Session Key Distribution Using Smart Cards

Victor Shoup Avi Rubin

Bellcore, 445 South St., Morristown, NJ 07960

{shoup,rubin}@bellcore.com

Abstract

In this paper, we investigate a method by which smart cards can be used to enhance the security of session key distribution in the third-party setting of Needham & Schroeder. We extend the security model of Bellare & Rogaway to take into account both the strengths and weaknesses of smart card technology, we propose a session key distribution protocol, and we prove that it is secure assuming pseudo-random functions exist.

1 Introduction

Let us recall the basic session key distribution problem in the third-party setting of Needham & Schroeder [4]. We have a *server* and a set of *hosts*, all of which contain secret keys. During the course of the protocol, on each host several *processes* attempt to establish session keys with processes on other hosts. The processes communicate with each other and with the server. We assume that an *adversary* has complete control over the communications network: it may deliver messages out of order, delete or modify messages, and create new messages or even initiate new processes. We also assume that the adversary can obtain session keys that have already been established, and can even corrupt a host, obtaining its long-term secret key.

Faced with such a powerful adversary, the goal of a session key distribution protocol is basically to prevent the adversary from obtaining any information about session keys that it should not "obviously" have.

In the important paper of Bellare & Rogaway [1], these notions were for the first time properly formalized, and a protocol was presented and proved secure, assuming the existence of pseudo-random functions (PRFs) [2]. Many papers have addressed the session key distribution problem, and we refer the reader to [1] for more references and a discussion of its history.

One source of insecurity in this setting is the storage of the long-term secret keys on the hosts. If *ever* an adversary can corrupt a host and obtain its secret key, then for that host *all* is lost: all conversations past and future encrypted using session keys established by processes on that host can be decrypted by the adversary. Faced with the reality of known (and unknown) security holes in operating systems and other software, this is a serious problem.

One way to address this problem is to store the secret keys on a "smart card," or other kind of secure, tamper-resistant chip, instead of on the host. However, there has been little formal analysis of exactly what security gains this approach might offer.

If we are going to place the secret key in a special computing device, requiring all access to the key to go through this device, it seems both useful and natural to make this device as simple and computationally limited as possible. This leads to the following point of view:

- we shall use a smart card only to generate session keys, which are then stored in the private memory of a process on the host which performs encryption or other cryptographic functions itself;
- we shall use a smart card only as a *stateless* probabilistic device, or probabilistic oracle: on input x, the card outputs a function whose value is determined by x, its secret key, and some random bits; in particular, the output does not depend on previous inputs.

The first assumption is justified by the very limited computational power of smart card chips, as well as their very low I/O bandwidth. The second assumption is justified by the very limited amount of memory available on smart cards, and allows an unbounded number of processes on a given host to be simultaneously executing the protocol with minimal interference with one another, and minimal overhead and bookkeeping on the smart card.

Given this setting, we extend the notion of security. As before, the adversary has complete control over the communications network, and we allow the adversary to obtain the session key from a process upon demand; more generally, the process will give the adversary all of the information it obtained through its interaction with the smart card.

Under normal circumstances, only a process on the host will be able to directly access the smart card. However, the adversary may occasionally be able to directly access the smart card. Note that this type of attack actually encompasses many possibilities: a network break-in or virus, the replacement of the host's software by rogue software, and physically stealing or borrowing the smart card. Obviously, while an adversary has access to a host's smart card, it can mimic a process on that host. In our formal definition of security, we shall require that this is essentially all the adversary can do; in particular, the adversary gains no knowledge of session keys generated at any other time, past or future.

Also, if an adversary physically steals the card, then with some effort and expense, it may be able to break the hardware security defenses of the card, and obtain its secret key. We of course allow this as a possibility, but require that session keys unrelated to this host are still secure.

We shall formalize the above security model, and present a protocol that is secure in this model. We build on the work of Leighton & Micali [3], designing a protocol in which the server is essentially just another smart card, and in which access to the server is required only the first time a process on one host talks to a process on another host. Also, both the server and the host smart cards need only compute a few pseudo-random functions, which might be very efficiently implemented using DES or a keyed version of a hash function such as SHA or MD5.

The rest of this paper is organized as follows. In §2, we show how to modify and extend the Leighton-Micali protocol to obtain a provably secure session key distribution protocol in the original Bellare-Rogaway model. We include this, since to obtain our main result, we need to make very similar arguments, and the result itself may be of independent interest. Then in §3 we present our formal definition of security in the smart card setting, and we describe and analyze our new protocol.

2 Session Key Distribution Without Smart Cards Revisited

2.1 The Bellare-Rogaway Model

We first recall the formal model and definition of security for a session key distribution given in Bellare & Rogaway [1], which we paraphrase and simplify somewhat.

At system initialization, a security parameter k is specified, along with some number n of hosts. For simplicity, we assume the hosts are named $1, \ldots, n$, although our definitions and results easily extend to host names chosen dynamically by an adversary. Some random bits are generated, and keys $K, K_1, \ldots, K_n$ are generated, K is stored in the server S and each K_i is stored in each host i.

For each pair i, j of hosts, and each $u \geq 1$, there is a process $\Pi(i; j, u)$ which is attached to host i and is (attempting) to establish a session with a process on host j. As there may be more than one such process, we index them by u, as indicated.

The adversary is a polynomial-time, probabilistic algorithm. On input 1^k, it chooses n, bounded by a polynomial in k, and then runs the system initialization routine. The adversary then interacts with the processes and the server. This interaction is simply a sequence of time-ordered question/answer pairs: the adversary asks a question addressed to a process, a host, or the server and gets an answer. The time-ordered list of questions and answers is called the *transcript*, and this together with the adversary's coin tosses is its *view*.

The simplest type of question and answer corresponds to the delivery of a message to a process or the server and a response. Note that a process carries state, so that the answer to one question may depend on previous questions sent to that process. As the interaction proceeds, a process may output a special message indicating *acceptance*, which implies that that process has established a session key. Note that in all of our protocols, the server carries no state, so we do not introduce the notion of multiple instances of the server.

There are two other types of questions the adversary can ask. First, the adversary can request that a process $\Pi(i; j, u)$ that has accepted reveal its session key; in this case, we say $\Pi(i; j, u)$ has been *opened*. Second, the adversary can request a host i to reveal its secret key K_i; in this case, we say that host i has been *opened*.

The session key distribution protocol specifies how any process will answer a sequence of questions, and how the server answers its questions. The protocol should also specify a *partner function*. This is a function that maps a process $\Pi(i; j, u)$ to a process $\Pi(j; i, v)$, or indicates no partner, and which can be computed efficiently from the adversary's transcript.

A process $\Pi(i; j, u)$ is said to hold a *fresh* session key if the following conditions hold: it has accepted, it is unopened, its partner (if it has one) is unopened, and hosts i and j are unopened.

Note that freshness can easily be determined from the transcript.

At the very end of the interaction, the adversary either simply stops, or *selects* a process holding a fresh session key; in the former case, the adversary scores 0; in the latter case, the adversary is given a random string and the session key of that process, in a random order, and must guess which is which; it scores 1 if correct and -1 if incorrect. The *advantage* of the adversary is defined as the absolute value of the expected value (over the entire experiment) of its score.

Finally, a session key distribution protocol is called *secure* if the following hold:

S1 for any adversary, for any pair of processes $\Pi(i; j, u)$ and $\Pi(j; i, v)$, if the adversary faithfully transmits messages between the two processes and the server, the two processes accept and share the same session key;

S2 for every adversary, its advantage is negligible (as a function of the security parameter k).

2.2 Protocol SK1

We now describe a new protocol, which we call *SK1*, for session key distribution in the third-party model, and prove its security in the Bellare-Rogaway security model. This protocol is a modification and extension of one proposed by Leighton & Micali [3] before the Bellare-Rogaway security model was proposed, and indeed, no formal statements about security are claimed or proved in that paper. We found that several modifications to the protocol were necessary to obtain our proof of security, even though it is not clear that without these modifications the protocol is insecure.

We assume that we have a PRF generator f that for security parameter k takes a k-bit secret key, produces a k-bit output, and allows inputs of up to $2k+1$ bits. For a key K and input x, we write $f_K(x)$ for the value of the PRF.

We assume that we have n hosts. At system initialization time, three random keys K, K', and K'' of length k are chosen as the secret keys of the server S.

We introduce some notation. For $1 \leq i, j \leq n$, let $K(i) = f_K(i)$ and $K(i,j) = f_{K(i)}(j)$. Similar notations are introduced for K' and K''. We define $P(i,j) = K(j,i) \oplus K'(i,j)$ and $A(i,j) = f_{K''(i)}(j \cdot P(i,j))$. Finally, we write $r \leftarrow R_k$ to denote assignment of a random k-bit string to r.

The secret key stored in host i consists of the triple $(K(i), K'(i), K''(i))$.

The protocol for a process A on host i to establish a session key with process B on host j runs as follows. We assume A initiates the protocol, calling A the *initiator* and B the *responder*. When a process accepts, it assigns the value of the session key to the variable ω.

Step 1a A sends (i,j) to S.

Step 1b A sets $r \leftarrow R_k$ and sends r to B.

Step 2a S receives (i,j) from A and sends $\pi = P(i,j), \alpha = A(i,j)$ to A.

Step 2b B receives r from A, sets $s \leftarrow R_k$, accepts setting $\omega = f_{K(j,i)}(0 \cdot s)$, and sends $s, f_{K(j,i)}(1 \cdot r \cdot s)$ to A.

Step 3a A receives π, α from S, rejects if $\alpha \neq f_{K''(i)}(j \cdot \pi)$, and otherwise computes $\kappa = \pi \oplus K'(i,j)$.

Step 3b A receives s, β from B, rejects if $\beta \neq f_\kappa(1 \cdot r \cdot s)$, and otherwise accepts setting $\omega = f_\kappa(0 \cdot s)$.

Finally, to complete the description of the protocol, we specify a partner function. An initiator $\Pi(i;j,u)$ has as its partner $\Pi(j;i,v)$ if $\Pi(i;j,u)$ received a message s, β in step 3b, and $P(j;i,v)$ is the unique responder of the form $P(j;i,\cdot)$ that sent a message s, β' in step 2b; otherwise, $\Pi(i;j,u)$ has no partner. A responder $P(j;i,v)$ has as its partner $\Pi(i;j,u)$ if $P(j;i,v)$ received the message r in step 2b and $\Pi(i;j,u)$ is the unique initiator of the form $\Pi(i;j,\cdot)$ that sent the message r in step 1b.

Before discussing security, we note that this scheme has a couple of practical advantages over the Bellare-Rogaway protocol. First, although the number of flows in both protocols is 4, this protocol requires only 2 parallel rounds of flows, while the Bellare-Rogaway protocol requires 3. Second, and more importantly, the server need only be contacted the first time a process on host i initiates the protocol with a process on host j: the $P(i,j)$, $A(i,j)$ values can be cached on host

i; moreover, this cache can be held in insecure memory, since we can can view this cache as being maintained by the adversary, and the proof of security applies without changing the model or the protocol. If process A finds what it needs in the cache, only 2 flows and 1 round are required.

We also note that the random strings r and s could each be replaced by counters, which is actually more secure.

2.3 Security of Protocol SK1

We now prove:

Theorem 1 *If f is a secure PRF generator, then protocol* SK1 *is secure.*

Condition S1 in the definition of security is clear, so we concentrate on S2. To prove Theorem 1, we show that if the protocol is insecure, then a simple 2-process protocol is insecure, and conclude by showing that this protocol is secure.

We now define a session key distribution protocol *SK2* involving two hosts H_1 and H_2 who share a random k-bit key L. The protocol allows a process A on H_1 to establish a session key with process B on H_2. Here A is always the initiator. The protocol runs as follows.

Step 1 A sets $r \leftarrow R_k$ and sends r to B.

Step 2 B receives r from A, sets $s \leftarrow R_k$, computes the session key as $f_L(0 \cdot s)$, accepts and sends $s, f_L(1 \cdot r \cdot s)$ to A.

Step 3 A receives s, β from B, rejects if $\beta \neq f_L(1 \cdot r \cdot s)$, and otherwise accepts and computes its session key as $f_L(0 \cdot s)$.

The partner function for protocol *SK2* is defined just as for *SK1*.

Lemma 1 *Assume f is a secure PRF generator. If* SK1 *is insecure, then* SK2 *is insecure.*

Proof. Suppose we are given an adversary that has a nonnegligible advantage over *SK1*. We view the entire interaction of the adversary, the processes, and the server as a single algorithm, and we transform this algorithm in several stages into an adversary that has a nonnegligible advantage over *SK2*.

Stage 1. First, we modify the algorithm so that at the beginning of execution a pair (i_0, j_0) of hosts is chosen at random. Now the algorithm runs as before, except that we insist that the adversary select a process of the form $\Pi(i_0; j_0, \cdot)$ or $\Pi(j_0; i_0, \cdot)$, making it stop without a selection otherwise. This decreases the adversary's advantage by a factor of $n(n-1)/2$, and so it is still nonnegligible.

We make two further modifications. We stop the entire algorithm if the adversary ever opens host i_0 or j_0. This does not change the adversary's advantage at all. We also stop the algorithm if the adversary does not select an initiator on i_0 or a responder on j_0. This decreases the adversary's advantage by a factor of 2, and so it is still nonnegligible.

Stage 2. We now replace all the keys $K(i), K'(i), K''(i)$ for $1 \leq i \leq n$ by random k-bit strings. The adversary's advantage must still be nonnegligible; otherwise, we would have a statistical test for the PRF collection $\{f_K(\cdot), f_{K'}(\cdot), f_{K''}(\cdot)\}$, which is impossible by assumption.

Stage 3. Next, we modify the algorithm so that any initiator $\Pi(i_0; j, u)$ rejects if it receives a message π, α in step 3a that was not previously output by the server on input (i_0, j). The adversary's advantage must still be nonnegligible, since otherwise we would have a statistical test for the PRF $f_{K''(i_0)}(\cdot)$.

Stage 4. We now replace $K(j_0, i)$ for all i and $K'(i_0, j)$ for all j by random k-bit strings. Again, the adversary must still have a nonnegligible advantage, since otherwise we would have a statistical test for $\{f_{K(j_0)}(\cdot), f_{K'(i_0)}(\cdot)\}$.

Stage 5. We next replace $P(i_0, j_0)$ by a random k-bit string. This change does not affect the probability distribution of the adversary's view, since $P(i_0, j_0) = K(j_0, i_0) \oplus K'(i_0, j_0)$, and $K'(i_0, j_0)$ does not affect the value of any other strings seen by the adversary, except through its affect on $P(i_0, j_0)$.

Stage 6. Now, given two hosts H_1 and H_2 taking part in protocol *SK2*, using the unknown but randomly chosen key L, we let the processes on H_1 play the role of initiator processes on host i_0, and the processes on H_2 play the role of responders on j_0. The key L takes the place of the key $K(j_0, i_0)$. The algorithm as it now stands can be executed without actually using the value of L, except indirectly through the responses of the processes on H_1 and H_2. Thus, the algorithm is an adversary with nonnegligible advantage over *SK2*. $\Box$

Lemma 2 *If f is a secure PRF generator, then protocol* SK2 *is secure.*

Proof. We argue by replacing $f_L(\cdot)$ by a truly random function F in the protocol, showing that this modified protocol is secure. If the original protocol were insecure, this would give us a statistical test for $f_L(\cdot)$.

It is straightforward to see that with overwhelming probability, the following events occur:

(i) all r-outputs by H_1-processes are distinct,

(ii) all s-outputs by H_2-processes are distinct, and

(iii) if any process A on H_1 accepts, and A output r and received s, then there is a process B on H_2 that received r and output s.

Conditioning on these events, it is easy to see that the adversary's advantage is 0. To see this, suppose the adversary were to select a process B on H_2 that holds $F(0 \cdot s)$. Then, by (ii), no other H_2-process holds $F(0 \cdot s)$, and by (i) and (iii), any H_1-process that holds $F(0 \cdot s)$ must be B's partner. Likewise, if the adversary were to select a process A on H_1 that holds $F(0 \cdot s)$, then by (i), (ii), and (iii), there are no other H_1-processes that hold $F(0 \cdot s)$, and by (ii) there is exactly one H_2-process that holds $F(0 \cdot s)$, which is A's partner. Thus, the adversary did not open any process holding $F(0 \cdot s)$, and hence has advantage 0. $\Box$

One final remark about the security of Protocol SK1 is in order. Note that process B essentially accepts unconditionally, without challenging A at all. Thus, an adversary could make B accept without any legitimate process A being involved at all. However, if the adversary does this, the adversary will have no idea what B's session key is, and so it is not clear why an adversary would do this. This point illustrates a difference between session-key distribution and entity authentication: usually, two parties engaging a session key protocol are not so much interested in verifying that they are really talking to each other *at that moment*, but rather, they want to be able to authenticate and/or encrypt messages that are sent after the protocol has terminated. Recognizing these two security goals as distinct is one of the contributions of the Bellare-Rogaway model. Of course, Protocol SK1 could easily be modified to provide mutual authentication as well, if that were desired.

3 Using Smart Cards

3.1 A New Security Model

We now define what we mean by security for a smart card based protocol. Our starting point is the model in §2.1, and we focus on the differences. As indicated in the introduction, secret keys will be stored in the smart cards, and the smart cards will then be used in a stateless fashion, as probabilistic oracles, to compute session keys which are then stored in the private memory of a process on its host.

System initialization is as before, except that the secret keys are stored in the smart cards, not the hosts.

The adversary may obtain question/answer pairs as before. However, the two *open* operations are defined differently, and there is a new operation: *access*.

First, the adversary may *open* any process at any time, even if it has not accepted; upon receiving an *open* request, the process must reveal a complete description of the conversation it has carried out with its smart card, (and the outcomes of its coin-tosses if the process is probabilistic). This type of attack is at least as powerful as reading the private memory of a process (which is presumably normally protected by operating system security defenses), including its session key, as well as obtaining the session key via cryptanalysis or other means.

Second, the adversary may *open* a host's smart card at any time. When this happens, the adversary is given the secret keys stored on the corresponding smart card. In practice, this type of attack will presumably be difficult to mount, as it entails physically obtaining the smart card, and somehow breaking the card's physical security defenses.

Third, the adversary may *access* a host's smart card at any time. That is, the adversary may ask any question of the smart card that conforms to the smart card's functional interface, and receive the corresponding answer. This type of attack subsumes several others, including a network break-in or virus, replacement of the host's software by rogue software, or physically stealing the card.

The transcript, view, and partner function are defined as before, except that the exact forms of the questions and answers are defined as above.

We now define the freshness of session keys to capture the intuition outlined in §1. A process $\Pi(i; j, u)$ holds a *fresh* session key if the following conditions hold: it has accepted, it is unopened, its partner (if it has one) is unopened, the smart cards on hosts i and j are unopened, and j's smart card has not been accessed between the times of $\Pi(i; j, u)$'s first and last questions.

Again, note that freshness can be determined from the transcript.

The protocol is said to be *secure* if condition S1 and S2 hold, exactly as before.

3.2 Protocol SK3

We now describe our smart card based protocol, *SK3*. This protocol is derived from protocol *SK1*.

Secret keys are generated just as for SK1, except that the keys are stored in the smart cards. Also, the smart card of host i is given a random k-bit string $T(i)$. We adopt all of the notation from §2.2, i.e., $K(i)$, $A(i,j)$, etc.

There are really three protocols here: a server interface, a smart card interface, and a process-to-process protocol.

The server interface is exactly as in *SK1*.

There are four types of queries that can be made of all smart cards, where we write $C_i(\cdot)$ for a query to i's smart card. For clarity, we sometime use host i and sometimes j.

- $C_i(1) = (r, f_{T(i)}(r))$, where $r \leftarrow R_k$.
- $C_j(2, i, r) = (s, \beta, \omega)$, where $s \leftarrow R_k$, $\beta = f_{K(j,i)}(1 \cdot r \cdot s)$, and $\omega = f_{K(j,i)}(00 \cdot s)$.
- $C_i(3, j, r, s, \pi, \alpha, \beta, \gamma) = ()$ or (δ, ω). This is computed as follows. Check if $f_{T(i)}(r) = \gamma$ and $f_{K''(i)}(j \cdot \pi) = \alpha$. If not, output $()$. Otherwise, set $\kappa = \pi \oplus K'(i,j)$. Check if $f_\kappa(1 \cdot r \cdot s) = \beta$. If not, output $()$. Otherwise, set $\delta = f_\kappa(01 \cdot s)$ and $\omega = f_\kappa(00 \cdot s)$ and output (δ, ω).
- $C_j(4, i, s, \delta) = 1$ if $f_{K(j,i)}(01 \cdot s) = \delta$, and 0 otherwise.

Finally, the process-to-process protocol is as follows. We assume process A is an initiator on host i, and process B is a responder on host j. Upon acceptance, a process assigns the session key to the variable ω.

Step 1a A sends (i, j) to S.

Step 1b A sets $(r, \gamma) = C_i(1)$ and sends r to B.

Step 2a S receives (i, j) from A and sends $P(i,j), A(i,j)$ to A.

Step 2b B receives r from A, sets $(s, \beta, \omega) = C_j(2, i, r)$, and sends s, β to A.

Step 3 A receives π, α from S and s, β from B. A computes $C_i(3, j, r, s, \pi, \alpha, \beta, \gamma)$. If this is $()$, A rejects, otherwise A accepts, assigns this value to (δ, ω), and sends δ to B.

Step 4 B receives δ from A, and accepts if $C_j(4, i, s, \delta) = 1$, and rejects otherwise.

The partner function for this protocol is defined exactly as in protocol *SK1*.

Before proving the security of Protocol SK3, we discuss its practicality. Although this protocol may at first look complicated, the smart card itself must do relatively little computation or book-keeping. In fact, we are currently working on a prototype implementation, using a very typical, inexpensive smart card, and we comment on our preliminary design.

First, we note that as in Protocol SK1, we can replace the random strings r and s by counters. This violates our "stateless" requirement, but in practice one usually needs to maintain state anyway to generate pseudo-random numbers. On a smart card, there is usually some writable, non-volatile memory, and we only need a single, say 8 byte, counter per smart card.

Second, we can assume that host names are actually hash values, so we can assume all host names are 16 bytes if, say, MD5 is used as the hash function.

Third, we might use keyed MD5 or SHA for the pseudo-random function, perhaps truncating its output to 8 bytes, and using 8-byte secret keys. SHA implementations are already available on many inexpensive smart cards.

The most complicated query that the smart card must process is the type 3 query. The bulk of the time will be spent performing the pseudo-random function evaluations, and moving data in and out of the card. The card must perform 6 pseudo-random function evaluations, input about 64 bytes of data, and output about 16 bytes of data. This is well within the capabilities of many commonly available, inexpensive smart cards.

3.3 Security of Protocol SK3

We now prove:

Theorem 2 *Assuming f is a secure PRF generator, then protocol* SK3 *is secure.*

Our proof of this theorem follows the same strategy as that of Theorem 1: we show that if SK3 were insecure, then a certain two-host smart card based protocol, *SK4*, would also be insecure. Then we prove that *SK4* is secure.

Protocol *SK4* involves two hosts, H_1 and H_2. Host H_1 has a smart card C_1 with secret keys L and T. Host H_2 has a smart card C_2 with the same secret key L. Card C_1 responds to two types of queries (1 and 3) and C_2 responds to two types of queries (2 and 4), as follows.

- $C_1(1) = (r, f_T(r))$, where $r \leftarrow R_k$.
- $C_2(2, r) = (s, \beta, \omega)$, where $s \leftarrow R_k$, $\beta = f_L(1 \cdot r \cdot s)$, and $\omega = f_L(00 \cdot s)$.
- $C_1(3, r, s, \beta, \gamma) = ()$ or (δ, ω). This is computed as follows. Check if $f_T(r) = \gamma$ and $f_L(1 \cdot r \cdot s) = \beta$. If not, output $()$. Otherwise, set $\delta = f_L(01 \cdot s)$ and $\omega = f_L(00 \cdot s)$ and output (δ, ω).
- $C_2(4, i, s, \delta) = 1$ if $f_L(01 \cdot s) = \delta$, and 0 otherwise.

The process-to-process protocol for *SK4* is as follows. We let A denote a process on H_1, which is always an initiator, and B denote a process on H_2, which is always a responder. Upon acceptance, the session key is stored in ω.

Step 1 A sets $(r, \gamma) = C_1(1)$ and sends r to B.

Step 2 B receives r from A, sets $(s, \beta, \omega) = C_2(2, i, r)$, and sends s, β to A.

Step 3 A receives s, β from B. A computes $C_1(3, r, s, \beta, \gamma)$. If this is $()$, A rejects, otherwise A accepts, assigns this value to (δ, ω), and sends δ to B.

Step 4 B receives δ from A, and accepts if $C_2(4, s, \delta) = 1$, and rejects otherwise.

The definition of the partner function is the same as above.

Lemma 3 *Assume f is secure PRF generator. Then if* SK3 *is insecure, then* SK4 *is also insecure.*

Proof. The proof of this is almost identical to that of Lemma 1, and so we omit it. □

Lemma 4 *Assume f is a secure PRF generator. Then protocol* SK4 *is secure.*

Proof. Again, we prove this by replacing f_L by a random function F and f_T by a random function G.

Now, an adversary can access a smart card directly, or indirectly via a process. Regardless of how these accesses occur, it is straightforward (if tedious) to see that with overwhelming probability, the following events occur:

(i) all r-outputs from type 1 queries are distinct,

(ii) all s-outputs from type 2 queries are distinct,

(iii) for any type 3 query that accepts with inputs r, s at time τ_3, r was output by a type 1 query at time τ_1 and s was output by a type 2 query with input r at time τ_2, where $\tau_1 < \tau_2 < \tau_3$,

(iv) for any type 4 query that accepts with input s at time τ_4, there is a type 3 query that accepted with inputs r, s for some r at time $\tau_3 < \tau_4$, and

(v) for any type accepting type 3 query with inputs r, s made by an unopened process, there is no other accepting type 3 query with inputs r', s for any r'.

Conditioning on these events, the adversary's advantage is 0. To see this, suppose an adversary were to select a process B on H_2 holding the fresh session key $F(00 \cdot s)$. Then B made a type 4 query at some time τ_4, and by (iii) and (iv), certain type 1, 2, and 3 queries were made, as described above, at times $\tau_1 < \tau_2 < \tau_3 < \tau_4$. By the freshness of B's key, the type 3 query was made by some process A on H_1, and was not made directly by the adversary. By (i), the type 1 query was also made by A, and by (ii), the type 2 query was made by B. It follows that A is B's partner, and hence could not have been opened. From (ii) and (v), it then follows that $F(00 \cdot s)$ was never revealed to the adversary.

Suppose the adversary were to select a process A on H_1 holding the fresh session key $F(00 \cdot s)$. Then A made a type 3 query at some time τ_3, and by (iii), certain type 1 and 2 queries were made at times $\tau_1 < \tau_2 < \tau_3$. By the freshness of A's key, the type 2 query was made by some process B on H_2, and not directly by the adversary. By (ii), B is A's partner, and so could not have been opened. From (ii) and (v), it follows that $F(00 \cdot s)$ was never revealed to the adversary. □

We remark that in Protocol SK3, unlike Protocol SK1, B must challenge A in order to meet our definition of security in the smart card setting. To see why this is so, suppose an adversary can access host i's smart card twice over some long period of time. Then at any time in between, the adversary could get B to accept a session key and encrypt some files, and with the second access, the adversary could get the session key, and decrypt the files. This is precisely the kind of attack we wish to foil.

References

[1] 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.

[2] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33:210–217, 1986.

[3] T. Leighton and S. Micali. Secret-key agreement without public-key cryptography. In *Advances in Cryptology–Crypto '93*, pages 456–479, 1993.

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