RZ 3120 (#93166) 19/04/1999
Computer Science/Mathematics 33 pages

Research Report

On Formal Models for Secure Key Exchange

Victor Shoup

IBM Research Division
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

On Formal Models for Secure Key Exchange

Victor Shoup

IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

A new formal security model for session key exchange protocols in the public key setting is proposed, and several efficient protocols are analyzed in this model. The relationship between this new model and previously proposed models is explored, and several interesting, subtle distinctions between static and adaptive adversaries are explored. We also give a brief account of anonymous users.

1 Introduction

In this paper, we investigate formal models of security for authenticated key exchange protocols in a public key setting where the only trusted party is an off-line *certification authority*.

Our work follows up on that of Bellare, Canetti, and Krawczyk [2], which is grounded in the multi-party simulatability tradition (see, e.g., [1]). This approach seems very attractive, because it formulates security in terms of the *service* a session key protocol should provide to a higher level protocol, rather than getting mired in the details of session key protocols themselves, many of which are irrelevant. Moreover, security in this model implies security against a whole range of specific attacks—even attacks yet to be discovered.

We correct several serious flaws in both the formal model and protocols presented in [2], and also extend their work in several ways. In particular, we give for the first time a reasonable formal treatment of the notion of *forward security*, and we take into explicit account the role of the certification authority. We also shed light on other aspects of the problem, including subtle differences between security against static adversaries and adaptive adversaries, and connections between this security model, and the previously proposed security model of Bellare and Rogaway [6] (see also [8, 9]).

We also give a brief account of anonymous users. In many situations, one of the two users in a key exchange protocol may not have a certificate. This can happen, for example, in SSL and SSH. Typically, the anonymous user can authenticate himself to the other with the session itself by means of a password. We show how our formal model can be easily adapted to deal with this situation.

In addition to definitions, we give several examples of key exchange protocols and proofs of there security. Additionally, these examples will highlight some of the subtle differences between definitions of security.

1.1 Previous Work

There is a vast literature on this subject.

The classic paper of Bird, *et al* [7], 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 symmetric key cryptography setting.

Subsequently, Bellare and Rogaway [4, 6] proposed a formal model of security for authenticated key exchange protocols, again in the symmetric key cryptography setting, where there is an on-line key distribution center. Their model captured the types of attacks discussed in [7]. Their formal models were subsequently adapted to the public key setting by Blake-Wilson, *et al* [8, 9]. The definitions of security here seem fairly compelling, but yet, they seem a bit "ad hoc" and it is not clear what implications these definitions have for higher-level protocols that use the session keys.

The "station-to-station" protocol was introduced in the classic paper of Diffie, *et al* [13]. This paper presents an authenticated key exchange protocol based on the basic, unauthenticated Diffie-Hellman key exchange protocol. The authors do a rather informal security analysis, and point out numerous pitfalls and attacks one should worry about.

We refer the reader to the above papers for more extensive bibliographies.

More recently, Bellare, Canetti, and Krawczyk [2] have proposed a quite different approach to formal security models for key exchange. This approach is similar to the 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. This proof involves showing the existence of a simulator.

As mentioned above, this approach to analyzing protocol security is very attractive. Unfortunately, as also mentioned above, there are serious deficiencies in [2] which we address here.

First, many points in the formal model in [2] are either too vague or too weak. For example, no account is given of how the ordinary *use* of a session key in a higher-level protocol is to be modeled, or of the role of the certificate authority. Because of this, their model fails to model "protocol interference" and "PKI" attacks, notions we discuss below in §2.

Second, both of the proposed session key exchange protocols in that paper are not even secure in the sense that the authors intended; or at least, it is clear that they cannot be proven secure in this sense using known techniques.

Third, [2] uses an idealized "authenticated links" model of communication to describe their protocols, and then shows how to translate these into "real world" protocols. While this approach has certain advantages, it also eliminates from consideration several interesting examples of secure "real world" protocols that do not have counterparts in the "authenticated links" model.

Fourth, [2] only considers one type of attack, wherein the adversary obtains all unerased state information held by a player at the time of corruption. There are other very interesting, and possibly more relevant, types of attacks that one should consider. Here, we distinguish between three types of corruptions:

(1) static corruptions,

(2) dynamic corruptions where only long-term secrets are exposed, and

(3) dynamic corruptions where all unerased state information is exposed as well.

Making these distinctions seems to be worthwhile, and leads to some interesting examples of protocols that are secure in some corruption models, but not in others. Perhaps more importantly, by considering type (2) corruptions, one can properly formulate the notion of *forward security* in the context of key exchange. Also, these different grades of corruptions shed some light on the Bellare-Rogaway (BR) security model. We show that BR security (with one essential "fix") is equivalent to our notion of simulation security against type (1) corruptions, even though the BR attack model superficially allows *adaptive* adversaries. We also show that when the BR model is strengthened in a natural way so as to guarantee forward security, we obtain our notion of simulation security against type (2) corruptions. A further distinction between [2] and this paper is that we give a much more fine-grained account of type (3) corruptions: the adversary corrupts individual user instances, and the damage is limited to these user instances. At the moment, our experience suggests that simulation security against type (3) corruptions cannot be easily captured in the style of the BR security model.

1.2 Outline of 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 present and prove the security of a Diffie-Hellman based key exchange protocol. In §5, we present and prove the security of a public key encryption based key exchange protocol. In §6, we show how to extend our formal model to model adversaries who adaptively corrupt users, basing their decision on information collected so far. These corruptions allow the adversary to obtain a user's long-term secret key, but nothing else. In §7, we consider the Diffie-Hellman and encryption protocols previously proved secure against static adversaries, and argue why we believe there is no chance to prove that they are secure against adaptive adversaries. In §8, we show how to modify the Diffie-Hellman protocol to attain security against adaptive adversaries. In §9, we show how to modify the encryption protocol to attain security against adaptive adversaries. In §10, we compare our models of security with the formal model developed by Bellare and Rogaway [6]. In §11 we consider more malicious adversaries who not only can obtain a users long-term secret, but can invade a user's computer and read the contents of its memory. In §12 we consider anonymous users. We make some final remarks in §13.

2 Protocol Interference and PKI attacks

To motivate certain aspects 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 [7].

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 station-to-station protocol.

The basic station-to-station 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 [13], 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 who "thinks" he is talking to B.

One criticism of this protocol 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 into computing on encryption $E_K(msg)$ for a message 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 certification authority to bind A's public key to $\tilde{A}$'s identity. Now, $\tilde{A}$ may have all the relevant documents to prove that he is indeed $\tilde{A}$, and suppose that this is satisfies the certification authority. If certificates are also exchanged as part of the protocol, then the adversary can replace A's certificate with $\tilde{A}$'s.

The reader might object: do not certification 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 certification 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, 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.

3 Formal Security Model—The Static Adversary Case

We present our formal notion of security, beginning with *static adversaries*, i.e., adversaries that make their decision as to whom to corrupt independently of the network traffic. In this case, such statically corrupted players do not have to 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.

We define security via simulation: we define an *ideal world model* that describes the *service* that a session key protocol should provide; then we define a reasonable *real world model* that describes what adversaries can and cannot do in real life. The definition of security intuitively says that for a secure protocol, whatever the adversary can do in the real world, he can already do in the ideal world—but by definition, an adversary cannot cause any trouble in the ideal world.

3.1 The Ideal System

In this section, we do not discuss session key protocols at all. Instead, we try to model the *service* that a session key protocol should provide to higher-level protocols: to allow joint, random session keys to be established, which can then be used in arbitrary higher-level protocols. There are absolutely no restrictions placed on these higher-level protocols: they might encrypt and authenticate messages in a session using very good cryptographic algorithms or very bad algorithms, and they even might—if they so choose—divulge the session key at the end of the session. But whatever these higher-level protocols do, they will not disrupt or interfere with the session key generation service itself.

Although the details of specifying this ideal model may seem somewhat legalistic, we believe they are fairly natural and intuitive. Moreover, we believe the model is useful in that it should allow one to reason in a fairly straightforward way about the behavior of application protocols using session keys.

Now the details.

We have a set of (honest) *users*, indexed $i = 1, 2, \ldots$. Each user i may have several *instances* $j = 1, 2, \ldots$. Thus, a pair of indices (i, j) completely specifies a particular instance of a particular user. We refer to (i, j) as a *user instance.*

Remark 1 *If you wish, you may 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. In this game, the adversary can perform one of four types of operations: *initialize user*, *initialize user instance*, *terminate user instance*, *start session*, *application*, and *implementation*. We explain in turn how each of these operations work.

3.1.1 Initialize user

In an *initialize user* operation, the adversary assigns an identity ID_i to a user 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

In an *initialize user instance* operation, a user instance (i, j) is specified, along with a value $role(i, j) \in \{\texttt{"open"}, \texttt{"connect"}\}$, as well as an identity PID_{ij}. User i must have been previously initialized, but (i, j) should not have been previously initialized.

Remark 2 *Intuitively, PID_{ij} represents the identity of the user that (i, j) wants to talk to. The value $role(i, j)$ identifies which of two roles the user instance is to have in establishing a connection. Inevitably, even in an ideal world, a connection will not be established simultaneously, but rather, one user instance will first "open" one side of the connection, and another user instance will later "connect" to it, completing the connection.*

3.1.3 Terminate user instance

In a *terminate user instance* operation, a user instance (i, j) is specified that has been previously initialized.

3.1.4 Start session

In a *start session* operation, the adversary specifies a user instance (i, j) that has been initialized but not terminated.

Now, the adversary also has to specify how the session key K_{ij} for (i, j) is generated, by specifying one of three types of *connection assignments*: *open*, *connect to* (i', j'), or *expose*. There are rules governing the legality of these assignments, and the session key K_{ij} is determined by the connection assignment.

open. This operation is allowed only if $role(i, j) = \texttt{"open"}$. The ring master generates K_{ij} at random. In this case we say that (i, j) *is open for connection.*

connect to (i', j')**.** This operation is allowed only if (1) $role(i, j) = \texttt{"connect"}$, (2) $PID_{ij} = ID_{i'}$ for some initialized user i', (3) $PID_{i'j'} = ID_i$, and (4) (i', j') is was opened for connection after (i, j) was initialized and is still open for connection. The ring master sets $K_{ij} = K_{i'j'}$. Also, after this point we say that (i', j') *is no longer open for connection.*

[1]Following circus terminology.

expose. This operation is allowed only if PID_{ij} has not been assigned to a user. The ring master sets $K_{ij} = K$, where K is specified by the adversary.

Note. Session keys are bit strings all of some agreed upon length.

Remark 3 *Note that a user instance who is opened for connection has no guarantee that anyone will eventually connect to it, but it can be sure that if anyone does connect to it, it will do so at a later time. Also, note that a user instance who connects to another user instance is guaranteed that it was opened for connection at some time in the past, but no earlier than when the connecting instance was initialized. Thus, the user instances in the two roles have asymmetric guarantees as to the existence and liveness of the connection. This asymmetry seems inevitable.*

We make a restriction on how the adversary computes connection assignments. Namely, this should be strictly a function of the transcript of the adversary's interactions (see §3.1.7 below) up to the current *start session* operation. In particular, the connection assignment should not depend on hidden random coins of the adversary. One reason for this will be discussed in §3.3.1.

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 *environment*.

In an *application* operation, the adversary specifies a function f on the environment and the set of session keys that have been defined so far. The function f must be an efficiently computable function that is described in some agreed upon notation (a polynomial sized circuit, for example). We will at times identify f with its description.

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 environment can be used to generate the message. Also, the encryption algorithm itself may use further random bits that also come from the environment. The bit string actually output by this user instance can be easily expressed by an appropriate function f of R and $\{K_{ij}\}$.*

Remark 5 *Having one global bit string R (instead of one bit string per user instance) 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 "do nothing" operation that otherwise has no effect on the game, except that the adversary simply makes a comment *cmnt*, 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.

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

initialize user	("`initialize user`", i, ID_i)
initialize user instance	("`initialize user instance`", $i, j, role(i,j), PID_{ij}$)
terminate user instance	("`terminate user instance`", i, j)
start session	("`start session`", i, j)
application	("`application`", $f, f(R, \{K_{ij}\})$)
implementation	("`impl`", *cmnt*)

3.1.7 Transcripts

To conclude this section, we describe how a *transcript* of the adversary 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.

Finally, for an adversary A^* we let $IdealWorld(A^*)$ denote the transcript.

Remark 6 *This value is of course a random variable, determined by the random bits of the adversary, the environment, and the values of the session keys. More precisely, $IdealWorld(A^*)$ is actually a vector, or "ensemble" of distributions.*

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 i and user instances (i, j). And as in the ideal system there is an adversary. In addition, there is a special player CA (Certification Authority). The CA has a public key/private key pair that is used to generate *certificates*.

Unlike in the ideal system, users and user instances are not just place holders. A user, when initialized, generates a public key/private key pair, and obtains a certificate from the CA, which is a signature under the CA's key on the user's identity and public key.

User instances are state machines. They implicitly have access to the user's private key, identity, and certificate, as well as the CA's public key. 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 `"continue"`, `"accept"`, or `"reject"`. The meaning of the status value is as follows:

`"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, j).

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

We can assume that user instances play complimentary roles. One user instance, the one who sends the last message, has the role of `"open"`, and the other user instance, the one who receives the last message, has the role of `"connect"`. Classifying the roles in this way is semantically meaningful in analyzing higher level protocols, whereas a classification based on who sends the first message is not.

As in the ideal system, the adversary plays a game. In the very first step in this game, the CA generates a public key/private key pair (PK_{CA}, SK_{CA}) for its digital signature scheme. The public key is made available to the adversary.

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

3.2.1 Initialize user

The adversary assigns an identity ID_i to user i, where user i was not previously initialized and ID_i has not been assigned to other users. User i generates a public key/private key pair (PK_i, SK_i). The CA generates a certificate $cert_i$ which consists of *both* (ID_i, PK_i) and a signature on (ID_i, PK_i) under PK_{CA}.[2]

3.2.2 Obtain certificate

The adversary obtains a certificate c from CA on (ID, s), where s is an arbitrary string, and ID is an identity, subject to the following rule: the set of identities used in *initialize user* operations and the set of identities used in *obtain certificate* operations must be disjoint.

Remark 7 *The point of this operation is to allow the adversary to operate under various identities. Alternatively, these identities can be thought of as corresponding to statically corrupted users. The CA is trusted to ensure that an honest user's identity is not bound to any public key other than that chosen by the user. Note, however, we do not trust the CA to do anything more than this.*

3.2.3 Initialize user instance

In this operation, the adversary chooses a user instance (i, j) that has not been previously initialized, and also specifies $role(i, j) \in \{\texttt{"open"}, \texttt{"connect"}\}$, and an identity PID_{ij}. User i must have been previously initialized.

3.2.4 Deliver message

In this operation, the adversary delivers a message $InMsg$ to an initialized user instance (i, j). As described above, the user instance updates its state, outputs a response message $OutMsg$, and reports its *status*. The response message and status information are delivered to the adversary.

3.2.5 Application

This is exactly the same as the *application* operation in the ideal system, except that the function now computed is a a function of the actual session keys $\{K_{ij}\}$ generated by user instances, as well as a random environment R.

[2]Note that in this paper we *do not* adopt the convention that a signature on a message includes the message itself.

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

initialize user	$(\texttt{"initialize user"}, i, ID_i)$, and $(\texttt{"impl"}, \texttt{"user public key"}, PK_i, cert_i)$
obtain certificate	$(\texttt{"impl"}, \texttt{"obtain certificate"}, ID, s, c)$
initialize user instance	$(\texttt{"initialize user instance"}, i, j, role(i, j), PID_{ij})$
deliver message	$(\texttt{"impl"}, \texttt{"message"}, i, j, InMsg, OutMsg, status)$, and $(\texttt{"start session"}, i, j)$ if $status = \texttt{"accept"}$, and $(\texttt{"terminate instance"}, i, j)$ if $status = \texttt{"reject"}$
application	$(\texttt{"application"}, f, f(R, \{K_{ij}\}))$

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 adversary is probabilistic; we can assume that all of its random coins are in an auxiliary input *coins*. The first record in the transcript is:

$$(\texttt{"impl"}, \texttt{"coins"}, coins).$$

The second record is of the form

$$(\texttt{"impl"}, \texttt{"CA public key"}, PK_{CA}).$$

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

For an adversary A, we let $RealWorld(A)$ denote the 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 two basic requirements.

The first is a *liveness* requirement. This requirement says that for any real world adversary A, whenever the adversary faithfully delivers messages between two user instances with complimentary roles and identities, both user instances accept.

The second requirement is a *simulatability* requirement. This requirement says that for every efficient real world adversary A, there exists an efficient ideal world adversary A^* such that $RealWorld(A)$ and $IdealWorld(A^*)$ are *computationally indistinguishable.*

Remark 8 *Note that the environment R is viewed as random. One could also formulate a definition of security without this stipulation. In this case, the transcript would be a function of R, and the simulatability requirement would be that for all R, $RealWorld(A, R)$ and $IdealWorld(A^*, R)$ are computationally indistinguishable. This could be done, and all of the theorems we prove about the security of protocols would still be true, except that the complexity assumptions would have to be re-cast as* non-uniform *complexity assumptions.*

3.3.1 Discussion

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

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

It may be useful to illustrate this point with a simple example. Suppose that a real world adversary A has the power to simply output a session key after it has been established (but not used). Everything that A can compute is ultimately a function of the real world transcript, and if at some point A can compute a session key, there is an efficiently computable function of the transcript up to that point whose output is the session key. Without loss of generality, we can assume that at the *very end* of the interaction, A requests the session key to be placed in the transcript via an appropriate *application* operation. Now, the appropriate statistical test passes if this key placed in the transcript at the end is equal to the key computed from the transcript prefix. If A really can guess the key, then this statistical test will pass with high probability in the real world. But then the protocol could not have been secure to begin with, since in the ideal world, this statistical test could not possibly pass with high probability, since *nothing* in the ideal world transcript (prior to the very end) is correlated with the session key.

Note that simulation security implies much more than just the inability of an adversary to guess a session key. First, it implies that the session key may be used in any application (such as encryption or message authentication) where a random key is required. Whatever properties the application would have with a truly random key, it also has with a session key. Second, it implies there can be no interleaving or protocol interference attacks.

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 environment 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.

Notice that if we want, at the end of the game we can "dump" all the hidden information $R, \{K_{ij}\}$ into the transcript using *application* operations. This allows a statistical test on transcripts to test relations on these hidden values, as well as on the values visible to the adversary.

Also notice that there is no connection information explicitly available in either real world or ideal world transcripts. However, this information is implicitly available to any statistical test distinguishing transcripts, since the adversary A^* implicitly defines an efficiently computable function from transcripts to connection assignments. This was the point of the restriction in §3.1.4 on how connection assignments are computed. On the one hand, we do not want to place artificial constraints in how connection assignments are computed (indeed, they will almost certainly be defined through a simple "matching conversation" criterion); on the other hand, it seems particularly unfair if a statistical test does not "know" what the intended connection assignments are.

Although we give A^* 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.

Another approach to handling connection assignments would be to define them via a "session ID." In the real world, one could allow a user instance to compute and output a session ID when it accepted. In the ideal world, these values would be specified by the adversary in a *start session* operation. One could easily formulate reasonable rules to establish connection assignments based on matching session IDs.

We have chosen to model higher level application protocols using *application* operations, wherein a function of the session keys and environment is computed. Superficially, it may appear that we are not allowing higher level protocols to have state; however, ultimately, any such state information is a function that can be computed as we have modeled it here. We could have easily formulated an equivalent, alternative definition which modeled higher level protocols as machines with state.

One difference between our approach and that of [2] is that in the ideal world transcript we explicitly log when a session key protocol starts and stops, although we ignore its implementation. In [2], only the successful termination of a session key protocol is logged. We believe that logging both start and stop models more closely the usual "protocol stack" paradigm, and it also allows us to formulate a weak guarantee to a user instance in the `"connect"` role as to the liveness of the connection, which is a property one might wish to exploit in the analysis of higher level protocols.

4 A Diffie-Hellman Based Protocol

In this section we describe and analyze a protocol based on the well-known, unauthenticated Diffie-Hellman key exchange [12]. We call our proposed protocol **DHKE**.

We first begin with a description of the protocol, followed by a discussion of the underlying intractability assumption (the Decisional Diffie-Hellman assumption), and then we present a proof of security.

4.1 Protocol DHKE

Each user generates 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 i's signature algorithm on msg.

We describe the protocol in terms of two users i and i'. User 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 i. We also assume a family of universal hash functions H_k, indexed by a randomly chosen bit string k of some specified length.

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

$i' \to i :\ g^y, k, sig_{i'}(g^x, g^y, k, ID_i), cert_{i'},$
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.

Remark 9 *Following tradition, our description is not entirely precise. Some user instance (i, j) is running the protocol on behalf of user i. This user instance was assigned a partner identity PID_{ij}. The identity $ID_{i'}$ written in the first flow is actually computed by (i, j) as PID_{ij}. Similar remarks apply to other computations in the protocol. It should be clear how to fill in the details.*

Remark 10 *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 the well-known station-to-station protocol. Indeed, the first message generated by i could be sent to several instances of i', at most one of which can actually end up sharing a key with i. At worst, this will lead to user instances (i', j') that are 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. 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. We should not expect a key exchange protocol to solve the two party agreement problem, which is anyway unsolvable in general.*

4.2 The Diffie-Hellman Decision Problem

Let G be a group of prime order q. The Computational Diffie-Hellman assumption (CDH), introduced by [12], is the assumption that computing g^{xy} from g^x and g^y is hard. We know of no truly practical protocols or cryptosystems whose security can be proven under the CDH. What is almost always needed, but never explicitly stated, is the Decision Diffie-Hellman assumption (DDH).

For $g_1, g_2, u_1, u_2 \in G$, define $DH(g_1, g_2, u_1, u_2)$ 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 DH is an efficient, probabilistic algorithm that computes DH correctly with negligible error probability *on all inputs*. The DDH is the assumption that there is no good algorithm for DH.

This formulation is equivalent to the more usual one where

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

There are a few useful random self-reductions that allow us to transform arbitrary inputs to DH into random inputs such that the DH evaluates to the same value.

4.2.1 DDH random self-reduction

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 DH to an equivalent, random input. From this, it follows that the two distributions

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

and

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

are computationally indistinguishable assuming DDH. Moreover, this property holds regardless of whether we consider one or both of g_1, g_2 random or fixed. This indistinguishability property was first observed by Stadler [19].

4.2.2 Using the DDH

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

First, it follows from the DDH, using a hybrid argument, that the two distributions

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

and

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

are computationally indistinguishable.

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 random bit first b is chosen at random, hidden from the view of the adversary. Next, the adversary is given

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

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 $1/2$ 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 to transform random group elements into random bit strings via a universal hash function. See [17, Chapter 8] for an exposition on the Entropy Smoothing Theorem.

The reader is referred to [10, 11, 18, 19] for further applications of and discussions about the DDH.

4.3 Security Analysis of DHKE

The security property for the signature schemes we require is existential unforgeability against adaptive chosen message attack, as defined in [16]. In the sequel, this is what we mean by a secure signature scheme.

Theorem 1 *Protocol* **DHKE** *is a secure key exchange protocol, assuming DDH, 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^* behave just like A, except that it will compute appropriate connection assignments, and the ring master will substitute actual session keys with idealized session keys. Whenever A^* is forced to *expose* a user instance, we have it extract the key from the real world user instance. This is something that the real world adversary cannot do, but that is not in itself a problem. Also, 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 and the real world users and CA. 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, j) for a user instance that is an *originator*, i.e., sends the first message in the protocol, and (i', j') for a user instance that is a *responder*, i.e., a user instance that sends the second message in the protocol. Recall, however, that the *role* of a user instance (`"open"` or `"connect"`) is determined by whether that player sends or receives the last message.

Case 1. Suppose a user instance (i', j') has received the first message in the protocol and has accepted.

Case 1a. If $PID_{i'j'}$ is not the identity of any initialized user, then we *expose* (i', j') in the ideal world. To do this, we need to specify the session key, which we simply extract from (i', j') in the real world.

Case 1b. Now suppose $PID_{i'j'} = ID_i$ for some initialized user i. We assert that at this point there is a unique user instance (i, j) such that $PID_{ij} = ID_{i'}$ and that (i, j) sent g^x in its outgoing message. This follows easily from the logic of the protocol and the security of the signatures. So we *open* (i', j') for connection, and the ring master substitutes the *actual* session key by a *random* session key. We have to argue that this substitution is unnoticable (computationally, that is). But this will follow using the argument in §4.2.2, provided the user instance (i, j) mentioned above has not be or ever will be exposed. But this is so, because $PID_{ij} = ID_{i'}$, and so such an *expose* operation is not allowed in the rules. The only possible outcomes for (i, j) are that it never accepts, it connects to (i', j'), or it connects to another instance of user i'.

Case 2. Suppose that user instance (i, j) has just received an incoming message in the protocol and has accepted.

Case 2a. If PID_{ij} is not the identity of any initialized user, then we *expose* (i, j), extracting the needed session key from (i, j) itself.

Case 2b. Now suppose $PID_{ij} = ID_{i'}$ for some initialized user i'. We assert that there is a unique user instance (i', j') open for connection 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, j) to (i', j'), and the ring master substitutes the actual session key of (i, j) with the session key of (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 liveness requirement is satisfied.

5 An Encryption Based Protocol

5.1 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 §4, we let $sig_i(msg)$ denote user i's signature on msg. Also, we let $E_i(msg)$ denote an encryption of msg under user i's public key.

We describe the protocol in terms of two user's i and i'.

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

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

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

5.2 Security analysis of EKE

To prove security, we need to assume that the encryption scheme is non-malleable, which is equivalent to the notion of security against adaptive chosen ciphertext attack [14]. For more details on this notion of security, see the paper [3].

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 §4.

Case 1. Suppose a user instance (i', j') has just received the first message in the protocol, and has accepted.

Case 1a. If $PID_{i'j'}$ is not the identity of any initialized user, then we *expose* (i', j') in the ideal world, extracting the key K from (i', j').

Case 1b. Otherwise, suppose $PID_{i'j'} = ID_i$ for some initialized user i. Then we *open* (i', j') for connection in the ideal world, and the ring master substitutes the session key held by (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, j) has just received a message.

Case 2a. Suppose that PID_{ij} is not the identity of any initialized user.

If the ciphertext α received was generated by any user instance (i', j') with $PID_{i'j'} = ID_i$, then we can safely make (i, j) reject, as the identity embedded in α ($ID_{i'}$) is not what (i, j) 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, j) run to completion, If it accepts, we *expose* (i, j), extracting the session key from (i, j). This of course makes implicit use of user i's decryption function, but we have taken care not to decrypt anything that was encrypted by a user instance (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 assumption made above in claim (1b) that we never decrypt ciphertexts created by user instances that were opened for connection.

Case 2b. If $PID_{ij} = ID_{i'}$ for some i', we proceed as follows. If the signature verification succeeds, then the ciphertext α must have been created by a unique user instance (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, j) to (i', j') is a valid operation at this moment, because of the uniqueness of the r values. So we *connect* (i, j) to (i', j'), and the ring master sets the session key of (i, j) to the key that it generated for (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 liveness requirement is trivial.

6 A Formal Model for Security Against Adaptive Adversaries

We now extend our formal security model to deal with *adaptive adversaries*—that is, adversaries that can adaptively corrupt users at any point in the interaction. There are many possible types of adaptive corruptions one can consider:

- upon corruption, the adversary learns the long-term secret key of the user;
- or additionally, the adversary learns the entire internal state of all the user instances, excluding data that has been *explicitly erased*;
- or additionally, the adversary learns *everything* about all the user instances, which essentially means the adversary sees all the random bits generated by the user instances.

We consider here only the first type of adaptive attack, where the adversary only learns the long-term secret key of a user. Additionally, we allow a corruption to encompass the possibility of a fault in the *CA*, allowing the adversary to obtain a certificate of his choice on a user's identity.

This threat seems the most realistic. However, in §11, we also consider corruptions where the adversary can obtain non-erased state information of user instances.

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 liveness and simulatability, will remain exactly the same.

6.1 The Real System

The adversary may execute a *corrupt user* operation. The adversary specifies a user i that has been previously initialized, but not previously corrupted, and obtains the user's private key SK_i.

The following two records are added to the transcript:

$$(\texttt{"corrupt user"}, i),$$

and

$$(\texttt{"impl"}, \texttt{"user secret key"}, i, SK_i).$$

Additionally, at any point in time after a *corrupt user* operation, we allow the adversary to perform *obtain certificate* operations on ID_i

Remark 11 *Note that a corruption reveals no information other than the user's secret key. We do not have to modify the model to allow an adversary to obtain session keys: if we want to model this, we can already do this via* application *operations.*

6.2 The Ideal World

The adversary may execute a *corrupt user* operation, specifying a previously initialized, uncorrupted user i. The record

$$(\texttt{"corrupt user"}, i)$$

is added to the transcript.

No information is given to the adversary in the ideal world when a user is corrupted (but see second remark below).

The only other change to the model is that we have to modify the rules governing the *legality* of the connection assignments made during during a *start session* operation. The *actions* that take place when performing these operations do not change, only the rules governing their legality.

The change here is minimal: we allow an *expose* connection assignment on a user instance (i, j) if either i has been corrupted, or if PID_{ij} is not assigned to an initialized user, or is assigned to a user that has already been corrupted.

Remark 12 *It is important to notice what* does not *change. In particular, if we wish, we may still make* open *and* connect *assignments, regardless of whether any of the relevant users have been corrupted. This keeps our definitions simple and natural: if Alice thinks she is talking to a user Bob, but Bob has had his secret key exposed, then Alice may indeed be talking to Bob or to the adversary. It also 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.*

6.3 A more conservative exposure rule

Notice that we allow an *expose* on (i, j) if user i itself has been corrupted. While this may seem fairly natural, one could make a more conservative exposure rule that required that PID_{ij} is not assigned to an uncorrupted user—corruption of i would not be sufficient by itself.

Such a conservative exposure 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 exposure rule.

Second, it would make a difference in which protocols would be considered secure. In §8 we will see examples of protocols that are secure under the liberal exposure rule but not secure under the strict exposure rule.

In the sequel, we will adopt the liberal exposure rule as the "official" rule, but will point out those situations where imposing the conservative rule would make a difference. One could also adopt the conservative rule as the "official" rule, but for right now, we content ourselves with observing the distinction, and pointing out where it makes a difference.

6.4 Non-forward security against adaptive attacks

The formal model we have presented for security against adaptive attacks captures both the notion of simulatability and of *forward secrecy.* This is because, in the ideal world, the adversary gets no information other than the user's long-term secret key.

One can easily formulate a notion of *non-forward* security against adaptive attacks, 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 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.

We know of no practical protocols that can be proved secure in the non-forward sense, but not in the forward sense.

7 Interlude: On the security of protocols DHKE and EKE against adaptive attacks

We now consider the security of protocols **DHKE** and **EKE** against adaptive attacks. We will argue, but not prove, that these protocols are not secure against adaptive attacks. That is, we will indicate the inherent difficulties involved in trying to build a simulator for these protocols. We do not know how to prove that no such simulators exist.

7.1 Protocol DHKE against adaptive attacks

Consider a user i who engaging in the protocol (as an initiator) with n users i' (as responders). Suppose that the first message in the protocol is delivered to each i', and each user i' generates a session key and a response message to be sent to back to the instances of i running the protocol. Now consider the scenario where before any of these messages are delivered to i, each user i' does something interesting with its session key in application protocol, like applying a one-way function to the key, causing this value to be placed in the real world transcript. After all n of the i' users have done this, the adversary selects a subset of the i' users in some weird, unpredictable way and corrupts this subset, obtaining their signing keys. Now, for the uncorrupted players, the adversary forwards their response messages to i. For the corrupted players, we do the following: if the response of the i' user was

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

the adversary instead sends the message

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

where h is a group element generated in some mysterious way by the adversary. the adversary can do this, since it has the signing key of user i'. Now, all instances of player i will accept. Finally, just to be explicit about it, we have *everybody* output their computed session keys.

In order to prove security, one would have to specify what the appropriate connection assignments would be. Now each of the i' users commit to there keys before we know which of them is going to be corrupted. So it seems that we have to perform an *open* assignment operation on each of these, substituting their real keys by random keys. Suppose an instance of user i has received the modified message from a corrupted user i'. We either have to *expose* it or *connect* it. But this instance of i does not share a key with anyone, so connecting it is not possible. So we have to *expose* it. But then the indistinguishability claim would be that given (g^x, g^y, g^z, h^x), we can not tell if $z = xy$ or not. This seems impossible to justify under the DDH. The value h, remember, was chosen by the adversary in some mysterious way that could depend on g^x, g^y, and g^z. These problems do not even go away if we consider non-forward security discussed in §6.4.

At this point, the reader might object that the problem here lies in our formal model. Indeed, one might suggest that our rules for connection assignments are too inflexible: we require that a connection assignment is determined as soon as a user instance terminates the session key protocol. In particular, we must decide to *open* for connection or *expose* immediately. However, this example also illustrates that we really could not exploit any extra flexibility anyway. Since all the i' users essentially publicly commit to their keys before being corrupted, we cannot wait—and we cannot hope to guess correctly who is eventually corrupted.

This is of course not a proof of anything, but we hope it is a convincing argument that proving simulatability using currently known techniques is out of reach. We also believe that the problem is a fundamental one, having more to do with the inherent *malleability* of Diffie-Hellman encryption, than with the particulars of our formal model.

7.2 Protocol EKE against adaptive attacks

At best, it is clear that all we could hope for is that protocol **EKE** is non-forward security against adaptive attacks (§6.4). This is because if we corrupt a user 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 n pairs of users i and 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 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.

8 Diffie-Hellman Based Protocols for Adaptive Attacks

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

8.1 Protocol DHKE-1

We now show how to modify protocol **DHKE** to obtain a protocol that is secure against adaptive attacks. 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.

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

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

$i \to i' : \; k_1$,
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 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 3 *Protocol* **DHKE-1** *is secure against adaptive attacks, assuming DDH, secure signatures, and that G 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, j) has just received its last message, and all the signatures are valid.

Case 1a. Suppose $PID_{ij} = ID_{i'}$ and some instance (i', j') with $PID_{i'j'} = ID_i$ received the g^x, G_i value sent by (i, j) and sent the g^y, k values received by (i, j). In this case, we make (i, j) accept in the ideal world, and we perform an *open* operation, 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 *authentication key* of (i, j) for future reference. Note that we do all of this, *even if user 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 a user, or is assigned to a corrupted user, or user i is itself corrupted. We extract the computed session key and authentication key from (i, j) in the real world, and *expose* (i, j) using the computed session key.

Case 2. Suppose that (i, j) has received its last message, and the signatures do not check. Then we let (i, j) reject in the ideal world, which is what it would do anyway in the real world.

Case 3. Suppose a player instance (i', j') has just received its last message, and all the signatures check.

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

This last point is crucial. User i may have been corrupted after (i, j) accepted and sent its last message, and in the meantime, (i, j) 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 *expose* (i', j') at this point. This is is the situation referred to in Remark 12.

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 i' is not corrupted, and that g^x, G_i came from a user instance (i, j) with $PID_{ij} = ID_{i'}$. Then in the ideal world, we simply make (i', j') reject. See below for a discussion of why this is valid.

Case 3b''. If we reach this subcase, by the logic of the protocol, the only way this could happen is if $PID_{i'j'}$ is not assigned to a user, or it assigned to a corrupted user, or user i' itself is corrupted. We then extract both the session key and the authentication key from (i', j'). We then test the if the received message is equal to the computed authentication key. If this test fails, we let (i', j') reject, just as it would in the real world. Otherwise, we let it accept, and perform an *expose* operation in the ideal world using the computed session key.

Case 4. Suppose that (i', j') has received its last message, and the signatures do not check. Then we let (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 ideal world. Notice that the only time we do an *open* or *connect* operation, we are guaranteed that there is a matching between the g^x and g^y values of the two user instances involved. Because this condition is symmetric, we will never expose one key, while substituting the other instance's key with a random key.

The faithfulness of the simulation now follows from the DDH. But there is one subtle point worth mentioning: the "forced" rejection by user instance (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', j') what he really wanted to do. But consider the user instance (i, j) referred to in that subcase. Since user i' has not been corrupted at this point, the adversary could not have forged any messages on behalf of user i'. Therefore, either (i, j) 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 i'. So at this point, under the DDH, the value k_1 that (i', j') is expecting is (computationally) independent from the adversary's view in the real world. Thus, letting (i', j') reject is the right action.

That completes the proof of the theorem.

Remark 13 *Key confirmation is a mysterious and ancient tradition practiced by protocol designers. There has never been a satisfying explanation of why they did this. Now we know: to allow a proof of simulatability against adaptive attacks.*

8.2 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 the station-to-station protocol.

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

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

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

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

Theorem 4 *Protocol* **DHKE-2** *is secure against adaptive attacks, assuming DDH and secure signatures.*

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

Case 1. Suppose (i,j) has just accepted.

Case 1a. Suppose $PID_{ij} = ID_{i'}$ and some instance (i',j') with $PID_{i'j'} = ID_i$ received the g^x, G_i value sent by (i,j) and sent the g^y, k value received by (i,j). Then we perform an *open* on (i,j), and K_{ij} is substituted with a random key. We will see below that (i',j') is not *exposed*, it either rejects or *connects* to (i,j). It follows by the DDH 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 i is corrupted, or PID_{ij} is not assigned to an uncorrupted user. So we *expose* (i,j).

Case 2. Suppose (i',j') has just accepted.

Case 2a. Suppose $PID_{i'j'} = ID_i$ and some instance (i,j) with $PID_{ij} = ID_{i'}$ sent the g^x, G_i value sent by (i,j) and received the g^y, k value sent by (i',j'). Then we make (i',j') *connect* to (i,j). From the arguments above, (i,j) has not been *exposed*, and it is thus clear that (i,j) is open for connection.

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

8.3 Protocol DHKE-3

Although protocols **DHKE-1** and **DHKE-2** are secure against adaptive attacks using the liberal exposure rule, it is perhaps interesting to note that they are apparently not secure under the conservative exposure rule (§6.3). To achieve this stricter sense of security, 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 strict exposure rule.

In the proof of Theorem 3, 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 i was corrupted, not because i' was corrupted. Under the conservative exposure rule, we are not allowed to *expose* here. The problem is even worse in case (3b′). There, we really need to *expose* if i' is corrupted, but again, the conservative exposure rule forbids this. The same problem that arose with G_i in the proof of Theorem 3 also arises in the proof of Theorem 4, 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 i. In the previously discussed Diffie-Hellman based protocols, this information was in the certificate of user i. In this protocol, we do not require this—under the strict exposure rule, it does not help.

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

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

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

Theorem 5 *Protocol* **DHKE-3** *is secure against adaptive attacks, using the conservative exposure rule, assuming DDH and secure signatures.*

The proof is almost identical to that of Theorem 4. The only difference is case (1b). Here we can conclude that if there was no matching (i', j'), then it must be the case that PID_{ij} did not belong to an uncorrupted user. We omit further details.

9 An Encryption Based Protocol for Adaptive Adversaries

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. 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.

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

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

The agreed upon session key is K, which user i obtains by computing $D(\alpha)$. As usual, both users check the relevant signatures. Additionally, user i checks that the decryption of α is of the right form, containing the expected identity.

Theorem 6 *Protocol* **EKE-1** *is secure against adaptive adversaries, 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', j') has just terminated successfully.

Case 1a. Suppose $PID_{i'j'} = ID_i$ and some instance (i, j) with $PID_{ij} = ID_{i'}$ sent the E value received by (i', j'). In this case, we *open* (i', j') for connection, substituting its real session key with a random one. This will be undetectable, provided we never decrypt this α. We justify this claim below.

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 $PID_{i'j'}$ is not assigned to a user or is assigned to a corrupted user. In this case, we perform an *expose* operation, extracting the session key from (i', j').

Case 2. Now suppose a user instance (i, j) has just received a message and all the signatures are valid.

Case 2a. Suppose the ciphertext α received was generated using E by a user instance (i', j') with $PID_{i'j'} = ID_i$ and $ID_{i'} = PID_{ij}$. Then we *connect* (i, j) to (i', j'). This will be valid, because of the uniqueness of the E values across different instances of user i (we are letting E double as a nonce here).

Case 2b. Suppose that the ciphertext α received was generated using E by a user instance (i', j') with $ID_{i'} \neq PID_{ij}$. In this case, we let (i, j) reject, without decrypting α. This would anyway happen in the real world, as the identity embedded in α is not what (i, j) is expecting.

Case 2c. Suppose that neither of the conditions in cases (2a) or (2b) hold. By the logic of the protocol, either PID_{ij} is not assigned to a user, or is assigned to a corrupted user. In this case, we run (i, j) to completion, and perform an *expose* operation if it terminates successfully. This involves an implicit application of a decryption algorithm. This is the only place we decrypt anything, so we have to make sure we do not invalidate the assumption made in case (1a). But if the decryption α was generated by a user instance (i', j') satisfying the conditions in case (1a), then case (2c) could not apply to (i, j).

Case 3. Suppose a user instance (i, j) has just received a message and not all signatures are valid. Then we just let (i, j) reject, just as it would in the real world.

The indistinguishability property follows from the assumed security of the encryption scheme (using a standard hybrid argument).

That completes the proof of the theorem.

Remark 14 *It is easy to see that protocol* **EKE-1** *remains secure even under the conservative exposure rule (§6.3).*

Remark 15 *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 16 *We have to generate a new public key/private key for encryption with every run of the protocol. For RSA-like schemes, this can be impractical. We can use a Diffie-Hellman based encryption scheme such as that of Cramer and Shoup [11], which is provably non-malleable assuming the DDH. In practice, one could use the same group all the time, changing the random exponents with every run. The total cost, excluding signatures, would be about 7 exponentiations for user i, and about 4 exponentiations for user i'. This assumes one uses an efficient hash functions, like SHA-1, under a second preimage collision resistance assumption. This compares to a cost of 2 exponentiations each in protocols* **DHKE-**n*. Thus, while protocol* **EKE-1** *uses just two passes, it does this at the cost of some extra computation. So,* **EKE-1** *might be preferable in a network where communication costs dominate computation costs, assuming, of course, that one actually cares at all about provable security.*

Remark 17 *We shall consider stronger forms of adversaries that can obtain all unerased internal state of user instances in §11. We will see that in this scenario, protocols* **DHKE-**n *are secure, whereas* **EKE-1** *is not.*

10 Comparison with the Bellare-Rogaway Model

Bellare and Rogaway [6] have presented a formal model for secure key exchange protocols. Technically speaking, their model applies only to the symmetric-key case where we have an on-line trusted center (the "Kerberos" model). It is relatively straightforward to adapt this model to the case we are examining here, without a trusted center (other than the *CA*). This program has been carried out by Blake-Wilson, *et al* [8, 9]. In these papers, they actually discuss two models: one for key exchange, and one for key exchange with key confirmation. It is the first model that adapts that of [6]. 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.

10.1 Definition of BR security model

Instead of recalling all the notation of [6, 8, 9], we show what the BR model essentially is in in terms of our notation.

Consider an adaptive real world adversary A, as we have described it, except that there are no *application* operations allowed (these are replaced by something else, see below). To prove BR security, one has to exhibit a *connection assignment* function. This is a function whose output is the connection assignments, *open*, *connect to* (i', j'), *expose*. This is a function of the partial transcripts built up during the execution of A with the system, and should be a fixed function that works for all A. Connection assignments must also satisfy the usual consistency constraints, as described in §3.1.4 and §6.2. There are, however, no actions associated with these connection assignments. For simplicity, let us call two user instances *partners* if one is connected to the other.

Now, there are three parts to the definition of BR security.

The first part is the same as the liveness requirement in the simulation security.

The second part says that if two user instances are partners, then their keys must agree.

The third part requires that the adversary has negligible advantage in the following game.

During execution, the adversary may perform any number of *reveal* operations. These may be applied to any user instance that holds a session key, regardless of the connection assignment of the user instance. Upon performing a *reveal* operation, the adversary receives the corresponding session key.

Additionally, the adversary may perform a single *test* operation on some user instance that holds the session key. At this point, a coin is flipped, and the adversary is either given the user instance's session key or a random string.

There is a restriction on the use of *test* operations. We call this the **test restriction**, which states that the following configuration is explicitly forbidden *at any time*: (i, j) was subject to a *test* operation and

T1 (i, j) has a connection assignment of *expose*, or

T2 (i, j) or its partner (if any) were the subject of a *reveal* operation, or

T3 i was corrupted, or PID_{ij} belongs to a corrupted user.

The adversary's advantage is defined to be the distance from 1/2 of the probability of guessing the outcome of the coin toss in the *test* operation.

That completes our description of BR security. There are a few small, technical differences between our presentation here that are not so important: the precise rules for establishing partners here and in the papers [6, 8, 9] all differ slightly.

However, we have taken the opportunity here to correct a serious flaw that appears in [6, 8, 9] that was pointed out to the authors of [6] by Charles Rackoff. In the formulation in [6, 8, 9], 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 from Charles Rackoff. Take any protocol that is secure in whatever model you like (as strong as or stronger than the BR model). Suppose this protocol ends with a flow from i to i'. Suppose H is some function that i' might apply to a session key in a reasonable application protocol (like an encryption or message authentication function). Now we construct a modified protocol by adding two extra flows:

$i' \to i : x = 0$

$i \to i' : y$, where $y = K$ if $x = H(K)$, and $y = 0$ otherwise

Note that player i' waits for the message y, but otherwise ignores it. Both parties compute the same session key as in the original protocol.

We think the reader would agree that this protocol should in fact be considered insecure under any reasonable definition of security. If the *test* operation is allowed in the middle of the game, it is easy to see that an adversary can obtain significant advantage. This can be done by forcing i' to accept before i does, doing a *test* on i', and then we can make i "spit out" its session key with probability $1/2$. However, if the *test* operation is allowed only at the end of the game, the adversary has no advantage, and in fact, the protocol would be secure under that definition of security. To see why this is so, we can define the connection assignments to be exactly the same in the new protocol as in the old. Now, the the adversary does a *reveal* operation on i', he will be able to make i "spit out" his session key, but he can not gain an advantage, because he will not be able to *test* i, because these instances of i and i' will be partners. We leave it to the reader to verify that this protocol is indeed secure under this definition.

In the BR security model, an adversary may adaptively corrupt players, so one could distinguish between static and adaptive BR security. However, these notions are equivalent.

Theorem 7 *Adaptive BR security is equivalent to static BR security.*

To see this, first note that adaptive BR security trivially implies static BR security. To show that static security implies adaptive security, consider an adaptive BR adversary A with non-negligible advantage. We can convert A into a static BR adversary A' with smaller, but still non-negligible advantage as follows. At the beginning of the game, we randomly choose two players i and i'. We pray that A chooses to perform his *test* operation on an instance (i, j) with $PID_{ij} = ID_{i'}$ (which implies that i and i' end up being initialized but not corrupted). Our prayer will be answered with non-negligible probability. Furthermore, A' never actually initializes or corrupts any users other than i or i', but simply controls them directly. The theorem now follows.

From the above argument, one can see as well that it does not matter if the corruptions in the BR model reveal ephemeral internal state information of user instances or not.

10.2 Relation between BR security and simulation security

Now, we want to formulate and prove that BR security (the above, corrected formulation) and simulation security against static adversaries are equivalent. To do this, we have to restrict the way the adversary A^* computes connection assignments to the way connection assignments are computed in the BR model. This means that a connection assignment must be computed as a function of the partial transcript built up so far, excluding the `"application"` records, and that this function should not depend on A, but should be a universal function that works for all A. This does not seem like a serious restriction, and we could have formulated simulation security with this restriction to begin with, but it seemed like an unnatural restriction; moreover, this restriction does not affect any of our proofs of security. *For the remainder of this section, this restriction on the adversary A^* in the ideal world model is implicitly in force.*

Theorem 8 *Security in the simulation model against static adversaries is equivalent to BR security.*

To prove that security in the simulation model implies BR security, one only need observe that a static BR attack game is just a particular game that can be easily modeled in the static simulation security model.

Now, to prove that BR security implies security in the simulation model against static adversaries. Assume a given protocol is BR secure. Then there exists an appropriate connection assignment function that we can let A^* use.

Now, whenever a user instance accepts, the simulator A^* does what the connection assignment function specifies. For an *expose* operation, we extract the user instance's actual session key to obtain the session key required for the *expose* operation in the ideal world game. Otherwise, if we *open* or *connect*, we let the ring master choose 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/real world game who uses the operations *reveal* and *test* to generate some keys, and generates other keys as random. 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 *exposed*—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 *exposed*.

The completes the proof of the theorem.

Remark 18 *Since BR security is equivalent to static simulation security, it is apparently not equivalent to the possibly stronger notion of non-forward adaptive simulation security (§6.4).*

10.3 The Forward BR security model

It is instructive to see where the proof of Theorem 8 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 BR game explicitly prohibits doing a *test* operation on anybody who is corrupted or is partnered with someone who is. If we think back to the discussion of protocol **DHKE** in §7.1, 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 to obtain a model equivalent to the simulation model. We call this the *forward BR security model*. It is the same as the original BR model, except that we replace the above **test restriction** by a **weak test restriction** which we obtain by simply dropping the restriction **T3**.

Notice the only change here: we have just erased the references to corrupted users. It is the *expose* connection assignment that we use to selectively prevent *test* operations where we do not want them—namely, *after* a corruption. This clearly captures the notion of *forward security*.

Theorem 9 *Adaptive security in the simulation model is equivalent to forward security in the BR model.*

The proof is straightforward. We omit the details.

10.4 Discussion

We have shown that the BR security model (with one crucial "fix," as discussed above) is equivalent to static simulation security. This is so, even though the BR model ostensibly allows "adaptive" corruptions.

We have also shown that by strengthening the BR security model in the natural way to capture the notion of forward security, we obtain a model that is equivalent to adaptive simulation security.

The notion of non-forward adaptive simulation security (§6.4) appears to be a notion of security that cannot be easily captured in the framework of BR security.

As we have seen, the real difference between BR security and simulation security arises because of adaptive corruptions. Now, a corruption is only truly adaptive if the adversary bases his choice of who and when to corrupt on the network traffic. One could make a reasonable argument that in practice, this is typically not the case—the choice is most likely completely determined by other factors. So perhaps simulatability against adaptive adversaries as we have defined it is stronger than necessary. It might be nice to have a weaker definition of simulation security (in both the forward and non-forward senses) that captured the notion that the adversary's choice of who and when to corrupt is pre-determined. However, since it is not too difficult to defend against truly adaptive attacks, it is not clear how worthwhile this approach would be.

11 Adaptive Corruption of User Instances

So far, we have only considered adaptive attacks in which the adversary obtains long-term secret keys of a user, and the ability to obtain bogus certificates for a user. A different form of attack is one in which the adversary gets to "look inside" a user instance and obtain all *unerased* state information.

It seems a debatable point as to how realistic such types of attacks are. The information that the adversary is gaining here is "ephemeral" data that is stored for a relatively short time while running the key exchange protocol. Carrying out this attack is much harder than obtaining a long-term secret key, or even obtaining a session key. Moreover, if the adversary really does have the power to find this ephemeral data, it is quite hard in practice to make sure that the data that was supposed to be erased cannot be found by the adversary: copies of such data may appear at several levels of a computer's memory hierarchy, and making sure that all such copies have been erased is in practice quite difficult.

Another reason why such attacks are a bit esoteric is that even if we achieve security against them in the session key protocol, it will be hard to extend the simulation to typical high level applications. Indeed, suppose a higher level application is using a session key to encrypt data. Now, one of the reasons we want to prove simulatability of the session key protocol is so that we can also prove simulatability of higher-level protocols. However, all of the practical symmetric key encryption schemes used—or even known—today will not allow one to achieve simulatability if the session key is revealed. At best, such schemes will provide semantic security, or even non-malleability, but *only* if the keys are never revealed.

While this attack model is perhaps a bit esoteric, it is still worth considering. One could also consider attack models where erasure is not possible. In this model, the adversary would simply obtain the random coins by the user instance, since any additional information could be easily computed by the adversary. However, we know of no practical protocols that are secure against such stronger types of attacks.

11.1 Definitions

We will only allow the adversary to corrupt a user instance that is running, i.e., has been initialized, but is not yet finished. We do not allow the adversary to corrupt a user instance that is already finished, since we can assume that at this point all information other than the session key has been erased, and the session key can be obtained via an *application* operation. Also, corrupting a user instance does not directly compromise the user's long term secret key; this can still be achieved via a *corrupt user* operation, but that is a separate, and orthogonal operation.

Obviously, when a user instance is corrupted, its session key is compromised, and so might the session key of a corresponding user instance. But this should be the extent of the "damage," and our formal definition captures this idea. Our definition is quite conservative on this point: if one user instance is corrupted, at most one other corresponding user instance can be considered compromised.

As in the corruption of a user, when a user instance is corrupted, some data is revealed to the adversary, but the user and the user instance continue to function normally, faithfully executing the protocol as if nothing happened.

Now the details.

In the real world, *corrupt user instance* operation adds the records

$$(\texttt{"corrupt user instance"}, (i, j))$$

and

$$(\texttt{"impl"}, \texttt{"user instance state"}, (i, j), \mathit{InternalState})$$

to the transcript, where *InternalState* includes all non-erased internal state of this user instance. The user instance (i, j) must have started the protocol, but not yet have terminated.

Now we describe how a *corrupt user instance* operation works in the ideal world.

When the adversary performs this operation in the ideal world, the record

$$(\texttt{"corrupt user instance"}, (i, j))$$

is added to the transcript. The user instance (i, j) must have started the protocol, but not yet have terminated.

Now we have to explain the details of the rules that capture our intuitive goal that corrupting one user instance should compromise at most one other corresponding user instance.

At any time, a corrupted user instance is either *unbound* or *bound.* Initially, upon corruption, it is unbound. The ideal-world adversary has at this time the option of making a (pre-mature) connection assignment, connecting (i, j) to a user instance (i', j'), provided all the usual conditions for making this connection assignment are met. If the adversary chooses this option, then the ring master gives the adversary $K_{i'j'}$, and user instance (i, j) becomes *bound.* We call this the *special connection rule.*

When a *start session* operation is performed, the rule allowing a connection assignment of *expose* is relaxed so that it may be applied to any user instance that has been corrupted at some time in the past, unless the adversary applied the special connection rule to this user instance, in which case, that connection assignment remains in force. We call this the *relaxed exposure rule.*

Further, the adversary is allowed to make an *expose* connection assignment for any user instance (i, j) by specifying an unbound corrupted user instance (i', j') such that (i, j) and (i', j') have complimentary roles and identities. If this is done, user instance (i', j') becomes bound. We call this the *special exposure rule.*

As usual, whenever the adversary specifies a connection assignment of *expose*, he himself supplies the session key K to the ring master.

That completes the description of the modifications to the ideal world interaction. The reader is advised to take this description quite literally: these are the *only* changes, and *nothing* goes in to the transcript other than what is specified above.

11.2 Implications

Theorem 10 *Protocol* **DHKE-3** *is secure against adaptive user corruptions and adaptive user instance corruptions, assuming DDH and secure signatures.*

This theorem assumes that the protocols are implemented so that secret data (x and y) is erased as soon as it is no longer needed. Note also that the choice of liberal vs. conservative exposure rule is orthogonal to the issue of user instance corruptions. This theorem holds using the conservative exposure rule.

The proof is fairly straightforward. We briefly sketch how one utilizes the special connection and exposure rules to carry out the simulation.

Suppose a user instance (i', j'), in the role of a responder, is corrupted at some point.

- Suppose (i', j') has had a matching conversation with a user instance (i, j), and (i, j) has already accepted at the time that (i', j') was corrupted. Then A^* will connect (i', j') to (i, j), using the special connection rule. In doing this, A^* will obtain K_{ij}, which it can then use to simulate the internal state of (i', j').

- Suppose (i', j') has a matching conversation with a user instance (i, j), but (i, j) accepts after (i', j') was corrupted. Then A^* will expose (i, j) using the special exposure rule.

Suppose that a user instance (i, j), in the role of an initiator, is corrupted at some point. Here, we can make use of the ordinary exposure and connection rules. When (i, j) eventually accepts, we simply expose it. If there is an (i', j') that accepts and has a matching conversation with (i, j), then the ordinary connection assignment rule will assign K_{ij} to $K_{i'j'}$ (it so happens that A^* knows K_{ij}, but that is irrelevant).

Remark 19 *It is also worth observing that* **DHKE** *is apparently insecure against adaptive user instance corruptions by considering an example that is much less subtle than that discussed in §7.1. A user i sends its message to user i'. We will have to* open *this instance of i', substituting g^{xy} with g^z, because nobody is as yet corrupted. But immediately after i' accepts, the adversary can corrupt i, obtaining x, because x can not be erased at this point. But now we will have an unrepairably inconsistent transcript. Apropos Remark 13, we now have a further explanation for the mysterious and ancient tradition of key confirmation.*

Note that protocol **DHKE-1** is apparently insecure against adaptive user instance corruptions. This is because of our requirement that at most one corresponding user instance is compromised when one user instance is corrupted. If the secret value x is obtained from an instance of user i in that protocol, the adversary can easily create a number of instances of a user i' such that all of their session keys are known to the adversary.

Also note that protocol **EKE-1** is also apparently insecure against adaptive user instance corruptions. The problem is that user i' may accept and start using a session key before i has received the message from i'. In the meantime, the adversary could corrupt i, obtaining is ephemeral decryption key.

To fix this problem, we could modify protocol **EKE-1** by adding a third "key confirmation" flow, as in **DHKE-3**. If we do this, non-malleability is no longer necessary—semantic security is enough. In fact, protocols **DHKE-n** can be viewed as a special case of this using ElGamal encryption [15].

It seems unlikely that we can obtain a two pass protocol that is secure against adaptive user instance corruptions, at least without using "random oracles" [5]. We have not fully worked out the proper formulation of random oracles in the context of session key protocols. However, it seems that it should be possible to formulate and prove that if **EKE-1** is modified so that the session key is $H(K)$ instead of K, then modeling H as a random oracle, **EKE-1** should be provably secure. To do the simulation, it seems that one must give A^* the ability to simulate the random oracle. This can be done by logging oracle queries with *implementation* records in the transcript. This means that one would have to rule out oracle access in *application* operations. This would mean that honest players running higher level protocols should not access the random oracle (just as they do not access the long term secret keys of the session key protocol).

12 Anonymous Users

In this section, we introduce a formal model of security for session key protocols in which one of the two users may be *anonymous*. By an anonymous user, we simply mean one without a certificate.

SSL and SSH are examples of session key protocols that allow an anonymous user.

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.

12.1 Definitions

Adding anonymous users is simple. We create a special user 0 with a special identity `"anonymous"`. An entity that wants to run the protocol as an anonymous user will simply use a user instance $(0, j)$ for this purpose. Note that user 0 has no associated secret key, and is considered to be pre-initialized. Since user 0 has no secret key, it cannot be corrupted.

Beyond the introduction of this special user 0, the rest of the model stays exactly the same as before, with one exception: the ideal-world adversary A^* is always free to make a connection assignment of *expose* for a user instance (i, j) if $PID_{ij} =$ `"anonymous"`. Intuitively, this relaxation of the exposure rule is necessary, since as we remarked above, an anonymous user may really be the adversary himself.

Note that we can combine anonymous users with the different types of corruptions (static user, adaptive user, and adaptive user instance) and different types of exposure rules (conservative and liberal).

12.2 Implications

Let us enhance protocol **DHKE-3** to accommodate anonymous users. We will let user i, the initiator, play the role of the the non-anonymous user, and user i', the responder, play the role of the anonymous user. The protocol is exactly the same as **DHKE-3**, except that the anonymous user does not have a certificate, and so sends neither this certificate nor a signature to user i. Here is the protocol.

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

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

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

Note that $ID_{i'} =$ `"anonymous"` in the above.

Theorem 11 *The above modification of protocol* **DHKE-3** *to accommodate anonymous users is secure against adaptive user corruptions and adaptive user instance corruptions, assuming DDH and secure signatures.*

Note that as with the original **DHKE-3**, this theorem holds under the conservative exposure rule. We leave the details of the proof to the reader.

13 Conclusion

There are several directions in which our models and protocols can be extended.

First, one might want to allow signature and decryption algorithms used in a session key protocol to be used in other protocols as well. Our formal security model disallows it, but it could be done as long as all messages that were signed or encrypted were appropriately tagged to specify to which protocol the message belonged. All protocols using the same signature or decryption algorithms would have to agree on a consistent set of rules for doing this.

Second, a key exchange protocol may actually consist of a number of protocols, with various versions, choices of cryptographic algorithms, etc. These notions can easily be adapted into our formal model. Perhaps the easiest way to do this is to bind such version information to a user's identity when running the protocol.

References

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

[2] 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.

[3] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions security for public-key encryption schemes. In *Advances in Cryptology–Crypto '98*, 1998.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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`.

[9] 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`.

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

[11] 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.

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

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

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

[15] T. El Gamal. A public key cryptosystem and signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

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

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

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

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