptography; the Secure Socket Layer (SSL) provides an example of a session key exchange protocol in the off-line setting; the TTP is usually called a *certificate authority* in this setting.

We propose a new model of security for key exchange protocols, and analyze the security of a number of protocols in this model. Our model is general enough to be applied in either the on-line or off-line setting. However, all of the examples of protocols we consider are in the off-line setting.

Despite the superficial simplicity of session key exchange protocols, it is all too easy to design protocols with security weaknesses. Indeed, the history of this subject is littered with the carnage of broken protocols. Typically, design flaws arise either by not carefully specifying what an attacker is able to do, or by not making the security goals precise, or by not making clear the requirements of the cryptographic primitives. Formal modeling, such as we do here, serves to prevent such design flaws.

Our work is inspired by the work of Bellare, Canetti, and Krawczyk [BCK98], which is grounded in the multi-party simulatability tradition (see, e.g., [Bea91, Can95]). This approach seems very attractive, because it specifies the *service* a session key protocol should provide to a higher-level protocol, rather than getting mired in the implementation details of session key protocols themselves, many of which are irrelevant. This type of definition yields a simple, natural, abstract interface to which a high-level protocol designer can program, without worrying about implementation details. Also, because of the simplicity and naturalness of the interface, it is easy to reason about the security properties of high-level protocols. Moreover, security in this model implies security against a whole range of specific attacks.

1.1 Our contributions

We summarize our main contributions:

- We present a detailed security model that addresses some technical shortcomings in [BCK98], and that extends and enriches their model.
- Our model takes into account the role of the *certificate authority*, making our trust assumptions explicit; this is essential in order to model a class of attacks which we call *Public Key Infrastructure (PKI) attacks* (see §2).

- Our model takes into account the *ordinary usage* of session keys in *arbitrary* higher-layer protocols, and how this usage is interleaved with the ongoing execution of the key exchange protocol, possibly interfering with the correct functioning of the key exchange protocol. That is, our definition of security has a built-in "protocol composition" theorem that *a priori* rules out subtle problems that can arise when composing a key exchange protocol with an arbitrary higher-level protocol. In particular, our model allows us to represent a class of attacks which we call *protocol interference* attacks (see §2).

- We classify and study in detail three different modes of corruption:

  static corruptions the adversary may operate under a number of aliases, but cannot corrupt honest users;

  adaptive corruptions the adversary can choose to corrupt an honest user, obtaining that user's *long-term secret* only;

  strong adaptive corruptions the adversary can choose to corrupt an honest user, obtaining all of that user's internal data that has not been explicitly *erased*.

  We give what we think are natural and useful definitions of security against these three different corruption modes. Our models for adaptive and strong adaptive corruptions capture the notion of *forward security* (a.k.a., *perfect forward secrecy*).

- We study the relationship between our definitions of security and those of Bellare and Rogaway [BR95] (see also [BJM97, BM97]). In particular, we show that their notion of security (with one essential fix) is equivalent to our notion of security against an adversary that makes static corruptions only, despite the fact that in their model the adversary may make strong adaptive corruptions.

- In addition to definitions, we give *many* examples of key exchange protocols and proofs of their security. These examples serve to highlight some of the subtle differences between modes of corruptions.

- We briefly sketch formal definitions and implementations of secure sessions, which can be built on top session key protocols. Arguably, the notion of a secure session protocol is more fundamental than that of a key exchange protocol, i.e., that the latter is merely one tool (among others) that one needs to build the former. Nevertheless, it appears that most of the subtlety in designing a secure session protocol already occurs in the design of the underlying key exchange protocol, so it seems worthwhile to study key exchange in isolation. However, in formulating definitions of security for key exchange, our main motivation is to get a definition that is useful in building a secure session protocol.

- We propose a formal model for session key exchange involving *anonymous users,* i.e., users who do not have a certificate or have otherwise registered with the TTP. Once a secure session is established in this setting, the anonymous user can authenticate his identity using a password.

1.2 Relation to previous work

There is a vast literature on this subject, which we shall not attempt to survey here. We refer the reader to [MvOV97, Chapter 12] for a more extensive historical discussion. We mention here just a few of the articles that are most relevant to this paper.

The seminal paper in this field is of course that of Needham and Schroeder [NS78]. This work was in the on-line TTP setting. However, one of the protocols in their paper was subsequently found to be flawed (see, e.g., [DS81]). Subsequent to [NS78], many other protocols have been proposed, many of which were also later found to be flawed.

Because of the subtlety of the flaws that can arise in key exchange protocols, formal logics have been developed (see, e.g., [BAN90]) that can help in finding protocol flaws. These formal methods, however, do not appear to give any meaningful security guarantees that can be used in the analysis of higher-level protocols that use the session key.

The paper of Bird, *et al.* [BGH+91] broke new ground by pointing out a class of subtle attacks called *interleaving attacks* which can arise when users are running several instances of a protocol in parallel. This work was in the on-line TTP setting.

The *station-to-station (STS) protocol* was introduced in the paper of Diffie, *et al.* [DvOW92]. This paper presents a session key exchange protocol based on the classical Diffie-Hellman key exchange protocol [DH76] (which establishes a long-term pair key, rather than a session key). The authors carry out a rather informal security analysis, and point out numerous pitfalls and attacks one should worry about. As we point out in §2, STS is vulnerable to PKI, protocol interference, as well as interleaving attacks.

Bellare and Rogaway [BR95] proposed a formal model of security for authenticated key exchange protocols, again in the on-line TTP setting. Their work represents the first attempt to lay a firm foundation for the analysis of key exchange protocols. Their formal model was subsequently adapted to the public key setting by Blake-Wilson, *et al.* [BJM97, BM97]. The definitions of security here seem fairly compelling, but yet, they also seem a bit technical and low level, and it is not at all clear what implications these definitions have for higher-level protocols that use the session keys. In fact, the definition of security in the Bellare-Rogaway model is flawed, in that it does not allow one to model protocol interference attacks. We discuss this point in §15.

More recently, Bellare, Canetti, and Krawczyk [BCK98] have proposed a quite different approach to formal security models for key exchange in the off-line TTP setting. This approach is similar to the simulation-based approach taken in the area of multi-party computation. One first defines an *idealized* version of a session key protocol, in which pairs of users can "magically" generate a shared random session key. Then to prove a *real world* protocol is secure, one shows that any real world adversary is constrained to behave essentially like an adversary operating in the ideal world.

The paper [BCK98] only considers adversaries that make what we have called *strong adaptive corruptions*. As already mentioned, we also study *static corruptions* and *adaptive corruptions.*

Certainly, the *static corruption* case is the simplest, most basic case, and deserves to be studied by itself.

Arguably, long-term secrets are in practice the most vulnerable secrets in the system; in a typical setting, they are stored on disk, perhaps protected by a password. Ephemeral data is much more difficult for an attacker to obtain. Therefore, it seems worthwhile to study the case of *adaptive corruptions* by itself, and to see what type of security guarantee we can achieve when the adversary is limited in this natural way. Also, this type of corruption model is more in line with the traditional study of key exchange protocols.

One criticism we have of [BCK98] is that, like [BR95], it is still a somewhat technical, low-level definition, and it is not at all clear what security properties higher-level protocols enjoy. Indeed, like [BR95], it appears to us that their definition does not properly model ordinary key usage and protocol interference attacks. We discuss this in more detail in §16. Aside from this, it is not at all clear what form a "protocol composition" theorem would take in their model. This is more a philosophical criticism than a technical one; however, we would argue that the whole point of making such a simulation-based definition is that such implications should be *built in* to the definition. In contrast, our definition comes with a "protocol composition" theorem literally built in.

Another shortcoming of the definition in [BCK98] that we discuss in §16 is that it offers no guarantee of forward security for established keys in the face of strong adaptive corruptions. Or at least, that appears to be the intention—as we discuss in §16, the intention is not really clear. Although their is nothing wrong with such a definition, it unfortunately rules out the possibility of building a secure session protocol (with private channels) that withstands strong adaptive corruptions on top of a protocol that only satisfies such a definition.

Further, their model does not give any account of the behavior of the certificate authority and of the distribution of public keys. Rather, all public keys for all users are generated and distributed to all users in an idealized initial set-up phase. In contrast, we explicitly model the role of the certificate authority. We believe this to be important, for three reasons. First, without this, one cannot represent PKI attacks. Second, in practice, certificates are typically delivered within the protocol itself, which could add to the round complexity of a protocol; because of this, the idealized initial set-up phase can obscure the true round complexity of a protocol. Third, it turns out that one can design quite efficient protocols based on the weakest possible trust assumption for the certificate authority—indeed, it seems that there is no point in assuming anything about the certificate authority beyond its ability to properly check the identity of a user.

The paper [BCK98] also advocates a "modular" approach to session key protocol design in which one implements a session key protocol on top of a communication network with ideal "authenticated links," and then implements an authenticated link network on top of a "raw" network without authenticated links. In contrast, we work exclusively in the "raw" network model. Our reason for this is that the "authenticated links" model somewhat obscures the true round and computational complexity of session key protocols, and more importantly, it also rules out certain very efficient protocols that do not arise from such a modular design approach. Although [BCK98] define security in the "raw" model as well as the "authenticated links" model, they do not consider any examples of protocols designed directly in the "raw" model. In contrast, we present several quite interesting protocols that exist only in the "raw" model.

Finally, another problem with [BCK98] is that both of the protocols presented and claimed to be secure (in the authenticated links model) actually are not, and apparently cannot be secure under any reasonable simulation-based definition of security.

As already mentioned, we propose a formal model for session key exchange involving *anonymous users*. In many situations, one of the two users in a key exchange protocol may not have a certificate. This already happens in SSL, and in fact, at the time of this writing, the vast majority of secure sessions established on the Internet are between a *server*, who has a certificate, and a *client* (the anonymous user), who does not. We show how our formal model can be easily adapted to deal with this situation, and present and analyze several protocols that work in this setting.

A server who establishes a secure session with an anonymous client has no idea who he is talking to. It may therefore be necessary for the client to authenticate himself to the server by means of a password. This is trivial to do in our model: having established a secure session, the client simply passes his password through the secure channel to the server. Our definition of security essentially guarantees everything one could possibly hope for in this setting, in particular, protection against off-line password guessing attacks, and against session "hijacking."

The problem of password-based authentication and key exchange itself has a long history; see, e.g, [BM92, GLNS93, HK98, Boy99]. To a large degree, our work generalizes and extends all of the previous work on this topic. Moreover, our work provides a formal model in which one can analyze protocols, like SSL, that yield a more flexible and modular approach to designing protocols between servers and anonymous clients: first establish a secure session between anonymous client and server, and then simply run other protocols like "telnet" or "FTP"—that may or may not require a password (or passwords)—on top of this secure session.

The current paper is a significantly revised version of [Sho99]. There are many fairly minor, technical changes. Most of the changes made here both simplify and "loosen" the definition of security, in an attempt to get at the "core" security issues. The most significant change is the treatment of strong adaptive corruptions. The paper [Sho99] already deals with such corruptions, but in a somewhat different way. Although the definitional approach in [Sho99] is workable, it is somewhat more cumbersome (and more restrictive) than the approach taken here.

1.3 Outline

Here is a guide to the rest of the paper.

- In §2, we discuss protocol interference and PKI attacks.
- In §3, we present our formal security model, restricted to the case of adversaries who statically corrupt users.
- In §4, we discuss in some detail the principle application of a key exchange protocol, namely, a secure session protocol. In particular, we sketch a formal simulation-based definition of security for a secure session protocol.
- In §5, we discuss the cryptographic primitives we need: secure signatures, non-malleable public-key encryption, and the Decisional Diffie-Hellman assumption. Readers already familiar with these can safely skip this section.
- In §6, we describe the precise role of the certificate authority in the protocols we present.
- In §7, we present and prove the security of a Diffie-Hellman based key exchange protocol **DHKE**.
- In §8, we present and prove the security of a public key encryption based key exchange protocol **EKE**.
- In §9, we discuss an extension to our security model that accommodates anonymous users, including a discussion of applications to secure sessions and password-based authentication in this setting. In particular, we present protocol **A-DHKE**, which extends protocol **DHKE** to anonymous users, and protocols **A-EKE-1** and **A-EKE-2**, which extend protocol **EKE**.
- In §10, we present a formal security model for key exchange that deals with adaptive corruptions, including a discussion of secure sessions and anonymous users in this corruption scenario.
- In §11, we re-examine protocols **DHKE** and **EKE** in the face of adaptive corruptions, showing that they are insecure in this scenario.
- In §12, we present several variants (**DHKE-1**, **DHKE-3**, **DHKE-3**) of protocol **DHKE** that are secure against adaptive corruptions, including variants (**A-DHKE-1**, **A-DHKE-3**) that deal with anonymous users.

- In §13, we present a variant **EKE-1** of protocol **EKE** that is secure against adaptive corruptions.
- In §14, we present a formal security model for key exchange that deals with strong adaptive corruptions, including a discussion of secure sessions and anonymous users in this corruption scenario.
- In §15, we compare our model of security with that of Bellare and Rogaway [BR95].
- In §16, we give a technical critique of the security model of Bellare, Canetti, and Krawczyk [BCK98].
- In §17, we make some concluding remarks.

2 Protocol Interference and PKI attacks

To motivate certain aspects of our new formal model, we will discuss two classes of subtle attacks: *protocol interference attacks*, and *PKI attacks*.

Protocol interference attacks are those where the seemingly benign use of a session key in a higher level protocol can interfere with the proper working of the session key protocol itself. This generalizes the interleaving attack of Bird, et al. [BGH+91].

PKI attacks involve adversaries who "hijack" honest users' public keys, obtaining certificates on an honest user's public key but with an identity determined by an adversary.

We will illustrate these attacks on the classic STS protocol.

The basic STS protocol uses a group G of order q and with generator g.

$A \to B: \ g^x$,
where $x \in \mathbf{Z}_q$ is random.

$B \to A: \ g^y, E_K(sig_B(g^x, g^y))$,
where $y \in \mathbf{Z}_q$ is random.

$A \to B: \ E_K(sig_A(g^x, g^y))$.

Here, $K = g^{xy}$, and E is a symmetric key cryptosystem. Before accepting, both A and B validate all the signatures. The key K is the session key.

Let us assume that the certificates of A and B are publicly available, and that the group G is described in, say, A's certificate.

As pointed out in [DvOW92], if we remove the encryptions on the signatures, then the protocol becomes insecure. We recall here the attack. Consider an adversary controlling a different identity $\tilde{A}$. Without the encryption on the last message, $\tilde{A}$ could generate for himself a signature $sig_{\tilde{A}}(g^x, g^y)$. This would result in the unacceptable situation where B "thinks" he is talking to $\tilde{A}$, but in fact shares a key with A, who "thinks" he is talking to B.

One criticism of STS is that it uses the resulting session key within the protocol itself. Not only does this leak partial information about the session key prematurely, but can lead to the phenomenon we called protocol interference above. In fact, the adversary can still carry out the same attack above, even with the encryptions. Suppose A has terminated the protocol and generated its last message. Now suppose that before A's response is ever delivered to B, the adversary interacts with the higher-level protocol using A's session key. Just suppose that in this higher level protocol, the adversary could convince A to compute an encryption $E_K(msg)$ of a message msg of the adversary's choice, and say that $msg = sig_{\tilde{A}}(g^x, g^y)$. Having obtained this encryption, the adversary forwards it to B, and we have successfully carried out the attack.

We can achieve the same result with a PKI attack. Suppose that the adversary can convince a relevant certificate authority to bind A's public key to $\tilde{A}$'s identity. Now, $\tilde{A}$ may have all the relevant documents to prove to the certificate authority's satisfaction that he "really is" $\tilde{A}$. If certificates are exchanged as part of the protocol, then the adversary can replace A's certificate with $\tilde{A}$'s.

The reader might object: do not certificate authorities verify not only the person is who he says he is, but that he "knows" the corresponding private key—by demanding, for example, a signature on a test message? Well, who knows what certificate authorities really do. So it seems better not to depend on this. Moreover, even if the authority makes such a check, it is not entirely clear how to analyze exactly what this buys us in terms of provable security properties using standard definitions. Anyway, by appropriately modifying the protocol, it is easy enough to defend against such PKI attacks, without making special assumptions about the certificate authority.

Besides protocol interference and PKI attacks, this protocol can also be attacked by a standard interleaving attack, as follows. The attacker can take the encrypted signature output by B in the second flow, and feed this back to B in the third flow. Thus, A will think he is talking to B, while B will think he is talking to another instance of himself. Note that for this attack to work, A and B must work with the same group parameters G and g.

3 Formal Security Model—The Static Corruption Case

We present our formal notion of security, beginning with *static corruptions*, i.e., adversaries that make their decision as to whom to corrupt independently of the network traffic (but otherwise are fully adaptive in everything else they do). In this case, such statically corrupted users do not explicitly exist in the model: they are all just absorbed into the adversary.

Since we want to let the adversary have arbitrary control over the network, we also eliminate the network: the adversary *is* the network. Moreover, it is the adversary that drives everything forward—all other players (the users and the TTP) are completely passive, and perform only the actions that the adversary instructs them to.

Security is defined via *simulation*, as follows.

We first define an *ideal world model* in which all the adversary can do is create and "connect" instances of users according to some intuitive and natural rules, whereby these user instances obtain random session keys that are hidden from the adversary. User instances that are "connected" share a common key, but keys are otherwise uncorrelated. As soon as a user instance obtains a key, he may begin to use it. For example, the user instance might encrypt messages with the key using a very good cipher or a very bad cipher or it might simply divulge the key. We place absolutely no restrictions on the use of a key; however, the adversary learns no more information about the key other than what is leaked through its use, and this information does not affect the orderly establishment of connections. In the ideal world, there is no TTP, nor are there certificates, signatures, encryptions, or even protocol message flows. This is all abstracted away, so that all that remains is the abstraction of the *service* a session key protocol is supposed to provide to a higher-lever protocol.

We then define a *real world model* that describes what adversaries can and cannot do in real life. This includes all the messy details of the TTP, certificates, etc., For both real-world and ideal-world adversaries, a transcript is generated that logs all important events as they happen. Security means that for every real-world adversary, there exists a corresponding ideal-world adversary, such that the transcripts that these two adversaries generate are computationally indistinguishable.

Simulatability is this sense is a very powerful notion. It implies that a high-level protocol designer can design and analyze his protocols *as if* they were running in the *ideal world*. As a general principle, whatever security properties one can prove about a high-level protocol running in the ideal world immediately transfer to the real world.

3.1 The ideal system

We now describe the workings of the ideal system. The basic idea is fairly natural and intuitive; however, it is important to specify all the rules of the game quite precisely, and so unfortunately, the details may at first sight seem somewhat legalistic.

We have a set of (honest) *users* U_i, indexed $i = 1, 2, \ldots$. Each user U_i may have several *user instances* I_{ij}, for $j = 1, 2, \ldots$.

Remark 1 *One might think of i as an IP address, and j as a port number. A session key can be thought of as securing a connection between two IP address/port number pairs.*

There is also an *adversary*. The adversary plays a game. Conceptually, it is convenient to think of the adversary's opponent as the *ring master*[1] whose job it is to generate certain random variables, and to enforce certain global consistency constraints. The adversary plays this game by issuing a sequence of operations to the ring master. There are six types of operations: *initialize user*, *initialize user instance*, *abort session*, *start session*, *application*, and *implementation*. We explain in turn how each of these operations work. As we shall see, the *application* operation is the only operation in which the ring master gives the adversary any information. Note that it is the adversary that drives the game forward—the uncorrupted parties and the ring master are purely passive, and simply react to the adversary's operations. Also note that all operations are performed sequentially and atomically.

3.1.1 Initialize user

This operation takes the form

$$(\textsf{initialize user}, i, ID_i).$$

This operation assigns the identity ID_i to user U_i. ID_i may be any bit string, subject to the restriction that this identity has not already been assigned to another user. Also, the *initialize user* operation may only be applied to users that have not already been previously initialized.

3.1.2 Initialize user instance

This operation takes the form

$$(\textsf{initialize user instance}, i, j, role_{ij}, PID_{ij}).$$

A user instance I_{ij} is specified, along with a value $role_{ij} \in \{0, 1\}$, as well as a partner identity PID_{ij}. User U_i must have been previously initialized, but I_{ij} should not have been previously initialized.

[1] Following circus terminology.

After execution of this operation, we say that user instance I_{ij} is *active*, and remains active until the execution of either an *abort session* or *start session* operation on I_{ij}.

Remark 2 *Intuitively, PID_{ij} represents the identity of the user that I_{ij} wants to talk to. The value $role_{ij}$ identifies which of two roles the user instance is to have in establishing a connection. We do not assign any meaning to this role—it is only a technical, symmetry breaking device. See also points (11) and (12) in §3.4.*

3.1.3 Abort session

This operation takes the form

$$(\mathsf{abort\ session}, i, j).$$

An *active* user instance I_{ij} is specified.

Remark 3 *Intuitively, this represents a failed attempt to establish a connection.*

3.1.4 Start session

This operation takes the form

$$(\mathsf{start\ session}, i, j, \mathit{connection\ assignment}\ [\ , \mathit{key}\ ]\ ).$$

An *active* user instance I_{ij} is specified.

The *connection assignment* specifies how the session key K_{ij} for user instance I_{ij} is generated. This connection assignment is one of the following:

- create,
- $(\mathsf{connect}, i', j')$, or
- compromise.

The optional *key* field is present in the *start session* operation only if the *connection assignment* is compromise.

Session keys are bit strings all of some agreed upon length. The session key K_{ij} is determined according to the connection assignment as follows.

create: the ring master creates K_{ij} as a random bit string.

$(\mathsf{connect}, i', j')$: In this case, the ring master sets K_{ij} equal to $K_{i'j'}$.

compromise: The ring master sets K_{ij} to *key*, the optional optional field in the *start session* operation.

There are rules governing the legality of these assignments. To describe these rules succinctly, we make the following definition. We say that two initialized user instances I_{ij} and $I_{i'j'}$ are *compatible* if

- $PID_{ij} = ID_{i'}$,
- $PID_{i'j'} = ID_i$, and

- $role_{ij} \neq role_{i'j'}$.

Now we present the rules governing the choice of connection assignments.

C1 The connection assignment create is always legal. When this *start session* operation completes, we say that I_{ij} is *isolated* (see rule **C2** below).

C2 The connection assignment $(\mathsf{connect}, i', j')$ is legal if $I_{i'j'}$ is a user instance that is still *isolated* (see rule **C1** above), and is *compatible* with I_{ij}. When this *start session* operation completes, $I_{i'j'}$ is no longer *isolated*.

C3 The connection assignment $\mathsf{compromise}$ is legal provided PID_{ij} is not assigned to a user.

The following definition will be useful later. If the connection assignment is $(\mathsf{connect}, i', j')$, then we say that user instances I_{ij} and $I_{i'j'}$ are *partners*. Note that this *partner relation* is symmetric, and that every user instance has at most one partner.

We shall make a restriction on how the adversary computes connection assignments—see §3.1.8 below.

We will often abuse terminology, and say things like "we *create* I_{ij}," or "we *connect* I_{ij} to $I_{i'j'}$," or "we *compromise* I_{ij}," to mean that user instance I_{ij} is prescribed the indicated connection assignment, i.e., create, $(\mathsf{connect}, i', j')$, or $\mathsf{compromise}$.

3.1.5 Application

Of course, the point of establishing a session key is then to run a higher-level application protocol using the session key. Any use of a session key will potentially leak information about the key to the adversary, which may affect his behavior. We do not want to restrict in any way the types of application protocols. Therefore, we let the adversary obtain any partial information about the session keys that he wishes. In addition to the session keys $\{K_{ij}\}$, we also suppose there is a *random* bit string R of some agreed upon length chosen at the beginning of the game (and not revealed to the adversary). We call R the *random input*.

More specifically, the *application* operation takes the form

$$(\mathsf{application}, f),$$

where f is a function—specified as a straight-line program or circuit (in some canonical notation)—on the random input and the set of session keys that have been defined so far. Upon executing this operation, the ring master gives the adversary $f(R, \{K_{ij}\})$.

If we want to, we can allow *application* operations to have side effects, i.e., to write to variables that may then be read by subsequent *application* operations. This would not have any effect (modulo polynomial-time computation), but would yield a model in which one could express higher level protocols more naturally and efficiently.

Remark 4 *As an example, in an application protocol using the session key, a user instance may encrypt a message using a symmetric key encryption function. The key to the encryption function might be derived from that user instance's session key. The message itself being encrypted may come from some distribution, so a sub-sequence of bits in the random input can be used to generate the message. Also, the encryption algorithm itself may use further random bits that also come from the random input. The bit string actually output by this user instance can be easily expressed by an appropriate function f of R and $\{K_{ij}\}$. This example is discussed at greater length in §4.*

Table 1: Operations and their records in the ideal world transcript

initialize user	$(\textsf{initialize user}, i, ID_i)$
initialize user instance	$(\textsf{initialize user instance}, i, j, role_{ij}, PID_{ij})$
abort session	$(\textsf{abort session}, i, j)$
start session	$(\textsf{start session}, i, j)$
application	$(\textsf{application}, f, f(R, \{K_{ij}\}))$
implementation	$(\textsf{implementation}, \mathit{comment})$

Remark 5 *One may conceptually partition R into segments so that individual user instances have a source of independent random bits. However, having one global bit string R allows us to model situations where users may share secret information (e.g., passwords) through some mechanism other than network communication.*

3.1.6 Implementation

An *implementation* operation is a "no op" or "comment card" that otherwise has no effect on the game, except that the adversary simply makes a *comment*, which is an arbitrary bit string. This may seem strange, but is an essential technical point in formulating security, which will hopefully become clearer later.

The form of this operation is

$$(\textsf{implementation}, \mathit{comment}).$$

3.1.7 Transcripts

We describe how a *transcript* is generated.

As the adversary executes operations in the ideal system, a transcript completely logging his activities is constructed. This transcript consists of a sequence of records.

Each operation adds a record to the transcript, as described in Table 1. Note that no connection assignment information is logged in a *start session* operation (but see §3.1.8).

For an adversary A^* we let $\mathit{IdealWorld}(A^*)$ denote the transcript.

Remark 6 *$\mathit{IdealWorld}(A^*)$ is of course a random variable, determined by the random bits of the adversary, the random input, and the values of the session keys. More precisely, $\mathit{IdealWorld}(A^*)$ is actually a vector, or "ensemble" of distributions, indexed by a "security parameter."*

3.1.8 Transcripts and connection assignments

Having defined the transcript, we now return to one technical point left unexplained in §3.1.4 concerning the calculation of connection assignments. We shall require that the connection assignment made in a *start session* operation be efficiently computable as a function of the transcript up to, and including, the relevant *start session* operation. The motivation for this will be discussed in §3.4, point (8).

3.2 The real system

We now describe a formal model for "real world" session key protocols.

As in the ideal system, we have users U_i and user instances I_{ij}. Also as in the ideal system there is an adversary. In addition, there is a special player T, representing a *trusted third party*. The third party T might be *on line*, as in the private-key setting, or *off line*, as in the public-key setting. We shall assume that T is initialized with a public key/private key pair PK_T/SK_T, although in the on-line TTP setting, this may be trivial.

Unlike in the ideal system, users and user instances are not just place holders.

When a user U_i is initialized with identity ID_i, a protocol-specific, probabilistic initialization routine registers user U_i's identity with T, and initializes user U_i's internal state, as follows. First, the initialization routine computes a *registration request*. Second, the pair $(ID_i, \textit{registration request})$ is sent to T to register the identity ID_i. Upon receiving this request, using a protocol-specific routine, T updates its internal state, and computes a *registration receipt*. Finally, U_i's initialization routine is given this *registration receipt*, and it then computes and stores its long-term state information in the variable LTS_i.

Note that for simplicity, we have opted for a simple, two-pass registration protocol between a user and T. While protocols allowing more interaction would be possible, we shall not need them.

A user instance I_{ij} is a probabilistic state machine. It implicitly has access to PK_T, ID_i, and LTS_i, and upon initialization, it is also assigned a value $\textit{role}_{ij} \in \{0, 1\}$ and a partner identity PID_{ij}. After starting in some initial state, its state may be updated by delivering a message, in response to which the user instance updates its state, generates a response message, and reports its status, which is one of $\mathsf{continue}$, accept, or reject. The meaning of the status value is as follows:

$\mathsf{continue}$: user instance prepared to receive another message.

accept: user instance is finished and has generated a session key; we denote by K_{ij} the session key generated by user instance I_{ij}.

reject: user instance is finished, but refuses to generate a session key.

When I_{ij} processes a message, we allow it to update LTS_i. None of the protocols we examine in this paper explicitly require this facility. However, some digital signature schemes require such a facility. One could also use such a facility to implement a pseudo-random bit generator to supply user instances with pseudo-random bits, instead of making user instances generate their own random bits.

Just as in the ideal system, the adversary plays a game against a ring master, and it is the adversary that drives the game forward by issuing a sequence of operations. In the very first step in this game, the trusted third party T generates a public key/private key pair. The public key is made available to the adversary.

Now we explain precisely the operations that can be performed by the adversary: *initialize user*, *register*, *initialize user instance*, *deliver message*, and *application*.

3.2.1 Initialize user

This operation has the form

$$(\mathsf{initialize\ user}, i, ID_i).$$

The adversary assigns an identity ID_i to user U_i, where user U_i was not previously initialized and ID_i has not been assigned to other users, nor has been used in a *register* operation (see below).

User U_i registers its identity with T and initializes its long-term internal state LTS_i, as described above. The adversary is not given any information.

3.2.2 Register

This operation has the form

$$(\mathsf{register}, ID, \textit{registration request}).$$

The adversary runs T's registration protocol directly with the given identity ID and the given *registration request*, and obtains the resulting *registration receipt*.

There is only one rule restricting the legality of this operation: the set of identities used in *initialize user* operations and the set of identities used in *register* operations must be disjoint. How this rule is enforced lies outside the model (but see §6).

Remark 7 *This operation allows the adversary to operate under various aliases. Alternatively, one can think of these as being the identities of statically corrupted users.*

3.2.3 Initialize user instance

This operation takes the form

$$(\mathsf{initialize\ user\ instance}, i, j, role_{ij}, PID_{ij}).$$

In this operation, the adversary chooses a user instance I_{ij} that has not been previously initialized, and also specifies $role_{ij} \in \{0, 1\}$, and an identity PID_{ij}. User U_i must have been previously initialized. The adversary is not given any information. After execution of this operation, we say that I_{ij} is *active*.

3.2.4 Deliver message

This operation takes the form

$$(\mathsf{deliver\ message}, i, j, InMsg).$$

In this operation, the adversary delivers a message $InMsg$ to an *active* user instance I_{ij}. As described above, the user instance updates its state, outputs a response message $OutMsg$, and reports its *status*. Also, as mentioned above, I_{ij} might also update LTS_i. The response message and status information are given to the adversary. If the *status* is not $\mathsf{continue}$, then user instance I_{ij} is no longer *active*.

In the off-line TTP setting, messages are sent only between user instances, but in the on-line TTP setting, messages sometimes need to be sent between user instances and T. In the latter case, we assume that messages are appropriately tagged to indicate if the sender/receiver is a user instance or T. Also, we assume that users interact with T in a strictly client/server fashion. The details of how this tagging is done are not important. Additionally, we need to add an operation

$$(\mathsf{deliver\ message\ to\ TTP}, InMsg)$$

that delivers a message to T; upon receipt of this message, T updates its internal state, and returns a response message $OutMsg$ to the adversary.

Note that in an actual implementation, a user instance might "time out" after some time if it is waiting for a message. Although there is no notion of absolute time in our model, the adversary can deliver a special "time out" message to a user instance to achieve the same effect.

Table 2: Operations and their records in the real world transcript

initialize user	$(\textsf{initialize user}, i, \mathit{ID}_i)$
register	$(\textsf{implementation}, \textsf{register}, \mathit{registration\ request}, \mathit{ID}, \mathit{registration\ receipt})$
initialize user instance	$(\textsf{initialize user instance}, i, j, \mathit{role}_{ij}, \mathit{PID}_{ij})$
deliver message	$(\textsf{implementation}, \textsf{deliver message}, i, j, \mathit{InMsg}, \mathit{OutMsg}, \mathit{status})$, and $(\textsf{start session}, i, j)$ if $\mathit{status} = \textsf{accept}$, and $(\textsf{abort session}, i, j)$ if $\mathit{status} = \textsf{reject}$
deliver message to TTP	$(\textsf{implementation}, \textsf{deliver message to TTP}, \mathit{InMsg}, \mathit{OutMsg})$
application	$(\textsf{application}, f, f(R, \{K_{ij}\}))$

3.2.5 Application

This operation takes the form

$$(\textsf{application}, f).$$

This is exactly the same as the *application* operation in the ideal system, except that the function now computed is a function of the actual session keys $\{K_{ij}\}$ generated by user instances, as well as a random input R. Note that R is independent of any random bits used by users or user instances during initialization and the during the execution of session key protocols.

3.2.6 Transcripts

We now describe the transcript generated by the adversary's game. This is a sequence of records that describes all the activities of the adversary and all the information available to it.

The first record in the transcript is

$$(\textsf{implementation}, \textsf{initalize system}, PK_T).$$

Each operation adds one or two records to the transcript, as detailed in Table 2.

For an adversary A, we let $\mathit{RealWorld}(A)$ denote the transcript.

3.3 Definition of security

We are finally ready to formulate the definition of security of a session key exchange protocol. There are three basic requirements.

Termination. Any (real world) user instance must terminate after a polynomially bounded number of messages are delivered to it (the bound must be independent of the adversary). In fact, we shall only consider here protocols that terminate after a constant number of rounds.

Liveness. For every efficient real world adversary A, whenever the adversary faithfully delivers messages between two compatible user instances (and T, in the on-line TTP setting), both user instances accept and share the same session key.

Simulatability. For every efficient real world adversary A, there exists an efficient ideal world adversary A^* such that $\mathit{RealWorld}(A)$ and $\mathit{IdealWorld}(A^*)$ are *computationally indistinguishable*.

3.4 Discussion

1. The liveness requirement rules out, for example, "do nothing" protocols that would trivially satisfy the simulatability requirement.

2. The simulatability requirement captures the intuition that any real world adversary does no more "damage" than an ideal world adversary, and by definition, an ideal world adversary is essentially benign.

3. We do not explicitly place any internal random bits used by the real-world adversary in the transcript. However, a real-world adversary can always force any information it wants into the transcript using an *application* operation. To see how this can be done, note that the function f specified in the *application* operation could very well be a "constant" function of the adversary's choice, and this "constant" can be an arbitrary bit string computed by the adversary. Admittedly, this is perhaps a bit artificial; alternatively, one could simply add a special operation that allows the real-world adversary to place "comments" in the transcript.

4. Also using *application* operations, the real-world adversary can arrange that the session keys K_{ij} along with the random input R are "dumped" into the transcript at the very end of the game. This will allow a statistical test attempting to distinguish the real-world and ideal-world transcripts access to otherwise hidden variables.

5. It may be useful to illustrate the definition of security with a simple example. Suppose that a real world adversary A has the power to simply output a session key just after it has been established (but not used). We can arrange that A forces its "guess" of the session key into the transcript, as described in point (3). We can also arrange that A forces the actual value of the session key into the transcript at the end of the game, as described in (4). In the real world, the guessed value of the key and the actual value would be equal, at least with non-negligible probability, assuming A could really break the scheme as described. In the ideal world, these two values would be equal with only negligible probability (at least for sufficiently long session keys). But this would immediately give us a statistical test to distinguish real-world from ideal-world transcripts. So either there is no such A, or the session key protocol is insecure.

6. Our definition of security implies much more than just the inability of an adversary to guess a session key. It has a sort of "built in" composition theorem, since the *application* operation allows session keys to be used in an arbitrary way by higher-level protocols. We believe that our definition is general enough so that *any* high-level protocol can be *directly* and *naturally* represented using appropriate *application* operations. The *simulatability* requirement implies that any event that happens in the real world must happen in the ideal world with essentially the same probability, as long as this event can be expressed as a function on the transcript (as augmented above in points (3) and (4)). It is in this sense that a high-level protocol designer can "pretend" he is working in the ideal world, rather than the real world.

   There is, however, one direction in which our definition could be extended. As it is, the function computed by an *application* operation depends on the session keys and on some hidden random variables. One might also want these functions to depend on some hidden values that cannot effectively be modeled as random variables. To deal with this, one could introduce an auxiliary input S to be used in both the ideal world and the real world, and the *simulatability* requirement would have to hold for all values of S. To prove the security of a protocol in this setting, one would have to make *non-uniform* intractability assumptions. Although one can extend the definition in this way, it is not clear to us that this is a particularly useful extension.

7. We emphasize that the *application* operation is mainly intended to model the *normal* use of a session key in a higher-level protocol (e.g., to implement a secure session, see §4), although it can also be used to model unintended information leakage as well (see §14.1).

8. There is no connection assignment information explicitly available in either the real-world or ideal-world transcripts. This is clearly inevitable, as the real-world transcript can not be expected to contain this information, and of course the ideal-world transcript is supposed to look just like the real-world transcript. However, this information is implicitly available to any statistical test attempting to distinguish the transcripts, since the ideal-world adversary implicitly defines an efficiently computable function from transcripts to connection assignments. This was the main point of the restriction in §3.1.8 on how connection assignments are computed. Indeed, it only seems fair that this information is available to the statistical test. Our definition of security essentially implies that any real-world adversary could be replaced by an equivalent adversary that explicitly announces its intended connection assignments.

9. Although we give the ideal-world adversary complete freedom in determining connection assignments, this freedom is quite superficial. Indeed, since all the session keys may be eventually dumped into the transcript, the adversary really has no freedom at all, if the protocol is actually secure. That is, for a secure protocol, connection assignments are unique. We could have restricted the way in which A^* computes connection assignments, but this would only complicate the definitions without any clear advantage.

10. Admittedly, the whole business of connection assignments is rather messy, even though we have tried to simplify it as much as possible. An alternative approach to handling connection assignments would be to define them via a "session ID." This is the approach taken by [BCK98]. In the real world, one would require that a user instance compute and output a session ID when it accepted. In the ideal world, the session ID would be specified by the adversary in a *start session* operation. One would then formulate rules to calculate connection assignments from session IDs. While this approach may have some appeal, it only seems to yield a more complicated definition of security, with additional, arguably unnecessary syntactic constraints. Moreover, it would require that protocols actually compute these session IDs. For many protocols, this would entail only a trivial modification, but for others, such as the one analyzed in [BR95], it would require a decidedly non-trivial modification.

11. The value $role_{ij} \in \{0,1\}$ assigned to a user instance may be a bit confusing at first. Session key exchange protocols are typically asymmetric in nature; moreover, higher-level protocols making use of a session key typically can benefit from this asymmetry as well. See §4 for an example of this.

12. Our definition of security does not imply any notion of *explicit key confirmation* (a.k.a. "explicit key authentication," see, e.g., [MvOV97, p. 492]). The notion of explicit key confirmation is usually rather vaguely defined, but in our terminology it could be phrased as the requirement that a user instance is guaranteed that a compatible user instance has accepted the same key.

    Some researchers distinguish between *unilateral* and *mutual* explicit key confirmation (providing the above guarantee to one or both user instances, respectively). We point out, however, that mutual explicit key confirmation is impossible to achieve. This is essentially a consensus problem, and is in general unsolvable in the presence of faulty communication links: the user instance that sends the last message can never "know" if this message will be delivered, and therefore can never "know" whether it really has established a connection with anyone. Although there may be a compatible instance of the key establishment protocol that holds the session key, that protocol instance may not terminate successfully, and so may not pass the key up to a higher-level protocol, which for all practical purposes is equivalent to not holding the session key in the first place. This is a rather subtle point that some researchers in the field have failed to appreciate, and in fact, some researchers have claimed that certain protocols actually provide mutual explicit key confirmation, e.g., this is claimed in [MvOV97, p. 516] for STS. At best, the notion of mutual explicit key confirmation is meaningless (this meaninglessness is simply obscured by the lack of precise definitions); at worst, it gives high-level protocol designers an unjustified sense of security.

As it happens, all of the protocols we examine in this paper, except protocol **A-EKE-2** in §9.3, do indeed provide *unilateral* explicit key confirmation. One could modify our definition of security so as to guarantee unilateral explicit key confirmation, by requiring that a user instance's *role* be correlated with its *connection assignment*. That is, one role would allow only the connection assignment *create*, and the other only *connect*. This is done, for example, in [BCK98]. One should note, however, that such a requirement would rule out otherwise perfectly good protocols, such as the one in [BR95]; moreover, it is not at all clear that higher-level protocols can truly profit from such a requirement.

13. In our definition of security, we make no attempt to isolate or formulate the notion of "entity authentication." Roughly speaking, this notion tries to capture the goal that two parties can engage in a protocol so that at the end of the protocol, they are sure that they were really talking to each other. This is sometimes pursued as an end in itself [DvOW92, BR93a, BM97], but it is not clear if this is very useful. Such a protocol only establishes that two entities were talking to each other in the past, but implies nothing about messages sent in the future—it is simply a secure, mutual "ping."

14. One technical point in our definition of security is: what happens when a user instance I_{ij} runs the session key protocol when its partner identity PID_{ij} has neither been assigned to the user nor has it been registered by the adversary? In typical protocols in the off-line TTP setting, I_{ij} will certainly not accept any session key, since it will expect to see a certificate containing the identity PID_{ij}, but will not. However, it is possible to concoct protocols in the on-line TTP for which I_{ij} will accept a session key, and this session key may not be known to the adversary, but may become known to the adversary at a later time if it registers the identity PID_{ij}; nevertheless, the protocol would satisfy our definition of security, as our definition makes no security guarantees when PID_{ij} is not assigned to a user (it may be given a connection assignment of compromise). It seems to be a debatable point as to whether this is acceptable. One could strengthen our definition by requiring that a user instance *rejects* if PID_{ij} has neither been assigned nor registered by the adversary. This would be a simple modification of the definition, and at any rate, would not affect the security of any of the protocols we discuss in this paper.

15. Our formal real-world model implicitly forbids higher level protocols from making use of the long-term private keys used in the session key protocol. The only "secret" information used in higher level protocols is contained in the random input R, which is independent of these

private keys. This significantly simplifies our model, and is anyway good security practice. If one did allow the same private keys to be used in the session key protocol and the higher level protocol, one would have to be very careful to prevent protocol interference.

4 The Principle Application: Secure Sessions

Perhaps the main reason for exchanging session keys is to establish secure sessions, so it is perhaps worthwhile to discuss this point in some detail.

Having established a shared key K, two user instances can then proceed as follows. Applying a pseudo-random bit generator to K, they can derive sub-keys $K^{(0)}$ and $K^{(1)}$. The two user instances can identify themselves according to their roles, so for the purposes of this session, the user instance with role 0 can be "player 0," and the user instance with role 1 can be "player 1." The key $K^{(0)}$ can then be used to implement a secure—i.e., private and authenticated—uni-directional channel from player 0 to player 1. This can be done using standard symmetric key encryption (semantically secure against chosen message attack) and a message authentication code. We assume that messages are transmitted as fixed-size blocks (the size may depend on a security parameter), and that each block is individually encrypted and authenticated, so that not only the integrity of the data in each block is preserved, but also the relative ordering of the blocks. Similarly, the key $K^{(1)}$ can be used to implement a secure uni-directional channel from player 1 to player 0. The two players can interleave the sending of message blocks on the two channels in an arbitrary way.

All of the functions for encryption and generating message authentication codes can be expressed in our formal model as appropriate *application* operations that are performed by the adversary. This would be the only access to session keys—and the bits in the random input used to implement the secure channel—that we would allow our adversary, but in general, we might allow the adversary access to other bits in the random input.

Our definition of secure key exchange then allows one to establish all of the properties one would expect if the shared key K were truly random.

In fact, one could carry our formal modeling one step further by giving a simulation-based definition for a secure session protocol. Although we do not pursue this in detail here, we sketch an approach for which it should be easy to fill in the details.

As usual, one would define an "ideal world" and the "real world." In the ideal world, the adversary would initialize users and user instances, as well as start sessions specifying connection assignments just as in the key exchange setting, except that in the secure session setting, there would be no mention of session keys—at this level of abstraction, that is an implementation detail, and not a part of the specification.

We say that the session for a user instance is *compromised* if the user instance has a connection assignment of compromise. We begin by describing the ideal workings of a channel associated with an uncompromised session.

Once a user instance has established a session, it has access to one input channel and one output channel. The basic operations are to write a message block to the output channel and to read a message block from the input channel. As usual, all activities are directed by the adversary.

Whenever the adversary requests a user instance to write a message block, this defines a *message block variable*. Associated with the output channel is a sequence of message block variables $X_1, X_2, \ldots$. The value of a message block variable X_r is computed as a function of the random input, and any previously defined message block variables in the system. This function is specified by the adversary. This is very similar to the *application* operation in the context of session key protocols, except that in this setting, all that happens is that variable X_r is defined, and the adversary is given no information about its value.

Whenever the adversary requests a user instance to read a message block, this also defines a message block variable. Associated with the input channel is a sequence of message block variables $X'_1, X'_2, \ldots$. For the rth read operation to be legal (for $r = 1, 2, \ldots$), the user instance must have a partner, and that partner must have performed at least r write operations. The value of the variable X'_r associated with this user instance's input channel is assigned the value of the variable X_r associated with its partner's output channel. The adversary can also explicitly close a user instance's input or output channel, after which it does not read or write any more messages.

The privacy of these channels is guaranteed by the fact that when a user instance writes to its output channel, the ideal-world adversary obtains no information beyond which it already knows; namely, that the message block is the value of the function it specified. The authenticity of these channels is guaranteed by the fact that the values of the message blocks received are equal to the values of the message blocks actually sent.

That deals with the case when a session is uncompromised. If it is compromised, all of the above guarantees are eliminated—specifically, when sending a message, the ideal-world adversary is simply given the message block directly, and when receiving a message, the adversary explicitly delivers a message block of its choice.

In addition to the above, the adversary may make arbitrary requests to obtain the values of specific functions on the random input and on defined message block variables, just like the *application* operation in the context of session key protocols. This allows us to model yet higher-level protocols that run on top of the secure session.

The main reason for using fixed-size blocks is that in general, we cannot hope to prevent the adversary from learning something about the lengths of transmitted messages, so we just fix these lengths in advance.

It should be straightforward to fill in the details of the ideal world specification, as well as to describe an appropriate formal model of the real world, and to show how to implement and prove the security of a secure session protocol on top of a secure session key protocol, using completely standard symmetric-key cryptographic techniques. Note that in the real world, user instances would also have message block variables associated with (virtual) input and output channels. Also, in formulating the definition of security, one would also have to formulate appropriate notions of *termination* and *liveness*, which should be straightforward. The *liveness* requirement would simply say that in the real world, to the extent an adversary faithfully delivers messages between two compatible user instances, these user instances effectively behave as partners with connected input/output channels, and all message blocks are effectively delivered without modification.

One important point to note, however, is that using standard implementation techniques, we will not be able to maintain simulatability if we allow the adversary to arbitrarily expose session keys. The problem is not that we cannot model such attacks in our formalization—in fact, we can quite easily. The problem is that a standard symmetric-key encryption of a message is a commitment to that message—if the key is later exposed, the simulator cannot make it look like something else was encrypted (but see §14.6).

5 Cryptographic Primitives

In this section, we discuss the cryptographic primitives that we will be using throughout the rest of this paper.

5.1 Digital signatures

We will make use of digital signature schemes, and the notion of security we will use is that of security against existential forgery against adaptive chosen message attack, as defined in [GMR88]. This is the strongest, and most useful notion of security.

Briefly, security in this sense means that it is infeasible for an adversary to win the following game. A public key/private key for the scheme is generated, and the adversary is given the public key. The adversary then makes a sequence of signing requests. The messages for which the adversary requests signatures can be adaptively chosen, i.e., they may depend on previous signatures. The adversary wins the game if he can forge a signature, i.e., can output a message other than one for which he requested a signature, along with a valid signature on that message.

Secure and fairly practical signature schemes can be constructed based on various intractability assumptions [DN94, CD96, GHR99, CS99]. Even more practical schemes can be constructed based on heuristic arguments (i.e., the "random oracle" model) [BR96, PS96].

5.2 Public key encryption

The notion of semantic security for a public-key encryption scheme was formalized by [GM84].

Briefly, security in this sense means that it is infeasible for an adversary to gain a non-negligible advantage in the following game. A public key/private key pair for the scheme is generated, and the adversary is given the public key. Then the adversary generates two equal length messages m_0, m_1, and gives these to an *encryption oracle*. The encryption oracle chooses a bit $b \in \{0,1\}$ at random, encrypts m_b, and gives the adversary the corresponding *target* ciphertext ψ'. Finally, the adversary outputs his guess at b. The adversary's advantage is defined to be the distance from $1/2$ of the probability that his guess is correct.

The formal definition of semantic security captures the intuitive notion that no information about an encrypted message is leaked to a *passive* adversary that only eavesdrops. In protocol design and analysis, a much more robust definition is often required that captures the intuitive notion of security against an *active* attack, in which the adversary not only can eavesdrop, but can inject his own messages into the network. The type of security one needs in this setting is *non-malleability*, also called *security against chosen ciphertext attack*, a notion that was formalized in the sequence of papers [NY90, RS91, DDN91].

The definition of non-malleability is the same as for semantic security, but with the following essential difference. The adversary is given access to a *decryption oracle* throughout the entire game; the adversary may request the decryption of ciphertexts ψ of his choosing, subject only to the restriction that after the target ciphertext ψ' has been generated, the adversary may not request the decryption of ψ' itself.

Another intuitive way to understand non-malleability (and the motivation for its name) is that a non-malleable encryption scheme essentially provides a *secure envelope*, that is, an envelope whose contents can neither be seen nor modified by an adversary.

Fairly practical non-malleable encryption schemes can be constructed based on the Decisional Diffie-Hellman assumption (see below) [CS98]. Even more practical schemes can be constructed based on heuristic arguments (i.e., the "random oracle" model) [BR93b, BR94, FO99].

5.3 The Decisional Diffie-Hellman assumption

Let G be a group of large prime order q and let $g \in G$ be a generator. The Computational Diffie-Hellman (CDH) assumption, introduced by [DH76], is the assumption that computing g^{xy} from

g^x and g^y is hard. It is a widely held belief that the security protocols such as STS is implied by the CDH assumption. This is simply false—under *any* reasonable definition of security—except in a heuristic sense that we discuss below in §5.3.3. What is almost always needed, but often not explicitly stated, is the Decisional Diffie-Hellman (DDH) assumption.

For $g_1, g_2, u_1, u_2 \in G$, define $DHP(g_1, g_2, u_1, u_2)$ to be 1 if there exists $x \in \mathbf{Z}_q$ such that $u_1 = g_1^x$ and $u_2 = g_2^x$, and 0 otherwise. A "good" algorithm for *DHP* is an efficient, probabilistic algorithm that computes *DHP* correctly with negligible error probability *on all inputs.* The DDH assumption is the assumption that there is no good algorithm for *DHP*.

This formulation is equivalent to the more usual one where

$$g_1 = g, g_2 = g^x, u_1 = g^y, u_2 = g^{xy}.$$

5.3.1 DDH random self-reduction

There are a few useful random self-reductions that allow us to transform arbitrary inputs to *DHP* into random inputs on which *DHP* evaluates to the same value.

Let g_1, g_2, u_1, u_2 be given such that $g_1 \neq 1$ and $g_2 \neq 1$. We can randomize u_1 and u_2 as follows:

$$\tilde{u}_1 = u_1^a g_1^b, \ \tilde{u}_2 = u_2^a g_2^b,$$

where $a, b \in \mathbf{Z}_q$ are chosen at random. Suppose that $u_1 = g^x$ and $u_2 = g_2^y$. If $x = y$, then $(\tilde{u}_1, \tilde{u}_2)$ is a random pair of group elements, subject to $\log_{g_1}(\tilde{u}_1) = \log_{g_2}(\tilde{u}_2)$. If $x \neq y$, then $(\tilde{u}_1, \tilde{u}_2)$ is a pair of random, independent group elements.

Next, we can randomize g_2 as follows:

$$\tilde{g}_2 = g_2^c, \ \tilde{u}_1 = u_1^a g_1^b, \ \tilde{u}_2 = u_2^{ac} g_2^{bc},$$

where $c \in \mathbf{Z}_q$ is chosen at random.

Additionally, we can randomize g_1 as follows:

$$\tilde{g}_1 = g_1^d, \ \tilde{g}_2 = g_2^c, \ \tilde{u}_1 = u_1^{ad} g_1^{bd}, \ \tilde{u}_2 = u_2^{ac} g_2^{bc},$$

where $d \in \mathbf{Z}_q$ is chosen at random.

With this transformation, we see that we can transform an arbitrary input to *DHP* to an equivalent, random input. From this, it follows that the two distributions

$$(g_1, g_2, g_1^x, g_2^y), \text{ random } g_1, g_2 \in G; \ x, y \in \mathbf{Z}_q,$$

and

$$(g_1, g_2, g_1^x, g_2^x), \text{ random } g_1, g_2 \in G; \ x \in \mathbf{Z}_q$$

are computationally indistinguishable under the DDH assumption. This random self-reducibility property was first observed by Stadler [Sta96] (and also independently in [NR97]).

5.3.2 Applying the DDH assumption

In the sequel, we will need to use a superficially stronger version of the DDH assumption, which in fact is implied by the DDH assumption.

First, it follows from the DDH assumption, using a hybrid argument (see [NR97]), that the two distributions

$$(g, (g^{x_i} : 1 \leq i \leq n), (g^{y_j} : 1 \leq j \leq m), (g^{x_i y_j} : 1 \leq i \leq n, 1 \leq j \leq m)$$

and

$$(g, (g^{x_i} : 1 \leq i \leq n), (g^{y_j} : 1 \leq j \leq m), (g^{z_{ij}} : 1 \leq i \leq n, 1 \leq j \leq m))$$

are computationally indistinguishable. Here, the base g is random, as are the exponents.

By a slightly more involved hybrid argument, it follows that an adversary's advantage in the following interactive version of the above distinguishing problem is negligible. In this game, a b is chosen at random, hidden from the view of the adversary. Next, the adversary is given

$$(g, (g^{x_i} : 1 \leq i \leq n), (g^{y_j} : 1 \leq j \leq m)).$$

For all i, j, we define $h_{ij} = g^{x_i y_j}$ if $b = 0$, and $h_{ij} = g^{z_{ij}}$ if $b = 1$. Now the adversary adaptively makes a sequence of requests. For any i he can ask to see x_i, for any j, he can ask to see y_j, and for any i, j he can ask to see h_{ij}. These requests are subject to the obvious restriction that if he asks for h_{ij}, he cannot also ask, or have asked, for x_i or y_j. At the end of the game, the adversary outputs his guess at b. The adversary's advantage is defined to be the distance from $1/2$ of the probability that his guess is correct.

We will use the above observations in the analysis of Diffie-Hellman based key exchange protocols. Additionally, we will also use the Entropy Smoothing Theorem (a.k.a., the Leftover Hash Lemma) to transform random group elements into random bit strings using a pair-wise independent hash function. See [Lub96, Chapter 8] for an exposition on the Entropy Smoothing Theorem. We will use this theorem as follows. Having computed a Diffie-Hellman key g^{xy}, we will derive a session key as $H_k(g^{xy})$, where H is a family of pair-wise independent hash functions, and k is a random index into this family of functions. Under an appropriate choice of parameters, the DDH assumption and the Entropy Smoothing Theorem imply that the distributions $(g, g^x, g^y, k, H_k(g^{xy}))$ and (g, g^x, g^y, k, K)—where K is a random bit string whose length equals the output length of H—are computationally indistinguishable.

5.3.3 Using random oracles

Instead of the DDH assumption, one can use the CDH assumption in combination with the *random oracle* model of security analysis (see [BR93b]). This is a heuristic model of analysis in which a cryptographic hash function F is treated *as if* it were a black box that contained a random function. This model has been used to analyze numerous cryptographic systems (see, e.g., [BR94] and [PS96]). In all of the Diffie-Hellman based key exchange protocols we analyze, if we compute the session key as $K = F(g^{xy})$, then the protocols can be proven secure in the random oracle model under the CDH assumption.

We can also combine the two approaches, obtaining the "best of both worlds." If we compute the session key as $K = H_k(g^{xy}) \oplus F(g^{xy})$, then we get a proof of security under the DDH assumption (without resorting to random oracles), and under the CDH assumption with random oracles.

5.3.4 Discussion

To make all of the above definitions and arguments precise, one should view the group G not as fixed, but as being generated by some probabilistic algorithm taking as input a sufficiently large security parameter. The above hybrid arguments can be readily adapted to the case where we have a heterogeneous system of groups G, each of which is generated in this way.

The DDH assumption appears to have first surfaced in the cryptographic literature in a paper by S. Brands [Bra93]. See [Bon98, CS98, NR97, Sta96] for further applications of and discussions about the DDH assumption. A potentially stronger version of the DDH assumption—which we shall not need in this paper—allows the adversary to choose one of the two bases g_1 or g_2 in the above distinguishability problem. Interestingly, it appears that allowing the adversary to choose one of the bases may give him more power than he would have if both bases were random. This is in contrast to the CDH and Discrete Logarithm assumptions, where it does not matter if the adversary chooses the base. It remains to be seen whether such a stronger version of the DDH assumption has useful cryptographic applications. We mention this here only because it seems that previous works involving the DDH did not make this distinction.

6 The Certificate Authority

In this section, we describe precisely the role of the trusted third party T as a certificate authority in this and all the other protocols in this paper.

We assume that when a user U_i is initialized (see §3.2), he generates a public key/private key pair PK_i/SK_i, and the *registration request* is PK_i. The trusted third party T, acting as a certificate authority, generates a certificate $cert_i$, which consists of a signature on (ID_i, PK_i) under PK_T. The *registration receipt* is simply $cert_i$. The long-term state information LTS_i of user U_i is $(SK_i, PK_i, cert_i)$.

Recall that the rules for registration (§3.2.1-§3.2.2) prevent two honest users from registering the same name, and prevent the adversary from registering an honest user's name. Of course, the enforcement of these rules lies outside our formal model. For example, in real life, the certificate authority might be able to reasonably enforce these rules by requiring the use of sufficiently descriptive names and by demanding adequate "proofs" of identity (passport, driver's license, etc.).

Note, however, that we will not require anything more of the certificate authority. In particular, we shall not require that a user proves that he "knows" the secret key corresponding to a public key when he gets a certificate—a practice that is sometimes advocated. So, for example, there is nothing stopping an adversary from obtaining a certificate that binds the name of a "corrupted user" (i.e., an alias under which the adversary is operating) to the public key of an honest user.

There are three reasons for not doing this. First, both from a trust and an efficiency point of view, it seems best to require as little of T as possible. Second, it is easy to design quite efficient key exchange protocols that are secure under our minimalistic trust assumption. Third, it is not clear how one would really exploit a "proof of knowledge" to get rigorous security proofs—from a technical point of view, "proofs of knowledge" are quite tricky to work with, since they often involve "rewinding," which can cause real problems when trying to build a simulator.

Note that as we have set things up, user certificates are not available in any "public directory." Instead, we shall require that user certificates are transmitted as a part of the session key protocol itself. This closely models what happens in practice. By implication, the adversary also does not get direct access to user certificates: the adversary must obtain user certificates by interacting with users, just like an honest user must. This is not at all a serious restriction, and we could easily add a *certificate request* command in the real system without changing any of the theorems we later prove.

7 A Diffie-Hellman Based Protocol

In this section we describe and analyze a protocol based on the classical Diffie-Hellman protocol key exchange [DH76]. We call our proposed protocol **DHKE**.

7.1 Protocol DHKE

Each user generates a public key/private key pair as follows. First, he chooses a public key/private key pair for a digital signature scheme. Second, he constructs a group G of prime order q, and selects a random generator g for this group. The user's public key consists of the public key for the signature scheme, and a description of G and g. The user's private key consists of the private key of the signature scheme. We denote by $sig_i(msg)$ the output of user U_i's signature algorithm on *msg*. Note that in this paper, signatures do not include the message being signed. We remind the reader that $cert_i$ denotes the certificate that binds user U_i's public key with his identity, as described in §6.

We describe the protocol in terms of two users U_i and $U_{i'}$. User U_i initiates the protocol, and in the description of the protocol, G, g, and q refer to the group information recorded in the public key of user U_i. We also assume a family of pair-wise independent hash functions H_k, indexed by a randomly chosen bit string k of some specified length.

$U_i \to U_{i'}:\ g^x, sig_i(g^x, ID_{i'}), cert_i,$
where $x \in \mathbf{Z}_q$ is chosen at random.
$U_{i'} \to U_i:\ g^y, k, sig_{i'}(g^x, g^y, k, ID_i), cert_{i'},$
where $y \in \mathbf{Z}_q$ is chosen at random, and k is a random hash function index.

The agreed upon session key is $H_k(g^{xy})$, computed in the usual way. Additionally, each player validates all certificates and signatures in the usual way, rejecting the protocol and refusing to generate a session key if any of these tests fail.

Our description is not entirely precise. Some user instance I_{ij} is running the protocol on behalf of user U_i, and likewise some user instance $I_{i'j'}$ is running the protocol on behalf of user $U_{i'}$. The identity $ID_{i'}$ written in the first flow is actually computed by I_{ij} as PID_{ij}, and by $I_{i'j'}$ as $ID_{i'j'}$. Similar remarks apply to other computations in the protocol. Also, we arbitrarily let the *roles* of the two user instances in this and other protocols in this paper be determined by who goes first.

Remark 8 *The reader may already have a funny feeling about this protocol, as it consists of only two flows, as opposed to the three flows used in STS. Indeed, the first message generated by U_i could be sent to several instances of $U_{i'}$, at most one of which can actually end up sharing a key with U_i. At worst, this will lead to user instances $I_{i'j'}$ that are permanently isolated. However,* any *session key protocol ultimately suffers from this problem: whoever sends the last message in the protocol does not "know" if it was ultimately delivered. As was pointed out in §3.4, point (12), we should not expect a key exchange protocol to solve the consensus problem, which is anyway unsolvable in general. Moreover, even if all messages in the session key protocol are delivered, there is in general no guarantee that any messages in higher-level application protocols will be delivered.*

7.2 Security analysis of DHKE

Theorem 1 *Protocol* **DHKE** *is a secure key exchange protocol, under the DDH assumption, and assuming all the digital signatures schemes employed are secure.*

We now prove this theorem.

We are given a real world adversary A. Our approach will be to transform A into an ideal world adversary A^*, and to simultaneously transform the real world ring master into an ideal world ring master, doing this without changing the transcript in any (computationally) discernible way. Basically, this will simply amount to having A^* run the adversary A just as in the real world, except as follows:

- A^* computes appropriate connection assignments, and the ring master in the ideal world substitutes real-world session keys with idealized session keys;

- whenever A^* chooses to *compromise* a user instance, it supplies a session key as part of the *start session* operation by extracting the key from the real-world user instance;

- for any implementation record that A's actions cause to be placed in the real-world transcript, A^* copies this record into the ideal-world transcript using a corresponding *implementation* operation;

- any *application* operations are evaluated by the ring master using the idealized session keys.

What we end up with, then, is an adversary A^* that is a system of interacting algorithms consisting of A, the real world users, and T. The main thing is to argue that these connection assignments are legal, and that the key substitutions are not detectable.

We make one more notational convention that will also be used in our other proofs of security. We will always write I_{ij} for a user instance that is an *originator*, i.e., sends the first message in the protocol, and $I_{i'j'}$ for a user instance that is a *responder*, i.e., a user instance that sends the second message in the protocol. For any two such user instances, the values $role_{ij}$ and $role_{i'j'}$ are complimentary.

Case 1. Suppose a user instance $I_{i'j'}$ has received the first message in the protocol and has accepted.

Case 1a. If $PID_{i'j'}$ is not assigned to a user, then we *compromise* $I_{i'j'}$ in the ideal world. To do this, we need to specify the session key, which we simply extract from $I_{i'j'}$ in the real world.

Case 1b. Now suppose $PID_{i'j'}$ is assigned to a user U_i. We assert that at this point there is a unique user instance I_{ij} such that $PID_{ij} = ID_{i'}$ and that I_{ij} sent g^x in its outgoing message. This follows easily from the logic of the protocol and the security of the signatures. So we *create* $I_{i'j'}$, and the ring master substitutes the *actual* session key with a *random* session key. We have to argue that this substitution is unnoticable (computationally, that is). But this will follow using the argument in §5.3.2, provided the user instance I_{ij} mentioned above has not been or ever will be *compromised*. But this is so, because $PID_{ij} = ID_{i'}$, and so such a *compromise* connection assignment is not allowed in the rules. The only possible outcomes for I_{ij} are that it never accepts, it connects to $I_{i'j'}$, or it connects to another instance of user $U_{i'}$.

Case 2. Suppose that user instance I_{ij} has just received an incoming message in the protocol and has accepted.

Case 2a. If PID_{ij} is not assigned to a user, then we *compromise* I_{ij}, extracting the needed session key from I_{ij} itself.

Case 2b. Now suppose PID_{ij} is assigned to a user $U_{i'}$. We assert that there is a unique isolated user instance $I_{i'j'}$ such that $PID_{i'j'} = ID_i$ and the values g^x, g^y and k match. This follows easily from the logic of the protocol and the security of the signatures. This allows us to *connect* I_{ij} to $I_{i'j'}$, and the ring master substitutes the actual session key of I_{ij} with the session key of $I_{i'j'}$ previously generated by the ring master. This substitution will be unnoticable because the session keys were the same in the real world.

That completes the proof of the simulatability requirement in the definition of a secure key exchange protocol. It is trivial to see that the termination and liveness requirements are satisfied.

8 An Encryption Based Protocol

8.1 Protocol EKE

Each user generates public key/private key pairs as follows. He chooses a public key/private key pair for a signature scheme, as well as a public key/private key pair for a public key encryption scheme.

As in §7, we let $sig_i(msg)$ denote user U_i's signature on msg. Also, we let $E_i(msg)$ denote an encryption of msg under user U_i's public key.

We describe the protocol in terms of two user's U_i and $U_{i'}$.

$U_i \to U_{i'} :\ r, cert_i,$
where r is a (sufficiently long) random bit string.

$U_{i'} \to U_i :\ \alpha = E_i(K, ID_{i'}), sig_{i'}(\alpha, r, ID_i), cert_{i'},$
where K is a random bit string.

The agreed upon session key is K. As usual, both users check the relevant signatures. Additional, user U_i checks that the decryption of α is of the right form, containing the expected identity.

8.2 Security analysis of EKE

Theorem 2 *Protocol* **EKE** *is a secure key exchange protocol, assuming secure signatures, and assuming the encryption scheme is non-malleable.*

We now prove this theorem. The basic structure of the proof is the same as that of Theorem 1 in §7.

Case 1. Suppose a user instance $I_{i'j'}$ has just received the first message in the protocol, and has accepted.

Case 1a. If $PID_{i'j'}$ is not assigned to a user, then we *compromise* $I_{i'j'}$ in the ideal world, extracting the key K from $I_{i'j'}$.

Case 1b. Otherwise, suppose $PID_{i'j'}$ is assigned to a user U_i. Then we *create* $I_{i'j'}$ in the ideal world, and the ring master substitutes the session key held by $I_{i'j'}$ with a random key. Because we are assuming the encryptions are non-malleable, this substitution will be undetectable, *provided the ciphertext α is never actually decrypted.* We justify this claim below.

Case 2. Now suppose a user instance I_{ij} has just received a message.

Case 2a. Suppose that PID_{ij} is not assigned to a user.

If the ciphertext α received was generated by any user instance $I_{i'j'}$ with $PID_{i'j'} = ID_i$, then we can safely make I_{ij} reject, as the identity embedded in α ($ID_{i'}$) is not what I_{ij} expects (PID_{ij}). This is done without ever decrypting α.

Otherwise, if the ciphertext α was not generated by any such user instance, we let I_{ij} run to completion. If it accepts, we *compromise* I_{ij}, extracting the session key from I_{ij}. This of course makes implicit use of the decryption function of user U_i, but we have taken care not to decrypt anything that was encrypted by a user instance $I_{i'j'}$ with $PID_{i'j'} = ID_i$. As we will see, this is the *only* place in the game where we decrypt anything. This justifies the claim made above in case (1b) that we never decrypt ciphertexts created by user instances with connection assignment create.

Case 2b. If PID_{ij} is assigned to a user $U_{i'}$, we proceed as follows. If the signature verification succeeds, then the ciphertext α must have been created by a unique user instance $I_{i'j'}$ with $PID_{i'j'} =$

ID_i, and the the identity embedded in the ciphertext must be $ID_{i'}$, so there is no need to actually decrypt it. Moreover, it is easy to see that connecting I_{ij} to $I_{i'j'}$ is valid at this moment, because the r values are all unique (at least with overwhelming probability). So we *connect* I_{ij} to $I_{i'j'}$, and the ring master sets the session key of I_{ij} to the key that it generated for $I_{i'j'}$. This substitution is undetectable, since the two session keys were the same in the real world.

It is clear from the arguments already made that the real world and ideal world transcripts are indistinguishable. That proves the simulatability requirement. The termination and liveness requirements are trivial.

Remark 9 *One of the main attractions of protocol* **EKE** *is that it can be implemented so as to minimize the computational efforts of one of the two users—user $U_{i'}$ in this case. First, one could use a low-exponent RSA signature scheme [BR96] for the certificates, so that certificate verification is cheap. Second, one could use a low-exponent RSA [BR93b, BR94] for the encryption scheme. Third, for the signature scheme that user $U_{i'}$ uses to sign messages in the protocol, one could use Schnorr's signature scheme [Sch91]. In this scheme, one can perform some "off line" computations so that the "on line" cost of signing is extremely cheap. Thus, if user $U_{i'}$ is a server that is heavily loaded at some times, but not at others, the server can perform the "off line" computations during non-peak hours, and thereby provide a fast response time during peak hours.*

9 Anonymous Users

In this section, we extend our formal model of security for session key protocols to model the setting where one of the two users establishing a session key is *anonymous*. By an anonymous user, we simply mean one without a certificate, so perhaps a better term would be *unauthenticated*.

Of course, to the non-anonymous user, the session key protocol itself can offer little protection, since the anonymous user could be the adversary, or an honest user, and the non-anonymous user cannot tell the difference. Typically, however, an anonymous user will authenticate himself within the secure session using a password. We discuss this in more detail in §9.4.

9.1 Definitions

Adding anonymous users is simple. We create a special user U_0 with the special identity $ID_0 = \mathsf{anonymous}$. An entity that wants to run the protocol as an anonymous user will simply utilize a user instance I_{0j} for this purpose. Note that user U_0 has no associated secret key or long-term state, and is considered to be pre-initialized.

Beyond the introduction of this special user U_0, the rest of the model stays exactly the same as before, with one exception. The compromise rule **C3** (see §3.1.4) regulating the connection assignment for a user instance I_{ij} is replaced by:

C3* The connection assignment compromise is legal provided PID_{ij} is not assigned to a user, or $PID_{ij} = \mathsf{anonymous}$.

That is, the ideal-world adversary A^* is always free to make a connection assignment of *compromise* for a user instance I_{ij} if $PID_{ij} = \mathsf{anonymous}$.

Remark 10 *Intuitively, this relaxation of the compromise rule is necessary, since as we remarked above, an anonymous user may really be the adversary himself. Note, however, that for any protocol that satisfies our definition of security, A^* cannot* always *choose to* compromise *such a user instance.*

Remark 11 *The fact that we group together all anonymous user instances under a single user* U_0 *is simply a technical convenience. In practice, such user instances will not typically be running on the same machine, and may indeed be running on the same machine on which a non-anonymous user is running.*

Remark 12 *There is nothing in our definition that rules out a protocol that attempts to establish a key between two anonymous user instances. However, our definition would provide no security guarantees for such a key, so we shall not consider such protocols here.*

9.2 A Diffie-Hellman based protocol

We can extend protocol **DHKE** to obtain the following protocol **A-DHKE**.

$U_0 \to U_{i'} :\ g^x,$
where $x \in \mathbf{Z}_q$ is chosen at random.
$U_{i'} \to U_0 :\ g^y, k, sig_{i'}(g^x, g^y, k, \textsf{anonymous}), cert_{i'},$
where $y \in \mathbf{Z}_q$ is chosen at random, and k is a random hash function index.

As usual, the anonymous user checks all the relevant signatures. Both users compute the session key as $H_k(g^{xy})$. Also, we assume here that the group used for the computations is shared by all anonymous users. It can wither be "hardwired" into all users, or can be part of the public key PK_T.

Note that we consider protocol **A-DHKE** to be an extension of protocol **DHKE**, meaning that both anonymous and non-anonymous key exchanges may take place in any combination.

Theorem 3 *Protocol* **A-DHKE** *is secure, under the DDH assumption, and assuming secure signatures.*

The proof is just a slight modification of the proof of Theorem 1. The only thing that changes is the calculation of the connection assignment in case (1a) of that proof. In this case, we shall *compromise* $I_{i'j'}$ if either

- $PID_{i'j'}$ is not assigned to a user, or
- $PID_{i'j'} = \textsf{anonymous}$ and there does not exist an anonymous user instance I_{0j} with $PID_{0j} = ID_{i'}$ that sent g^x as its first message in the protocol.

The rest of the proof goes through without change.

Remark 13 *The reader may have noticed that protocol* **A-DHKE** *is vulnerable to the following PKI "attack." An adversary could take the public key of an honest user* B*, and register a name* $\hat{B}$ *with the same public key. Then by replacing* B*'s certificate in the protocol with* $\hat{B}$*'s certificate, the adversary could make the anonymous user think he has a key established with* $\hat{B}$*, whereas he really shares a key with* B*. Our definition of security does not rule out this "attack," and it is easy to carry out this "attack" on protocol* **A-DHKE***. However, we argue that this "attack" is spurious. Indeed, if the session key is used for the purpose of establishing a secure session, then the adversary can* always *achieve the same effect much more easily, as follows. He could separately establish session keys with the anonymous user and with* B*, and then just act as a "bridge" between these two users, decrypting and re-encrypting messages as necessary. Also note that one could try to prevent the above "attack" on* **A-DHKE** *by having* $U_{i'}$ *include his identity* $ID_{i'}$ *in the message that it signs in the second flow; however, although this would make the "attack" more difficult to mount, one could not rule it out under the standard definition of secure signatures.*

9.3 Two encryption based protocols

We extend protocol **EKE** in two different ways. The first protocol, **A-EKE-1**, runs as follows.

$U_i \to U_0$: $r, cert_i$,
where r is a (sufficiently long) random bit string.

$U_0 \to U_i$: $E_i(K, \mathsf{anonymous}, r)$,
where K is a random bit string.

As usual, the agreed upon session key is K, and user U_i checks that the values embedded in encrypted message are correct.

Theorem 4 *Protocol* **A-EKE-1** *is secure assuming secure signatures and non-malleable encryption.*

The proof of this theorem is just a slight modification of the proof of Theorem 2. We need to modify only case (2b) of that proof when $PID_{ij} = \mathsf{anonymous}$.

- If α was not created by an anonymous user instance, we let the protocol run to completion, and *compromise* I_{ij}, should it accept. If I_{ij} accepts, this involves an implicit decryption of α, but by the logic of the protocol, α cannot be one of the ciphertexts from step (1b) that we are not allowed to decrypt.
- Otherwise, if α was created by a (necessarily unique) user instance $I_{0j'}$, then there are two sub-cases.
  - If the value r received by $I_{0j'}$ matches that sent by I_{ij}, then we *connect* I_{ij} to $I_{0j'}$.
  - Otherwise, we let I_{ij} reject, since that is what I_{ij} would anyway do.

  In both sub-cases, we do not decrypt α.

Remark 14 *We have written protocol* **A-EKE** *with the non-anonymous user U_i in the role of the initiator, and the anonymous user U_0 in the role of the responder. In a typical setting, however, the non-anonymous user is a "server," and the anonymous user is a "client." In such a setting, we would typically expect the client to initiate the protocol. If that is the case, then the protocol must contain an initial flow from the client to the server, just to get things started, so the protocol would actually require three flows.*

Remark 15 *This protocol can be implemented so that the computational burden on the client is very minimal, by using low-exponent RSA based encryptions and signatures. This might be useful in some settings where the client is computationally limited; unfortunately, in many settings, is turns out to be the server who is already computationally overburdened.*

Here is an alternative, rather amusing protocol **A-EKE-2**. Let f be a pseudo-random function family, indexed by a key K.

$U_i \to U_0$: $cert_i$.

$U_0 \to U_i$: $E_i(K, \mathsf{anonymous})$,
where K is a random bit string.

$U_i \rightarrow U_0 : r,$
where r is a random bit string.

In this protocol, the agreed upon session key is computed as $f_K(r)$.

Theorem 5 *Protocol* **A-EKE-2** *is secure assuming secure signatures, non-malleable encryption, and a secure pseudo-random function.*

We leave this proof as an exercise for the reader.

Remark 16 *Since the last flow from U_i to U_0 is not authenticated, if the adversary modifies the value r while it is in transit, then both user instances will simply be permanently isolated. Thus, while the protocol satisfies our definition of security, it does not guarantee* explicit key confirmation *for either originator or responder (see §3.4, point (12)).*

Remark 17 *The point of this protocol is to address the issue raised in Remark 14. In the client/server setting described there, protocol* **A-EKE-2** *would in the worst case require four flows. However, if the client happens to already have the certificate of the server stored locally, only two flows are necessary. Indeed, protocols* **A-EKE-1** *and* **A-EKE-2** *could be combined so that the client uses the former if it does not already have the server's certificate, and the latter if it does.*

9.4 The principle application: secure sessions

We can extend the formal security model and implementation sketched in §4 for a secure session protocol to include anonymous users. Actually, nothing changes, except that we let the adversary make connection assignments using the modified rule **C3***, described above.

We can go one step further, if we wish, and consider the situation where the anonymous user authenticates himself to the non-anonymous user by means of a password. Now, once an anonymous user has an established a secure session, he can simply send his password through the secure channel. Sometimes this message can even be piggy-backed on the last message of the key exchange protocol, in which case there is no extra communications cost. There is really nothing more to it. It is easy to see that given the properties of a secure session, such a password-based scheme will have all the properties one could possibly hope for; in particular,

- an adversary trying to guess a password cannot do any better than an "on line" password guessing attack, and
- if an anonymous user establishes a session and then authenticates himself within the session using a password, an adversary cannot afterwards "hijack" the session.

As already mentioned in §1, our approach to this problem is perhaps an attractive alternative to the approach taken by many other authors. By appropriately defining secure key exchange in the anonymous user setting, we can easily analyze such protocols without worrying about passwords. Then using a standard implementation for secure sessions on top of the key exchange protocol, and passing the password through the secure channel, we get a password-authenticated secure session essentially "for free." Moreover, one can implement a single "low layer" communication protocol that implements secure sessions with anonymous users, without any passwords; on top of this, one can implement arbitrary protocols that may or may not require passwords: "telnet," "FTP," etc.

10 A Formal Model for Security Against Adaptive Corruptions

We now extend our formal security model to deal with *adaptive corruptions.* In an adaptive corruption, the adversary obtains a user's long-term secret, but nothing else.

In §14, we shall consider *strong* adaptive corruptions, in which the adversary obtains ephemeral data as well as long-term secrets.

Note that our formal model already allows one to model the exposure of session keys (using appropriate *application* operations), but as was pointed out in §4, a typical implementation of a secure session protocol built on top of a secure key exchange protocol will not be secure (in the sense of simulatability) if we allow the exposure of session keys.

Additionally, we allow a corruption to encompass the possibility of a fault in the certificate authority, whereby the adversary obtains a certificate of his choice on a user's identity.

Our approach will be that when a user is corrupted, that user continues to play along in all interactions as usual, following its protocol correctly. Of course, the adversary, having obtained the secret key, can interact with other users, "pretending" to be this user.

We need to modify both the real system model and the ideal system model. The definition of security, defined in terms of termination, liveness, and simulatability, will remain exactly the same.

10.1 The real system

The adversary may execute a *corrupt user* operation, which takes the form

$$(\mathsf{corrupt\ user}, i).$$

The adversary specifies a user U_i that has been previously initialized, and obtains the user's long-term state LTS_i.

The following two records are added to the transcript:

$$(\mathsf{corrupt\ user}, i),$$

and

$$(\mathsf{implementation}, \mathsf{corrupt\ user}, LTS_i).$$

Additionally, at any point in time after a *corrupt user* operation, we allow the adversary to perform *register* operations using the identity ID_i

Note that since LTS_i may change over time, we allow a *corrupt user* operation to be applied to an already corrupted user.

10.2 The ideal system

The adversary may execute a *corrupt user* operation,

$$(\mathsf{corrupt\ user}, i),$$

specifying a previously initialized user U_i. The record

$$(\mathsf{corrupt\ user}, i)$$

is added to the transcript.

No information is given to the adversary in the ideal world when a user is corrupted.

The only other change to the model is that we have to modify the rules in §3.1.4 governing the legality of the connection assignments made during during a *start session* operation applied to a user instance I_{ij}.

The change here is minimal. We change rule **C3** as follows:

C3′ The connection assignment compromise is legal if either

- PID_{ij} is not assigned to a user,
- PID_{ij} is assigned to a corrupted user, or
- user U_i is corrupted.

Remark 18 *It is important to notice what* does not *change. In particular, the ideal-world adversary is free to make the connection assignments* create *and* connect*, regardless of whether any of the relevant users have been corrupted—he is never forced to make the connection assignment* compromise*. This keeps our definitions simple and natural: if Alice thinks she is talking to a user Bob, but Bob has had his long-term secret key exposed, then Alice may indeed be talking to Bob or to the adversary. This gives our simulators the flexibility they need to deal with situations where a user is corrupted while it is in the middle of an on-going protocol.*

10.3 A more conservative compromise rule

Notice that we allow a connection assignment of compromise for I_{ij} if user U_i itself has been corrupted. While this may seem fairly natural, one could make a more conservative compromise rule that required that PID_{ij} is not assigned to an uncorrupted user—corruption of U_i would not be sufficient by itself.

Such a conservative compromise rule makes a difference.

First, it would make a difference in the inferences one could make in higher-level protocols. For example, Alice could infer that a supposed message from Bob in a secure session was indeed from Bob unless Bob was corrupted—it would not matter if Alice's long-term secret key had been exposed or not. This is precisely the same inference that Alice could draw if the message were authenticated directly with a digital signature. This inference could not be drawn under the liberal compromise rule.

Second, it would make a difference in which protocols would be considered secure. In §12 we will see examples of protocols that are secure under the liberal compromise rule but not secure under the conservative compromise rule.

In the sequel, we will adopt the liberal compromise rule as our "default" rule, but will point out those situations where imposing the conservative rule would make a difference.

10.4 The principle application: secure sessions

We continue our discussion about the principle application of session key exchange protocols, namely, to build a secure session protocol. We can adapt the formal model and implementation of a secure session protocol sketched in §4 to deal with adaptive corruptions. In fact, nothing really changes, except the rules for connection assignments. The important thing to note, however, is that if a user instance starts a session, and that session is initially uncompromised, then *it will never be compromised*, even if one of the relevant parties is corrupted while the session is ongoing.

10.5 Non-forward security against adaptive corruptions

Our definition of security against adaptive corruptions captures the intuitive notion of *forward security*. One can easily formulate a notion of *non-forward* security against adaptive corruptions, wherein the adversary in the ideal world would also obtain all the relevant session keys, which means, all the session keys established by instances of the corrupted user U_i, as well as all session keys $K_{i'j'}$ with $PID_{i'j'} = ID_i$. Absolutely nothing else would change: in particular, none of this extra information would be logged in the ideal world or real world transcripts.

This notion of non-forward security does not seem to be very attractive, for two reasons. First, it does not seem to be any easier to achieve non-forward security than to achieve forward security. Second, it would be very difficult to build a practical secure session protocol that was secure against adaptive corruptions on top of such a key exchange protocol.

10.6 Anonymous users

It is trivial to adapt the definition of security with respect to anonymous users (see §9) to incorporate adaptive corruptions. All that changes is rule **C3′** is §10.2, so that the the connection assignment compromise is also legal when $PID_{ij} = \mathsf{anonymous}$ (as in rule **C3*** in §9.1). As the anonymous user U_0 does not have any long-term state, it cannot be corrupted.

11 Interlude: On the insecurity of protocols DHKE and EKE against adaptive corruptions

In this section, we argue that protocols **DHKE** and **EKE** are insecure against adaptive corruptions.

11.1 Protocol DHKE against adaptive corruptions

Consider a user instance I_{ij} who is engaging in the protocol (as an initiator) with a compatible user instance $I_{i'j'}$ (as a responder). Suppose that the first message in the protocol is delivered to $I_{i'j'}$, so that $I_{i'j'}$ computes a session key $K_{i'j'} = H_k(g^{xy})$, along with a response message to be sent to back to I_{ij}. At this point, the adversary reveals $K_{i'j'}$ using an appropriate *application* operation. Next, the adversary corrupts user $U_{i'}$ before the response message is delivered to I_{ij}, so that the adversary obtains the signing key of $U_{i'}$. If $I_{i'j'}$'s response message was

$$(g^y, k, sig_{i'}(g^x, g^y, k, ID_i)),$$

the adversary instead delivers the message

$$(h, k, sig_{i'}(g^x, h, k, ID_i))$$

to I_{ij}, where h is a group element chosen in some mysterious way by the adversary. The adversary can do this, since it has the signing key of user $U_{i'}$. Now, I_{ij} will accept and compute its session key $K_{ij} = H_k(h^x)$, and we also reveal this session key using an appropriate *application* operation.

Now, at the point in time when $I_{i'j'}$ generated its session key, it was not corrupted, so the only possible connection assignment for $I_{i'j'}$ is create. This means that $K_{i'j'}$ should be indistinguishable from a random key. But we cannot hope to prove this under the standard DDH assumption, since the additional information $H_k(h^x)$ is available to any statistical test.

We believe that the problem is a fundamental one, having more to do with the inherent *malleability* of Diffie-Hellman based encryption, than with the particulars of our formal model.

11.2 Protocol EKE against adaptive corruptions

At best, it is clear that all we could hope for is that protocol **EKE** is secure against adaptive corruptions in the non-forward sense described in §10.5. This is because if we corrupt a user U_i and obtain his private decryption key, we can easily compute all of the session keys that were ever sent encrypted to it in the protocol.

But things are much worse than that. Consider the following scenario. Say we have n pairs of users U_i and $U_{i'}$, with all users distinct. Now we let all n pairs run the session key protocol, and start using their session keys in application protocols. Now the adversary corrupts a random subset of the U_i users, obtaining their long-term decryption keys.

How could we simulate this? When the session key protocols terminate, we want to substitute all of the actual session keys with random keys before the users start using them in an application protocol. We have to, because we have no idea which subset the adversary will corrupt. But when we obtain the decryption keys, we will have an inconsistent transcript: the actual ciphertexts decrypt to values different from the substituted session keys.

12 Diffie-Hellman Based Protocols for Adaptive Corruptions

In this section, we examine three Diffie-Hellman based protocols that are secure against adaptive corruptions. We call these **DHKE-**n for $n \in \{1, 2, 3\}$.

12.1 Protocol DHKE-1

We now show how to modify protocol **DHKE** to obtain a protocol that is secure against adaptive corruptions. We call this protocol **DHKE-1**. It is essentially the same as **DHKE**, but with an additional "key confirmation" flow.

The system set up is the same as before. Additionally, we need a pseudo-random bit generator *BitGen*.

The protocol runs as follows.

$U_i \to U_{i'} :\ g^x, sig_i(g^x, ID_{i'}), cert_i,$
where $x \in \mathbf{Z}_q$ is chosen at random.

$U_{i'} \to U_i :\ g^y, k, sig_{i'}(g^x, g^y, k, ID_i), cert_{i'},$
where $y \in \mathbf{Z}_q$ is chosen at random, and k is a random hash function index.

$U_i \to U_{i'} :\ k_1,$
where $(k_1, k_2) = BitGen(H_k(g^{xy}))$.

The agreed upon session key is k_2, where $(k_1, k_2) = BitGen(H_k(g^{xy}))$ as above. In addition to all the usual signature checks, user $U_{i'}$ checks the value k_1 is as expected. We assume that k_1 is a sufficiently long bit string (of length, e.g., linear in the security parameter).

Theorem 6 *Protocol* **DHKE-1** *is secure against adaptive corruptions, under the DDH assumption, and assuming secure signatures and that BitGen is a secure pseudo-random bit generator.*

We now prove this theorem, which follows the general outline of all our other proofs so far. That is, we show how to transform a real world A into an equivalent ideal world A^*.

Let G_i denote the description of the group that is contained in $cert_i$.

Case 1. Suppose a user instance I_{ij} has just received its last message, and all the signatures are valid.

Case 1a. Suppose PID_{ij} is assigned to user $U_{i'}$, and some instance $I_{i'j'}$ with $PID_{i'j'} = ID_i$ received the g^x, G_i values sent by I_{ij} and sent the g^y, k values received by I_{ij}. In this case, we make I_{ij} accept in the ideal world, and we give it the connection assignment `create`, whereby the ring master chooses a random string for the session key. Additionally, we will generate a random string k_1, which we will call the *confirmation key* of I_{ij} for future reference. Note that we do all of this, *even if user* $U_{i'}$ *has been corrupted.*

Case 1b. Suppose the condition in case (1a) does not hold. By the logic of the protocol, the only way this could happen is if PID_{ij} is not assigned to an uncorrupted user, or user U_i is itself corrupted. We extract the computed session key and confirmation key from I_{ij} in the real world, and *compromise* I_{ij} using the computed session key.

Case 2. Suppose that I_{ij} has received its last message, and the signatures do not check. Then we let I_{ij} reject in the ideal world, which is what it would do anyway in the real world.

Case 3. Suppose a player instance $I_{i'j'}$ has just received its last message, and all the signatures check.

Case 3a. Suppose $PID_{i'j'}$ is assigned to user U_i, and some instance I_{ij} with $PID_{ij} = ID_{i'}$ sent the g^x, G_i values received by $I_{i'j'}$ and received the g^y, k values sent by $I_{i'j'}$. Let k_1 be the confirmation key of I_{ij} (see case (1a)). We then test if the last message received by $I_{i'j'}$ is equal to k_1. If not, we let $I_{i'j'}$ reject. Otherwise, we *connect* $I_{i'j'}$ to I_{ij}. As this our only rule for *connecting* two user instances, it is easy to see that no other user instance has connected to I_{ij}, and hence it is still *isolated.* Note that we do all of this, *even if user* U_i *has been corrupted.*

This last point is crucial. User U_i may have been corrupted after I_{ij} accepted and sent its last message, and in the meantime, I_{ij} may very well have started to *use* its session key. In the simulation we have already substituted K_{ij} with an idealized random key, and so we cannot afford to *compromise* $I_{i'j'}$ at this point. This is is the situation referred to in Remark 18.

Case 3b. Suppose the condition in case (3a) does not hold. There two further sub-cases to consider.

Case 3b′. Suppose that at this point in the game, user $U_{i'}$ is not corrupted, and that g^x, G_i came from a user instance I_{ij} with $PID_{ij} = ID_{i'}$. Then in the ideal world, we simply make $I_{i'j'}$ reject. See below for a discussion of why this is valid.

Case 3b″. If we reach this sub-case, by the logic of the protocol, the only way this could happen is if $PID_{i'j'}$ is not assigned to an uncorrupted user, or user $U_{i'}$ itself is corrupted. We then extract both the session key and the confirmation key from $I_{i'j'}$. We then test the if the received message is equal to the computed confirmation key. If this test fails, we let $I_{i'j'}$ reject, just as it would in the real world. Otherwise, we let it accept, and *compromise* $I_{i'j'}$ in the ideal world using the computed session key.

Case 4. Suppose that $I_{i'j'}$ has received its last message, and the signatures do not check. Then we let $I_{i'j'}$ reject in the ideal world, which is what it would do anyway in the real world.

Clearly, we have not broken any of the rules governing the ideal world simulation. But we also have to show that the resulting simulation is faithful to the real world. Notice that the only time we make two user instances partners, the corresponding values of G_i, g^x, g^y and k match. Because this condition is symmetric, we will never compromise one key, while substituting the other instance's key with a random key.

The faithfulness of the simulation now follows from the DDH assumption, but there is one subtle point that requires further comment: the "forced" rejection by user instance $I_{i'j'}$ in case

(3b$'$). We have to argue that this is what would have happened in the real world, since we never asked $I_{i'j'}$ what he really wanted to do. But consider the user instance I_{ij} referred to in that subcase. Since user $U_{i'}$ has not been corrupted at this point, the adversary could not have forged any messages on behalf of user $U_{i'}$. Therefore, either I_{ij} has not accepted (either it rejected or has not yet received the second message in the protocol), or it has accepted using some value $g^{y'}$ generated independently by another instance of user $U_{i'}$. So at this point, under the DDH assumption, the value k_1 that $I_{i'j'}$ is expecting is (computationally) independent from the adversary's view in the real world. Thus, letting $I_{i'j'}$ reject is the right action.

That completes the proof of the theorem.

Remark 19 *Key confirmation is a mysterious and ancient tradition practiced by protocol designers, as was already alluded to in §3.4, point (12). There has never been a satisfying explanation of why they did this. Now we know: to allow a proof of simulatability against adaptive corruptions.*

We can extend protocol **DHKE-1** to deal with anonymous users, obtaining the following protocol **A-DHKE-1**. As in protocol **A-DHKE** (see §9.2), we assume that all anonymous users work with a shared group.

$U_0 \to U_{i'}$: g^x,
where $x \in \mathbf{Z}_q$ is chosen at random.

$U_{i'} \to U_0$: $g^y, k, sig_{i'}(g^x, g^y, k, \textsf{anonymous}), cert_{i'}$,
where $y \in \mathbf{Z}_q$ is chosen at random, and k is a random hash function index.

$U_0 \to U_{i'}$: k_1,
where $(k_1, k_2) = BitGen(H_k(g^{xy}))$.

We leave it to the reader to verify that this protocol is secure with respect to our definitions of anonymous users (§9) and adaptive corruptions (§10.6).

12.2 Protocol DHKE-2

We presented protocol **DHKE-1** as we did because it is a minimal modification of **DHKE** and has an interesting proof of security. An alternative is the following **DHKE-2**, which is closely related to STS.

The set up is just as in **DHKE**.

$U_i \to U_{i'}$: $g^x, cert_i$,
where $x \in \mathbf{Z}_q$ is random.

$U_{i'} \to U_i$: $g^y, k, sig_{i'}(g^x, g^y, k, ID_i), cert_{i'}$,
where $y \in \mathbf{Z}_q$ is random and k is random.

$U_i \to U_{i'}$: $sig_i(g^x, g^y, k, ID_{i'})$.

Theorem 7 *Protocol* **DHKE-2** *is secure against adaptive corruptions, under the DDH assumption, and assuming secure signatures.*

The proof of security for **DHKE-2** is actually more straightforward than for **DHKE-1**.

Case 1. Suppose I_{ij} has just accepted.

Case 1a. Suppose PID_{ij} is assigned to user $U_{i'}$ and some instance $I_{i'j'}$ with $PID_{i'j'} = ID_i$ received the g^x, G_i values sent by I_{ij} and sent the g^y, k values received by I_{ij}. Then we *create* I_{ij}, and K_{ij} is replaced with a random key. We will see below that $I_{i'j'}$ is not *compromised*, it either rejects or *connects* to I_{ij}. It follows from the DDH assumption that the substitution will go unnoticed.

Case 1b. If the condition in case (1a) does not hold, then by the logic of the protocol, either U_i is corrupted, or PID_{ij} is not assigned to an uncorrupted user. So we *compromise* I_{ij}.

Case 2. Suppose $I_{i'j'}$ has just accepted.

Case 2a. Suppose $PID_{i'j'}$ is assigned to user U_i and some instance I_{ij} with $PID_{ij} = ID_{i'}$ sent the g^x, G_i value sent by I_{ij} and received the g^y, k value sent by $I_{i'j'}$. Then we make $I_{i'j'}$ *connect* to I_{ij}. From the arguments above, I_{ij} has not been *compromised*, and it is thus clear that I_{ij} is still isolated.

Case 2b. Suppose the condition in case (2a) does not hold. Then by the logic of the protocol, $PID_{i'j'}$ is not assigned to an uncorrupted user, so we *compromise* I_{ij}.

12.3 Protocol DHKE-3

Although protocols **DHKE-1** and **DHKE-2** are secure against adaptive corruptions using the liberal compromise rule, it is perhaps interesting to note that they are apparently not secure under the conservative compromise rule (§10.3). To achieve security in this stricter sense, there seems to be no easy way to repair **DHKE-1**, but **DHKE-2** can be relatively easily repaired as follows. We call this protocol **DHKE-3**.

Before describing **DHKE-3**, let us see where things go wrong for **DHKE-1** and **DHKE-2** with the conservative compromise rule.

In the proof of Theorem 6, consider case (1b). The conversations may not have matched because the description of the group G_i may not have matched, which may have happened because user U_i was corrupted, not because $U_{i'}$ was corrupted. Under the conservative compromise rule, we are not allowed to *compromise* here. The problem is even worse in case (3b$''$). There, we really need to *compromise* if $U_{i'}$ is corrupted, but again, the conservative compromise rule forbids this. The same problem that arose with G_i in the proof of Theorem 6 also arises in the proof of Theorem 7, but that is the only problem that arises.

Now we describe **DHKE-3**. The set up is the same as above. Recall that G_i is the description of the group used by user U_i. In the previously discussed Diffie-Hellman based protocols, this information was in the certificate of user U_i. In this protocol, we do not require this—under the conservative compromise rule, it does not help.

$U_i \to U_{i'} :\ G_i, g^x, cert_i,$
where $x \in \mathbf{Z}_q$ is random.

$U_{i'} \to U_i :\ g^y, k, sig_{i'}(G_i, g^x, g^y, k, ID_i), cert_{i'},$
where $y \in \mathbf{Z}_q$ is random and k is random.

$U_i \to U_{i'} :\ sig_i(G_i, g^x, g^y, k, ID_{i'}).$

Theorem 8 *Protocol* **DHKE-3** *is secure against adaptive corruptions, using the conservative compromise rule, under the DDH assumption, and assuming secure signatures.*

The proof is almost identical to that of Theorem 7. The only difference is case (1b). Here we can conclude that if there was no matching $I_{i'j'}$, then it must be the case that PID_{ij} is not assigned to an uncorrupted user. We omit further details.

We can also extend protocol **DHKE-3** to handle anonymous users, obtaining the following protocol **A-DHKE-3**.

$U_i \to U_0 :\ G_i, g^x, cert_i,$
where $x \in \mathbf{Z}_q$ is random.

$U_0 \to U_i :\ g^y, k,$
where $y \in \mathbf{Z}_q$ is random and k is random.

$U_i \to U_0 :\ sig_i(G_i, g^x, g^y, k, \mathsf{anonymous}).$

We leave it to the reader to verify that this protocol is secure with respect to our definitions of anonymous users (§9) and adaptive corruptions (§10.6). Like protocol **DHKE-3**, this protocol is secure using the conservative compromise rule.

13 An Encryption Based Protocol for Adaptive Corruptions

We now present a simple two pass key exchange protocol using public key encryption. It is very similar to our protocol **EKE**, except that here we use *ephemeral* public keys, instead of a fixed, long-term public key. Alternatively, one can view it as a modification of protocol **DHKE**, where we replace malleable Diffie-Hellman encryption by a non-malleable encryption scheme. We call this protocol **EKE-1**.

In this scheme, each user generates a public key/private key pair for a signature scheme. This public key is what goes in his certificate. Each user also uses a key generation algorithm $KeyGen()$ for a non-malleable public key cryptosystem. The output of $KeyGen()$ is a public key/private key pair (E, D).

The protocol runs as follows.

$U_i \to U_{i'} :\ E, sig_i(E, ID_{i'}), cert_i,$
where $(E, D) = KeyGen()$.

$U_{i'} \to U_i :\ \alpha = E(K), sig_{i'}(\alpha, E, ID_i), cert_{i'},$
where K is a random bit string.

The agreed upon session key is K, which user U_i obtains by computing $D(\alpha)$. As usual, both users check the relevant signatures.

Note that unlike protocol **EKE**, user $U_{i'}$ does not need to include his identity $ID_{i'}$ in the encrypted message.

Theorem 9 *Protocol* **EKE-1** *is secure against adaptive corruptions, assuming secure signatures, and assuming the encryption scheme is non-malleable.*

We now prove this theorem, following the outline of all the previous proofs.

Case 1. Suppose $I_{i'j'}$ has just terminated successfully.

Case 1a. Suppose the value E received by $I_{i'j'}$ came from some I_{ij} such that $PID_{i'j'} = ID_i$ and $PID_{ij} = ID_{i'}$. Then we *create* $I_{i'j'}$, and replace the session key $K_{i'j'}$ with a random string.

Case 1b. Suppose the condition in case (1a) does not hold. Then by the logic of the protocol and the security of the signatures, it must be the case that $PID_{i'j'}$ is not assigned to an uncorrupted user. So we *compromise* $I_{i'j'}$.

Case 2. Suppose I_{ij} has just terminated the protocol successfully.

Case 2a. Suppose that there is a user instance $I_{i'j'}$ such that $PID_{i'j'} = ID_i$ and $PID_{ij} = ID_{i'}$, and $I_{i'j'}$ received the value E sent by I_{ij} and sent the value α received by I_{ij}. Then we *connect* I_{ij} to $I_{i'j'}$. Since the values E generated by different instances of user U_i are (almost surely) unique, this connection assignment will (almost surely) be valid. In connecting I_{ij} to $I_{i'j'}$, we set the session key K_{ij} equal to $K_{i'j'}$, and thus we do not bother to decrypt α.

Case 2b. Suppose the condition in (2b) does not hold. Then by the logic of the protocol and the security of the signatures, it must be the case that PID_{ij} is not assigned to an uncorrupted user. So we *compromise* I_{ij}. As usual, we extract the actual session key from I_{ij}. This makes implicit use of the decryption function of user instance I_{ij}. It is easily verified that this decryption does not affect the indistinguishability of the substitution made in case (1a), since if we decrypted the ciphertext from case (1a) under the decryption key from (1a), we would be in case (2a), and not case (2b).

That completes the proof of the theorem.

Remark 20 *It is easy to see that protocol* **EKE-1** *remains secure even under the conservative compromise rule (§10.3).*

Remark 21 *The differences between the Diffie-Hellman based protocols and* **EKE-1** *illustrate an interesting phenomenon. The real problem with protocol* **DHKE** *in the face of adaptive corruptions is the* malleability *of Diffie-Hellman based encryption. This problem can be fixed either by using a* non-malleable *cryptosystem, or by adding extra interaction.*

Remark 22 *We have to generate a new public key/private key for encryption with every run of the protocol. For RSA-based schemes, this can be impractical, as prime number generation can be quite slow. A more practical approach would be to use a Diffie-Hellman based scheme, such as Cramer-Shoup [CS98] or Fujisaki-Okamoto [FO99], but to generate the group just once, and to use the same group with each run of the key exchange protocol (which does not affect the security).*

14 Strong Adaptive Corruptions

In this section, we consider even more powerful real-world adversaries; namely, adversaries which can adaptively corrupt users, and when a user is corrupted, not only does the adversary obtain the user's long-term secret, but he also obtains any internal, ephemeral data that has not been *explicitly erased.* We call such a corruption a *strong corruption* to distinguish it from the notion of corruption we have already studied in which the adversary obtains only the long-term secret of a user.

One could consider a model that allows a mix of corruptions and strong corruptions, but we shall not do that here, if only for the sake of simplicity. Instead, we will assume that there are only strong corruptions in this section.

In defining security against strong adaptive corruptions, in increasing the power of the adversary, we have to relax the security guarantees. Therefore, under our definitions, security against strong adaptive corruptions *does not* imply security against adaptive corruptions. The two notions of security are incomparable.

In §14.1, we give a precise definition of strong corruptions for a real-world adversary; in §14.2, we identify those session keys that are *inherently vulnerable* when a user is strongly corrupted, which motivates our definition of strong corruptions in the ideal world in §14.3.

In §14.4 we present a key exchange protocol that is secure under our definition.

In §14.5 we shall sketch a formal definition of a secure session protocol in the context of strong adaptive corruptions.

In §14.6 we shall show how to efficiently implement such a secure session protocol on top of a key exchange protocol satisfying our definition of security.

None of the above applies to the anonymous user setting, which requires special treatment. We discuss this in §14.7.

14.1 Strong corruptions in the real world

When a user is strongly corrupted, we assume the real-world adversary obtains that user's long-term secret, as well as all the unerased data of each of that user's instances that is still *active*, i.e., still running the key exchange protocol. We can assume that when an instance of the key exchange protocol terminates, all internal data is erased. Let us emphasize that the adversary *does not* obtain any session keys. The reason for this is that session keys belong to higher-level protocols that use the session keys, and they have the right to erase these keys. Whatever application-specific data we wish to make accessible to the adversary when a user is corrupted we can model by an appropriate use of *application* operations, in conjunction with the strong corruption operation. This may or may not include session keys.

So the only change to the real-world adversary is that he may execute the operation

$$(\textsf{strong corrupt user}, i),$$

where U_i is an initialized user. Upon execution of this operation, the adversary obtains that U_i's long-term secret, as well as all the unerased data of each I_{ij} that is still *active*. Additionally, as for ordinary corruptions (see §10.1), the adversary may subsequently register the identity ID_i without any of the usual restrictions.

Upon execution of this operation, the following records are added to the transcript:

$$(\textsf{strong corrupt user}, i),$$

and

$$(\textsf{implementation}, \textsf{strong corrupt user}, \textit{exposed data}),$$

where *exposed data* consists of the long-term secret and unerased ephemeral data of U_i, as described above.

As for ordinary corruptions, the user instances belonging to the corrupted user continue to play along, and we allow a user to be corrupted multiple times.

14.2 Inherently vulnerable keys

Note that when a user U_i is strongly corrupted in the real world, some session keys held by *other* users may be vulnerable. For example, suppose a user instance $I_{i'j'}$ with $PID_{i'j'} = ID_i$ has completed the key exchange protocol and is currently *isolated*, that is, it has connection assignment $\textsf{create}$, and no user instance has *connected* to it. This implies that there may be some *active* user instance I_{ij} that would eventually *connect* to $I_{i'j'}$, and the internal state of such an I_{ij} contains enough information to compute $K_{i'j'}$. If U_i is corrupted at this moment, then the real-world adversary can compute $K_{i'j'}$. The inherent vulnerability of such keys is the motivation for our definition of strong corruptions in the ideal world: in the ideal world, the adversary is given all such *inherently vulnerable keys*, but nothing more.

14.3 Strong corruptions in the ideal world

The changes to the ideal system are quite minimal.

The ideal-world adversary may execute the operation

$$(\mathsf{strong\ corrupt\ user}, i)$$

to corrupt U_i. The *first* time U_i is corrupted, the ideal-world adversary is given all session keys of the form $K_{i'j'}$, where $I_{i'j'}$ is a user instance such that $PID_{i'j'} = ID_i$ and $I_{i'j'}$ is currently *isolated.*

This operation is logged in the transcript as

$$(\mathsf{strong\ corrupt\ user}, i).$$

Note that no information about the keys given to the adversary is logged in the transcript.

Those are the *only* changes. We will use only the default, i.e., liberal, compromise rule in conjunction with strong corruptions.

Remark 23 *In making connection assignments in this model, we do not really need the flexibility discussed in Remark 18. Indeed, without loss of generality, we could require that the ideal-world adversary make the connection assignment* compromise *whenever this was legal.*

14.4 A secure key exchange protocol

It turns out that protocol **DHKE-1** in §12.1 satisfies our definition of security against strong adaptive corruptions, assuming internal data is appropriately erased. In particular, this means that before a responder $I_{i'j'}$ has sent the second flow, it has erased all internal data except for k_1 and k_2. Of course, we assume that when the protocol terminates, *all* internal data is erased.

Theorem 10 *Protocol* **DHKE-1** *is secure against strong adaptive corruptions, under the DDH assumption, and assuming secure signatures and a secure pseudo-random bit generator.*

The proof follows the same lines as all of our other proofs. We begin by describing the connection assignments.

Case 1. Suppose (originator) I_{ij} has just accepted.

Case 1a. If it is legal to *compromise* I_{ij}, we do so, extracting the key K_{ij} from the real-world I_{ij}.

Case 1b. Otherwise, we *create* I_{ij}, and let the ring master substitute the real-world key K_{ij} with an ideal, random key.

Case 2. Now suppose (responder) $I_{i'j'}$ has just accepted.

Case 2a. If it is legal to *compromise* $I_{i'j'}$, we do so, extracting the key $K_{i'j'}$ from the real-world $I_{i'j'}$.

Case 2b. Otherwise, there must be a unique, *compatible*, *isolated* user instance I_{ij}, and we *connect* $I_{i'j'}$ to I_{ij}.

We next have to show how the ideal-world adversary simulates the the *exposed data* of user instances when a user is strongly corrupted. This will be done simply by extracting the necessary information from the corresponding real-world user instance, but with the following, essential exception. Suppose $U_{i'}$ is the user being corrupted. Consider a responder user instance $I_{i'j'}$ such that

- $I_{i'j'}$ has sent the second flow in the protocol, but not yet received the third, and

- there exists a *compatible*, *isolated* user instance I_{ij} that sent the value g^x received by $I_{i'j'}$ and received the values g^y, k sent by $I_{i'j'}$.

In this case, the ideal-world adversary is given the ideal key K_{ij}, and he replaces the value of the variable k_2 in the internal state of $I_{i'j'}$ with K_{ij}.

The reader can now easily verify that the resulting ideal-world transcript is computationally indistinguishable from the real-world transcript.

Remark 24 *Protocol* **DHKE** *is insecure against strong adaptive corruptions for the same reason that it was insecure against ordinary adaptive corruptions (see §11.1). Moreover, it can also be attacked in another way. Suppose an originator instance I_{ij} is waiting for the response message from its partner-to-be $I_{i'j'}$, who has accepted a session key, and that U_i is strongly corrupted at this time. Then the real-world adversary obtains the exponent x held by I_{ij}—this cannot be erased, since I_{ij} needs this to compute the session key. Thus, the key $K_{i'j'}$, given g^y and x, will certainly* not *look like a random key.*

Remark 25 *Protocol* **EKE** *is insecure against strong adaptive corruptions for the same reason that it was insecure against ordinary adaptive corruptions (see §11.2). Also, protocol* **EKE-1** *is subject to the attack in the previous Remark. One could fix protocol* **EKE-1** *by adding an extra "key confirmation" flow like in protocol* **DHKE-1***. However, if one makes this fix, then the encryption scheme need no longer be non-malleable—ordinary semantic security suffices. Indeed, protocol* **DHKE-1** *can be seen as a special case of such a scheme using Diffie-Hellman based encryption.*

14.5 Defining secure sessions with strong adaptive corruptions

We now continue the discussion of secure session protocol started in §4, and discuss aspects of the formal security model and implementation which must be changed to accommodate strong adaptive corruptions.

In the real world model, a secure session protocol will itself have some internal, unerased data that a real-world adversary will obtain when the corresponding user is strongly corrupted. There is a fundamental limitation as to what we can expect a secure session protocol to achieve in the face of such corruptions: between the time that a message block is sent and received, the receiver must have some secret information that will allow it to decrypt the message block; therefore, if the receiver is corrupted while the message block is in transit, the adversary will learn the contents of that message block. Furthermore, when either sender or receiver are corrupted, all subsequent message blocks that are sent are also vulnerable. It would appear that we could not expect to avoid this, and accordingly, this is precisely what our definition of security guarantees.

To take the above discussion into account, we modify the ideal world attack scenario for secure sessions in §4 as follows.

The ideal world adversary initializes users and user instances, and starts sessions, as usual. The adversary makes connection assignments subject to the usual rules in §3.1.4 and §10.2. As in §4 there is no notion of a session key—that is an implementation detail. Rather, the connection assignments indicate how input/output channels are interconnected.

The sender and receiver on a secure channel shall synchronize the delivery of message blocks. To do this, there are four operations: *send ready signal*, *receive ready signal*, *send message block*, and *receive message block*. The receiver on the channel executes an alternating sequence of operations: *send ready signal*, *receive message block*, *send ready signal*, etc. Likewise, the sender on the channel executes an alternating sequence of operations: *receive ready signal*, *send message block*, *receive* *ready signal*, etc. As usual, the adversary schedules everything, but when neither of the two users involved is corrupted, the adversary is constrained as follows: corresponding ready signals and message blocks cannot be received before they are sent. This ensures that the sender never gets ahead of the receiver, so that at most one message block is *in transit*, i.e., sent but not yet received. As such messages are inherently vulnerable, it is important that the sender and receiver have explicit control over this, even in the ideal world. We shall also allow an input or output channel to be explicitly closed. A message block is no longer considered to be in transit if the message was sent, but the receiver closed its input channel. If a user instance has no partner, then by definition it cannot receive a ready signal, and so it will never send a message block, nor will it receive one.

As in §4, the adversary specifies that the message block sent is computed according to some specific function, but otherwise learns no additional information about the message block; in addition, the message block received by the receiver is equal to the one sent, thus maintaining the integrity of the channel.

That is the normal operation of a secure channel, when the session is not compromised. Consider a user instance I_{ij}. If I_{ij}'s connection assignment is initially compromise, then its session is *compromised* from the very start. Otherwise, it becomes *compromised* when either U_i or the user assigned to PID_{ij} (if any) is corrupted.

So long as I_{ij}'s session is uncompromised, everything works as described above. More precisely, if I_{ij} receives the rth ready signal, then it must have a partner, and that party sent r ready signals. If I_{ij} receives the rth message block, then it must have a partner, and that partner must have sent r message blocks, and the message block received will be equal to the message block sent.

Once I_{ij}'s session becomes compromised, all bets are off. More precisely, the adversary may make I_{ij} receive a ready signal whenever it wants. The adversary may make I_{ij} receive a message block whenever it wants, and moreover, the value of the message block is specified by the adversary, and may be chosen however the adversary wishes. When I_{ij} sends a message block, the adversary directly obtains the value of the message block.

Also, whenever a user U_i is corrupted, the values of any message blocks that are *in transit* at that moment, and are to be received by some user instances I_{ij} belonging to U_i, are given to the adversary.

That completes our sketch of the ideal world. We believe that we have given enough details so that the reader could rather unambiguously fill in the rest of the details of a complete definition of a secure session. Note that in the real world, when a user is strongly corrupted, any unerased data in higher-level protocols that is supposed to become available to the adversary upon a strong corruption can be made available through the usual mechanism of allowing the adversary to compute specific functions on the random input and message block variables.

14.6 Implementing secure sessions with strong adaptive corruptions

Let us assume that a user instance has just accepted a session key K obtained from a session key protocol.

Using a secure pseudo-random bit generator, it derives sub-keys for its input and output channel, and then *erases* the session keys. For clarity, we will describe the operation of a single uni-directional channel in terms of a *sender* and *receiver*.

Both sender and receiver have local variables *auth*, *seed*, and *pad*, which are initially derived from the session key using a pseudo-random bit generator. The value *auth* will be used as a key to a message authentication code MAC. This value is never erased or changed for the life of the session. The value *seed* will be used as input to a pseudo-random bit generator PRG. The value *pad* will be used to encrypt message blocks as a one time pad. Both the values *seed* and *pad* will be updated with each message block sent/received, effectively erasing the old values. We shall write "set $(seed, pad) = PRG(seed)$" to denote the action of applying the pseudo-random bit generator to *seed*, and *overwriting* the old values of *seed* and *pad*. Both sender and receiver have local counters r that are initially set to 0.

send ready signal Increment r, and send the message

$$\alpha = (\textsf{ready signal}, r), MAC_{auth}(\alpha).$$

receive ready signal Increment r, and receive the message

$$\alpha = (\textsf{ready signal}, r), MAC_{auth}(\alpha),$$

validating the MAC and checking the value of r received is equal to the value of local variable r.

send message block First, let X be the value of the message block to be sent, and set $Y = X \oplus pad$. Now, set $(seed, pad) = PRG(seed)$. Third, send the message

$$\beta = (\textsf{message block}, r, Y), MAC_{auth}(\beta).$$

receive message block First, receive the message

$$\beta = (\textsf{message block}, r, Y), MAC_{auth}(\beta),$$

validating the MAC and checking the value of r received is equal to the value of local variable r. Second, compute the message block $X = Y \oplus pad$. Third, set $(seed, pad) = PRG(seed)$.

That completes the description of the implementation, except to say that when one of the MACs fails, a user instance closes the channel. A user instance could also choose to unilaterally close a channel, perhaps reflecting a "time out" condition. When a channel is closed, all the internal data associated with that channel are erased.

It is not difficult to show that if the session key is established using a key exchange protocol that is secure against strong adaptive corruptions, and if we implement the channels as described here, we get a secure session protocol that is secure in the sense defined in §14.5. We do not state this as a theorem, since our definition of a secure channel is not quite formal enough to justify the use of the term "theorem"; nevertheless, once all the details in the definition of a secure channel were filled in in a reasonable way, one would indeed obtain something worthy of being called a "theorem."

We sketch how such a theorem would be proved. Suppose that a user instance wanted to send its first message block. Before it would send the encryption of this block, it awaits the first ready signal from its partner. Now if either of the two relevant users are corrupted, the simulation is trivial, since the simulator (i.e., ideal-world adversary) has the right to obtain the value of the relevant message blocks. Otherwise, if neither user is corrupted, then the definition of secure key exchange (as well as the security of the pseudo-random bit generators and the MAC) implies that the sender must have partner, and that this partner indeed sent the ready signal. Now, so long as neither sender nor receiver are corrupted, whenever a message block is sent, our simulator just generates the encryption Y as a random bit string. If the receiver becomes corrupted while a message is in transit, then the simulator computes the receiver's *pad* as $pad = X \oplus Y$, where X is the value of the actual message block, which the simulator obtains. In this way, the simulator constructs a consistent-looking internal state for the receiver's *pad* value. The reason this works is that the synchronization of sender and receiver guarantees that the receiver has erased all data that was used to compute its current value of *pad*. Therefore, the current value of *pad* is indistinguishable from a random bit string. That is the trickiest bit of the simulation, since after the corruption, the simulator's task is much easier—it obtains all subsequent message blocks generated by the sender, and does not need to respect any of the synchronization or integrity constraints.

We leave the rest of the details to the reader. The main idea of this proof essentially appears in Beaver and Haber [BH92], although their setting and the details of their solution are slightly different.

We note that our definition of security for key exchange protocols with respect to strong adaptive corruptions is actually stronger than necessary for the purpose of constructing a secure session with respect to strong adaptive corruptions. Indeed, one can show that the above implementation of a secure session, together with protocol **DHKE** in §7 is already a secure session with respect to strong adaptive corruptions.

This may seem a bit strange at first, but is really not so surprising. Our definition of security of a session key protocol defines a natural, robust, and intuitive interface, but it is a bit stronger than necessary for the particular application of building a secure session. This is analogous to the design of a software library routine interface: for a particular application, the library routine provides more functionality than necessary, and therefore, the implementation may not be the most efficient possible. In §15.6 we present an alternative definition for key exchange secure against strong adaptive corruptions. This definition is less natural and weaker than the definition presented in this section, but is just strong enough to build a secure session.

14.7 Anonymous users

One can adapt the definition of security with respect to anonymous users (see §9) to incorporate strong adaptive corruptions. There are some technical issues that need to be addressed, however.

14.7.1 Strong corruptions in the real world

Although there is no long-term secret associated with an anonymous user, *active* anonymous user instances may have unerased data that could be obtained by an adversary. In discussing strong adaptive corruptions for ordinary (non-anonymous) users in §14.1, we grouped together all user instances associated with that user, so that the adversary obtains all the unerased data associated with the user instances belonging to that user. This models the natural situation where all the user instances belonging to that user run on the same machine, and so a corruption of a user corresponds to a corruption of that machine. We do not want to group all anonymous user instances together in this way. Therefore, our real world adversary strongly corrupts *individual* anonymous user instances.

14.7.2 Strong corruptions in the ideal world

We have to modify the definition in §14.3 accommodate anonymous users.

When an anonymous user instance is corrupted, we allow the ideal-world adversary to specify a set S of *compatible*, *isolated* user instances. The adversary is given all the session keys held by the user instances in S. Moreover, the connection assignment of all the user instances in S is changed from create to compromise. This means, in particular, that no user instance may in the future *connect* to one of the user instances in the set S. None of the information about S or the keys they hold is explicitly logged in the transcript, but like connection assignments in general, we assume that the set S can be computed as a function of the transcript.

14.7.3 A secure key exchange protocol

We leave it to the reader to verify that protocol **A-DHKE-1** (see §12.1) is secure against strong adaptive corruptions in the sense we have just defined.

14.7.4 Defining secure sessions with strong adaptive corruptions

We now discuss the changes necessary to the definition of secure sessions presented in §14.5 to accommodate anonymous users.

First, the ideal-world adversary makes connection assignments just as in §14.7.2.

Second, the definition in §14.5 of when a session is *compromised* has to be modified as well:

- Suppose I_{ij} is a user instance with $PID_{ij} = \mathsf{anonymous}$. If I_{ij}'s initial connection assignment is compromise, then its session is *compromised* from the very start. Otherwise, the session becomes compromised when its connection assignment is changed from create to compromise (as in §14.7.2), or U_i is corrupted, or I_{ij}'s partner (if any) is corrupted.

- Suppose I_{0j} is an anonymous user instance. If I_{0j}'s initial connection assignment is compromise, then its session is *compromised* from the very start. Otherwise, the session becomes compromised when I_{0j} is itself corrupted, or if the user to which PID_{0j} is assigned is corrupted.[2]

Third, when an anonymous user instance is corrupted, any message block that is *in transit* to this user instance at that moment is given to the ideal-world adversary.

Otherwise, everything works just as in §14.5.

It may be worthwhile to spell out some of the implications of this definition. The implication for a user instance I_{ij} with $PID_{ij} = \mathsf{anonymous}$ is as follows. Suppose it has just started its session and has received a ready signal or message block. Then either

- the session is already compromised, and all bets are off, or

- the session is not compromised, I_{ij} has a partner, the ready signal or message block came from that partner, and the usual guarantees for the session will be in force so long as neither U_i nor I_{ij}'s partner are corrupted.

In the first case, when the session is already compromised, then either

- U_i has been corrupted,

- I_{ij} has a partner who has been corrupted, or

- I_{ij} has no partner and never will.

[2] If $PID_{0j} = \mathsf{anonymous}$, then the session would also be considered compromised if I_{0j}'s connection assignment is changed from create to compromise; however, as pointed out in Remark 12, this is not an interesting situation to consider.

The implication for an anonymous user instance I_{0j} is as follows (we assume $PID_{0j} \neq$ anonymous). Suppose it has just started its session and has received a ready signal or message block. Then either

- the session is already corrupted, and all bets are off, or
- the session is not compromised, I_{0j} has a partner, the ready signal or message block came from that partner, and the usual guarantees for the session will be in force so long as neither I_{0j} nor the user to which PID_{0j} is assigned is corrupted.

In the first case, when the session is already compromised, then either

- I_{0j} has been corrupted, or
- the user to which PID_{0j} is assigned is corrupted.

14.7.5 Implementing secure sessions with strong adaptive corruptions

Nothing changes here at all. The implementation in §14.6 can be used without change. As in §14.6, we mention that protocol **A-DHKE** in §9.2 is actually sufficient.

15 Comparison with the Bellare-Rogaway Model

Bellare and Rogaway [BR95] have presented a formal model for secure key exchange protocols. Technically speaking, their model applies only to the on-line TTP setting. However, it is relatively straightforward to adapt this model to the off-line TTP setting. This program has been carried out by Blake-Wilson, *et al.* [BJM97, BM97]. For lack of a better name, let us call this the *BR model.*

We want to compare our model of security, which we might call the *simulation model*, to the BR model.

15.1 The BR model

Instead of recalling all the notation of [BR95, BJM97, BM97], we show what the BR model essentially is in in terms of our notation.

In the BR model, the attack scenario is the same as the "real system" in the simulation model, except that the *application* operations are restricted to be of a special type which we describe below. Their model allows strong adaptive corruptions, but of course, one could consider restricted adversaries that make only static corruptions or (ordinary) adaptive corruptions.

The definition of security in the BR model consists of three parts. The first two are *termination* and *liveness*, which are exactly the same as in the simulation model. For lack of a better name, we call the third part the *BR security property.*

Although there is no notion of an "ideal system" in the BR model, there is a notion of a connection assignment. To establish the BR security property of a particular key exchange protocol, one must exhibit a *connection assignment function.* Connection assignment functions in the BR model are the same as in the simulation model; in particular, they are subject to the usual rules in §3.1.4 and §10.2; however, there are some additional, technical restrictions. Namely, the connection assignment function must be *universal*, i.e., there is one that works for all adversaries, and furthermore, it must be *application independent*, a technical restriction that we describe in the next paragraph.

An *application independent* connection assignment function is one which can be computed as a function of the partial transcript obtained from the full transcript by deleting all records pertaining to *application* operations.

This is a natural restriction, and it does not affect the analysis of any protocols that we know of. This restriction will be needed in the proof of Theorems 12 and 13, which relate security in the BR model with security in the simulation model.

Just to be explicit, we are assuming here the default, i.e., liberal, compromise rule (§10.3).

The BR security property is the following: there exists a connection assignment function such that for all adversaries A, the advantage that A has in the following game is negligible.

The game played by A runs as follows. The adversary executes any of the usual commands except that the *application* commands are one of two types, which we call *reveal* and *test*. In a *reveal* operation, the adversary obtains any session key of its choice belonging to a user instance whose connection assignment is create or compromise. The adversary may execute any number of *reveal* operations. In a *test* operation, the adversary specifies a user instance whose connection assignment is create; at this point, a coin is flipped, and the adversary is either given the user instance's session key or a random string, depending on the outcome of the coin flip (which is not in the adversary's view). The *test* operation may only be executed once.

There is an additional restriction on the *test* operation, which we shall call the *test restriction*:

> if I_{ij} is subject to a *test* operation, then at no time before or after the *test* operation may the adversary corrupt U_i or the user (if any) to which PID_{ij} is assigned.

The adversary's *advantage* is defined to be the maximum of

- the distance from 1/2 of the probability of guessing the outcome of the coin toss in the *test* operation, and
- the probability that two user instances that are partners (as determined by the connection assignment function) do not share the same session key.

Remark 26 *Note that since the adversary already wins the game if he can make two partners accept different session keys, there is no need to allow* reveal *or* test *operations to be applied to user instances that* connect *to other user instances.*

15.2 Correcting a flaw in the original BR model

There are a few small, technical differences between our presentation here and in [BR95, BJM97, BM97] that are not so important. However, we have taken the opportunity here to correct a serious flaw that appears in [BR95, BJM97, BM97] that was pointed out to the authors of [BR95] by Charles Rackoff. In the formulation in [BR95, BJM97, BM97], the *test* operation is only allowed to be performed *at the very* end of the adversary's execution, whereas we have allowed it to occur at any time. This is important, because without this, the definition does not detect "protocol interference" as was discussed in §2.

We can illustrate this point with an example derived from one suggested by Charles Rackoff. Consider the following modification of protocol **DHKE-1** (see §12.1), which we call protocol **DHKE-1′**. This protocol works just like **DHKE-1**, except as follows. Suppose a user instance belonging to $U_{i'}$ in the role of responder is waiting for the third flow of the protocol, which consists of the confirmation key k_1; if instead of k_1 it receives a message of the form $(\textsf{core dump}, \mathit{BitGen}(k_2))$, then it terminates the protocol with a status of reject, and generates a final outgoing message consisting of k_2. Recall that k_2 is (what would have been) the session key, and that *BitGen* is assumed to be a secure pseudo-random bit generator. We assume here that k_2 has the right length so that it may be used as an input to *BitGen*.

We hope the reader would agree that protocol **DHKE-1$'$** should be considered insecure under any reasonable definition of security. Indeed, if user a user instance I_{ij} in the role of originator establishes a session key K, and just happens to output $BitGen(K)$ in a higher-level protocol before its last message is delivered to its would-be partner $I_{i'j'}$, the adversary can cause $I_{i'j'}$ to "core dump," handing K to the adversary on a silver platter.

If the *test* operation is allowed in the middle of the game, it is easy to see that an adversary can obtain significant advantage, and so this protocol is not secure under our definition. However, if the *test* operation is allowed only at the end of the game, the adversary has only a negligible advantage (since the adversary could not hope to compute $BitGen(k_2)$ on its own), and the protocol would be secure under that definition of security.

Admittedly, this example is a bit contrived, but nevertheless illustrates the point. Another, perhaps more convincing reason for allowing the *test* operation to occur at any time is to get an equivalence theorem between security in the BR model and security in the simulation model (see Theorem 12 below), which suggests that this is a robust notion of security.

This flaw in [BR95] illustrates the danger of making a technical, low-level definition without carefully exploring its relationship with a more natural, higher-level notion of security.

15.3 The equivalence of strong adaptive and static corruptions in the BR model

In the BR model, one could distinguish between security against static, adaptive, and strong adaptive corruptions. However, it turns out that these notions are equivalent, provided we make an additional, quite natural restriction on the connection assignment function which we call *local computability.*

Intuitively speaking, a *locally computable* connection assignment function is one that determines the connection assignment for a particular user instance I_{ij} using only those parts of the transcript that might have something to do with user U_i or PID_{ij}. More precisely, this means that the connection assignment should be computable by a function applied to the subsequence of records in the transcript obtained by deleting records corresponding to these operations:

- all *application* operations,
- all *initialize user*, *initialize user instance*, *deliver message*, and *corrupt user* operations which refer to a user other than U_i or the user (if any) to which PID_{ij} is assigned, and
- all *register* operations which refer to an identity other than ID_i and PID_{ij},

Note that in the on-line TTP setting, we do not delete any of the *deliver message to TTP* operations.

Local computability is a natural restriction, and we know of no protocols whose security analysis is affected by making this restriction.

In the following theorem, we assume that connection assignment functions are *universal* and *locally computable.* Perhaps the same theorem could be proven using a more clever argument without requiring *local computability.*

Theorem 11 *In the BR model, security against static, adaptive, and strong adaptive corruptions are equivalent.*

It is clear that in the BR model, security against strong adaptive corruptions implies security against adaptive corruptions, and that security against adaptive corruptions implies security against static corruptions.

Now, to show that security against static corruptions implies security against strong adaptive corruptions. Suppose a key exchange protocol is secure against static corruptions. This implies the existence of a connection assignment function. However, one technical point we have to deal with is that this connection assignment function is not defined for transcripts containing *corrupt user* operations. We therefore need to extend the domain of definition of the given connection assignment function, which is easy to do by exploiting the local computability property of the given connection assignment function. To calculate the connection assignment for a user instance I_{ij}, if either U_i is corrupted, or PID_{ij} is assigned to a corrupted user, then we *compromise* I_{ij}. Otherwise, we compute the connection assignment using the given connection assignment function, using only the relevant local information in the transcript, which does not contain any *corrupt user* operations.

Now consider an an adversary A that makes strong adaptive corruptions, and suppose that A has non-negligible advantage in the game defining the BR security property. Here is how we can convert A into an adversary A' that makes only static corruptions, and that has a smaller, but still non-negligible advantage, using a standard "plug and pray" argument. At the beginning of the game, A' randomly chooses two players U_i and $U_{i'}$. A' never actually initializes any users other than U_i or $U_{i'}$, and A' never corrupts any users. All users besides U_i and $U_{i'}$ that A might initialize and perhaps corrupt are simply under the direct control of A', and these are never initialized as users. Instead, A' simulates the view of A, and does whatever A does. We pray that A does not corrupt either U_i or $U_{i'}$, and that A chooses to perform his *test* operation on an instance I_{ij} with $PID_{ij} = ID_{i'}$. Our prayers will be answered with non-negligible probability, and if they are not, we simply stop the game.

It is easy to see that if A has a non-negligible advantage, then so will A'. The theorem now follows.

15.4 Relation between the BR model and the simulation model

Now, we want to formulate and prove that security in the BR model and security against static corruptions in the simulation model are equivalent. To do this, we have to restrict the way the ideal-world adversary A^* in the simulation model computes connection assignments to the way connection assignments are computed in the BR model. This means that the connection assignment function must be *universal* and *application independent*, as described in §15.1. *For the remainder of this section, this restriction on the adversary A^* in the ideal world model is implicitly in force.*

Theorem 12 *Security against static corruptions in the simulation model is equivalent to security against static corruptions in the BR model.*

To prove that security against static corruptions in the simulation model implies security in the BR model, one only need observe that the game defining the BR security property is just a particular game that can be easily represented in the simulation model.

Now, to prove that security in the BR model implies security against static corruptions in the simulation model. Assume a given protocol is secure in the BR model, so there exists an appropriate connection assignment function.

Let A be a real-world adversary in the simulation model. We construct the corresponding ideal-world adversary A^* as follows. Generally, A^* does whatever A does. Whenever a user instance accepts, A^* makes the connection assignment using the connection assignment function mentioned in the previous paragraph. For a compromise connection assignment, A^* extracts the user instance's actual session key to obtain the session key required for the *start session* operation in the ideal world game. Otherwise, for any other connection assignment, the ring master chooses the keys according to the rules in the ideal world game. Of course, *application* operations are evaluated by the ring master using the idealized session keys.

Now to show indistinguishability of ideal-world and real-world transcripts. If there were a good statistical test, then we could easily apply a hybrid argument to construct an adversary with significant advantage in the BR game, using *reveal* operations as necessary, and using a single *test* operation to distinguish two adjacent hybrid distributions where there is significant gap in the expectation of the statistical test's output. The details of this are straightforward, but bear in mind that it is the *ring master* in a hybrid ideal world/real world game who uses the operations *reveal* and *test* to generate some keys, and generates other keys as random bit strings. The ideal world *adversary* only has indirect access to these keys through *application* operations, except that he performs *reveal* operations on user instances that are *compromised*—he needs to get these keys from somewhere, since they need to be specified by the ideal world adversary during the *start session* operations for user instances that are *compromised*.

The completes the proof of the theorem.

Notice that in making the above hybrid argument, we needed the ability to perform the *test* operation at an arbitrary point.

Theorem 12, together with Theorem 11, imply that security against static corruptions in the simulation model is equivalent to security against strong adaptive corruptions in the BR model, provided we restrict to *universal*, *locally computable* connection assignment functions.

15.5 Forward security in the BR model

It is instructive to see where the proof of Theorem 12 breaks down in the face of adaptive corruptions. It is in the hybrid argument. We need to be able to "plant" a *test* operation at an appropriate place in the execution, where the statistical test will notice a difference. But it may very well be the case that all the places that would be useful are "off limits" due to the rules of the BR game. This is because the *test restriction* in the game defining the BR security property prohibits a *test* operation on anybody who is *at any time* corrupted or is partnered with someone who is. If we think back to the discussion of protocol **DHKE** in §11.1, we can see that the only useful keys to *test* are off limits due to this restriction.

But this observation also tells us how to strengthen the BR model. We call this *forward security against adaptive corruptions* in the BR model. The only change is that we drop the *test restriction*. Notice the difference: instead of a "blanket" ban on *test* operations, we instead rely on the much more "precise" *compromise* connection assignment to selectively prevent *test* operations where we do not want them—namely, after a corruption. This clearly captures the notion of *forward security*.

Theorem 13 *Security against adaptive corruptions in the simulation model is equivalent to forward security against adaptive corruptions in the BR model.*

The proof is straightforward. We omit the details. Note that the theorem holds using either the liberal or conservative compromise rule (§10.3).

15.6 An alternative definition of security against strong adaptive corruptions

We know of no definition in the "BR style" that is equivalent to the notion of security against strong adaptive corruptions as we have defined it in the simulation model (see §14). Moreover, as we have seen in §14.6, our definition of security in §14, while natural, was stronger than necessary for the purpose of building a secure session protocol. It turns out that we can easily modify the definition of security in the BR model to obtain a definition of security against strong adaptive corruptions that is weaker and much less natural than that in §14, but is just strong enough to build a secure session protocol.

This alternative definition of security is a modification of the definition of security in the BR model in §15.1. As in that section, we make use of connection assignment functions that are *universal* and *application independent*, and we use the default, i.e., liberal, compromise rule. All corruptions are strong adaptive corruptions.

Here are the changes we need to make:

(1) We allow the connection assignment of a user instance to change from create to $\mathsf{compromise}$, as follows. If a user instance I_{ij} has a connection assignment of create and is still *isolated* at the point in time in which either user U_i or the user (if any) to which PID_{ij} is assigned is strongly corrupted, then I_{ij}'s connection assignment is changed to $\mathsf{compromise}$.

(2) As in §15.5, we drop the *test restriction*. However, if the *test* operation is applied to I_{ij}, the adversary may not perform a corruption that would cause I_{ij}'s connection assignment to become $\mathsf{compromise}$.

(3) A user instance whose connection assignment is changed to $\mathsf{compromise}$ is no longer considered *isolated*, and no user may connect to it.

Discussion

Note that without loss of generality, we can assume that a connection assignment function always *compromises* a user instance whenever that is legal.

We leave it to the reader to verify the following:

- protocol **DHKE** in §7 is secure against strong adaptive corruptions under this alternative definition, and
- any protocol that is secure under this alternative definition, together with the implementation of a secure session in §14.6, yields a secure session protocol that is secure in the sense of §14.5.

16 Comparison with the Model of Bellare, Canetti, and Krawczyk

Bellare, Canetti, and Krawczyk [BCK98] define security against an adversary very similar to ours: he has complete control of the network, and can make strong adaptive corruptions. In fact, this is the only corruption mode they consider. Their definition of security is also based on simulation, but there are several differences in both detail and substance between our definition and theirs.

One issue alluded to in §1 about the model in [BCK98] is its treatment of the *ordinary use* of session keys, e.g., as encryption or authentication keys. There is no analogue of our notion of an *application* operation in their model. Rather, in their model, when a session key is established, its value is silently written to a transcript which may be input to a statistical test, but only *after* the adversary has completed its attack. However, the value of the key, or any values derived from it, are not generally available to the adversary while the attack is ongoing. An exception to this is that the adversary may execute an explicit *corrupt session* operation, whereby the adversary obtains the session key itself.

It is not at all clear what this *corrupt session* operation is supposed to represent. In order to be able to properly model ordinary key usage and protocol interference attacks, it would seem we must assume that *all* sessions are corrupted—or at least those whose session keys are actually used. At the very least, then, the term "corrupt session" then has a somewhat misleading connotation; indeed, maybe a better name would be "use session key." Moreover, this definition of security suffers from a more serious problem. Because any use of a session key essentially implies that the ideal-world adversary has the session key itself, it would seem that any key that is *ever used* is potentially *completely vulnerable*. Indeed, what is to keep a key that is available to the ideal-world adversary from popping up, say, in a protocol message flow? The ideal-world adversary can simulate this, since it knows the key. As a concrete example, protocol **DHKE-1′** in §15.2 is secure under the definition of security in [BCK98]. To see this, note that a user instance $I_{i'j'}$ will not "core dump," except with negligible probability, unless the corresponding session has been corrupted. But if the session is corrupted, the ideal-world adversary has the session key and therefore can easily simulate the "core dump." However, as we have already argued, protocol **DHKE-1′** should not be considered secure under any reasonable definition of security.

Another aspect of [BCK98] is how it models strong adaptive corruptions. According to their definition, when a user is corrupted in the ideal world, "the effect is that all the keys known to [that user] become known to the adversary." It is a bit hard to understand the motivation for this. First, as we have pointed out above, any session key that is actually used must already be available to the adversary via a *corrupt session* operation, and so many of the keys given to the ideal-world adversary upon the corruption of a user are redundant. Second, it would appear that the intention of this definition is to avoid any guarantee of forward security—this point is not clear in the paper, but the authors indicate that forward security is an issue to be tackled in a subsequent (and as yet to appear) version of the paper, and so it seems safe to assume that their definition is not meant to imply forward security. But as we have already remarked in §10.5, we cannot build a secure session protocol on top of a key exchange protocol that does not guarantee forward security. Our opinion is that it makes little sense to define security for a session key protocol that cannot be used as a building block for secure sessions. Indeed, one of the underlying technical themes of our work is that forward security and simulatability with respect to adaptive corruptions are in many ways two sides of the same coin.

Another aspect of the way corruptions are modeled in [BCK98] is the way in which the inherently vulnerable keys (see §14.2) are made available to the ideal-world adversary. This issue is not explicitly addressed in [BCK98], and it is not immediately clear that these keys are available to the adversary. At one point, it is stated in a parenthetical, and seemingly motivational remark, that "we envision that the value [of a session key] is handed to [a user] by the trusted party." It is only by interpreting this remark to have a specific descriptive meaning, rather than being purely motivational, that one avoids an unsatisfiable definition of security.

Also as mentioned in §1, the two protocols presented and analyzed in [BCK98] in the "authenticated links" model are actually insecure, under their definition of security and under any reasonable simulation-based definition of security. One of the protocols is a two-pass Diffie-Hellman protocol that is insecure for the very same reasons that our protocol **DHKE** is insecure against strong adaptive corruptions (see §11.1 and Remark 24). However, we should point out that if the authenticated links are implemented using one of the techniques described in [BCK98], the resulting protocol in the "raw" (i.e., unauthenticated links) model apparently happens to be secure. The other protocol is based on public key encryption, and is insecure for the very same reasons that our protocol **EKE** is insecure against strong adaptive corruptions (see §11.2 and Remark 25).

17 Conclusion

The methodology of modern theoretical cryptography is maturing to a point where it can take on tasks that have traditionally belonged to the domain of "security engineering." This seems useful, as many security problems are often viewed as "implementation errors" which we believe could be more fruitfully viewed as cryptographic design errors. Probably the main reason for these differing points of view is simply that the high-level cryptographic designers and implementors have usually been more or less disjoint sets of people.

The activity of designing a formal security model for something as complicated as a key exchange protocol or a secure session protocol is similar in many ways to that of designing a software interface. In designing a software interface, there is no "right" or "wrong." One wants an interface that will be easy to understand and to use in an intuitive way. One also wants an interface that can be effectively implemented. Because of the richness of the environment in which session key protocols are used, there will always be room for debate on many of the details of such a security model. We hope that this paper has at least served to make explicit most of the important choices that one encounters in designing such a security model, even if the reader disagrees with some of the particular choices that we have made.

Having a formal security model and a "provably secure" protocol in that model is no panacea. Indeed, it is possible that the model is flawed somehow; in particular, it may not be rich enough to express a particular type of realistic attack. And of course, the proofs may contain errors or the underlying intractability assumptions could turn out to be false. Nevertheless, the activity of designing such models and analyzing protocols in these models is a worthwhile activity: only by doing so can we hope to increase our understanding of the protocols we use in practice, and to design better protocols (or even have a meaningful way to measure "better"). It is also an ongoing activity: when weaknesses in the model or errors in the proofs are uncovered, then these must be repaired.

Although a healthy amount of skepticism is always appropriate, an irrational rejection of the entire approach of formal modeling and proofs as nothing but "snake oil" does not seem helpful. Such an anti-intellectual attitude is unfortunately not so rare in the security research community. It only serves to retard meaningful scientific progress, and to perpetuate the mystique surrounding the self-proclaimed "high priests" of computer security.

Acknowledgments

Thanks to Ran Canetti for several enlightening discussions on the topic of session key exchange; in particular, for conversations dealing with anonymous users, and also for pointing out some places in a previous version of this paper where some aspects of [BCK98] were mistakenly mischaracterized. Thanks to Charles Rackoff for our annual discussions on key exchange and secure sessions. Also thanks to Christian Cachin and Michael Waidner for their comments on an earlier version of this paper.

References

[BAN90] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Compter Systems*, 8:18–36, 1990.

[BCK98] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *30th Annual ACM Symposium on Theory of Computing*, 1998.

[Bea91] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4:75–122, 1991.

[BGH+91] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutton, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In *Advances in Cryptology–Crypto '91*, pages 44–61, 1991.

[BH92] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology–Eurocrypt '92*, pages 307–323, 1992.

[BJM97] S. Blake-Wilson, D. Johnson, and A. Menesez. Key agreement protocols and their security analysis. In *Sixth IMA International Conference on Cryptography and Coding*, 1997. On line version: `cacr.math.uwaterloo.ca/~sblakewi`.

[BM92] S. Bellovin and M. Merrit. Encrypted key exchange: password-based protocols secure against disctionary attacks. In *Proc. 1992 IEEE Computer Society Conf. on Research in Security and Privacy*, pages 72–74, 1992.

[BM97] S. Blake-Wilson and A. Menesez. Entity authentication and key transport protocols employing asymmetric techniques. In *Security Protocols Workshop*, 1997. On line version: `cacr.math.uwaterloo.ca/~sblakewi`.

[Bon98] D. Boneh. The Decision Diffie-Hellman Problem. In *Ants-III*, pages 48–63, 1998. Springer LNCS 1423.

[Boy99] M. Boyarsky. Public-key cryptography and password protocols: the multi-user case. In *6th ACM Conf. on Computer and Communication Security*, 1999.

[BR93a] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology—Crypto '93*, pages 232–233, 1993.

[BR93b] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR94] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—Crypto '94*, pages 92–111, 1994.

[BR95] M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In *27th Annual ACM Symposium on Theory of Computing*, pages 57–66, 1995.

[BR96] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Advances in Cryptology–Eurocrypt '96*, pages 399–416, 1996.

[Bra93] S. Brands. An efficient off-line electronic cash system based on the representation problem, 1993. CWI Technical Report, CS-R9323.

[Can95] R. Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, The Weizmann Institute of Science, Dept. of Computer Science and Applied Mathematics, 1995.

[CD96] R. Cramer and I. Damgard. New generation of secure and practical RSA-based signatures. In *Advances in Cryptology–Crypto '96*, pages 173–185, 1996.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology–Crypto '98*, pages 13–25, 1998.

[CS99] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *6th ACM Conf. on Computer and Communications Security*, 1999.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22:644–654, 1976.

[DN94] C. Dwork and M. Naor. An efficient existentially unforegeable signature scheme and its applications. In *Advances in Cryptology–Crypto '94*, pages 218–238, 1994.

[DS81] D. Denning and G. Sacco. Timestamps in key distribution protocols. *CACM*, 24(8):533–536, 1981.

[DvOW92] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and authenticated key exchange. *Designs, Code, and Cryptography*, 2:107–125, 1992.

[FO99] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology–Crypto '99*, pages 537–554, 1999.

[GHR99] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology–Eurocrypt '99*, pages 123–139, 1999.

[GLNS93] L. Gong, M. Lomas, R. Needham, and J. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE J. on Selected Areas in Communications*, 11(5):648–656, 1993.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17:281–308, 1988.

[HK98] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. In *5th ACM Conf. Computer and Communications Security*, pages 122–131, 1998.

[Lub96] M. Luby. *Pseudorandomness and Cryptographic Applications.* Princeton University Press, 1996.

[MvOV97] A. Menesez, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[NR97] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, 1997.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21:993–999, 1978.

[NY90] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, 1990.

[PS96] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology–Eurocrypt '96*, pages 387–398, 1996.

[RS91] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology–Crypto '91*, pages 433–444, 1991.

[Sch91] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4:161–174, 1991.

[Sho99] V. Shoup. On formal models for secure key exchange. IBM Research Report RZ 3120, April 1999.

[SNS88] J. Steiner, C. Newman, and J. Schiller. Kerberos: an authentication service for open network systems. In *Proc. USENIX Winter Conf.*, pages 191–202, 1988.

[Sta96] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology–Eurocrypt '96*, pages 190–199, 1996.