Environmental Requirements for Authentication Protocols

Ran Canetti[1] and Catherine Meadows[2] and Paul Syverson[2]

[1] IBM T.J. Watson Research Center, POB 704
Yorktown Heights, NY 10598
canetti@watson.ibm.com
[2] Naval Research Laboratory
Center for High Assurance Computer Systems
Washington, DC 20375
{meadows, syverson}@itd.nrl.navy.mil

Abstract. Most work on requirements in the area of authentication protocols has concentrated on identifying requirements for the protocol without much consideration of context. Little work has concentrated on assumptions about the environment, for example, the applications that make use of authenticated keys. We will show in this paper how the interaction between a protocol and its environment can have a major effect on a protocol. Specifically we will demonstrate a number of attacks on published and/or widely used protocols that are not feasible against the protocol running in isolation (even with multiple runs) but become feasible in some application environments. We will also discuss the tradeoff between putting constraints on a protocol and putting constraints on the environment in which it operates.

1 Introduction

Much work has been done on ensuring that cryptographic protocols satisfy their security requirements. This work usually assumes a very pessimistic model; it is assumed that the protocol is running in a network controlled by a hostile intruder who can read, alter, and delete messages, and moreover that some legitimate users of the network may themselves be in league with the intruder, who as a result will have access to a certain number of keys and cryptographic algorithms. As might be expected, it is difficult to ensure that a protocol attains its security goals in such an environment, and there are a number of cases in which security flaws have been found in protocols well after they were published or even fielded. However, a number of people have developed formal requirements, that if satisfied, are intended to guarantee that the protocol operates correctly, as well as formal techniques for proving that these requirements hold. Usually, these requirements boil down to requiring that secrets (e.g. cryptographic keys) not be revealed and that, if one side or the other thinks that the protocol completed, then it should not have been interfered with in any “meaningful” way.

However, most of these formal requirements, as well as the tools for proving them correct, suffer from a limitation. They are generally used to reason about a single set of message flows running in isolation, and usually do not take into account the applications to which the protocol is put, the mechanisms of which the protocol is making use, and any similar protocols with which the protocol may be interacting. But all of these can have a major impact on the security of the protocol.

This paper demonstrates, via some attacks on well-established protocols, the dangers of ignoring the environment (which includes all other protocols that may be running concurrently with the protocol in question) when analyzing the security of authentication and key-exchange protocols. By "environment" we mean the combination of four factors, which we can think of as existing above, below, and on the same level as the protocol, respectively:

1. the intentions and capabilities of an attacker;
2. the applications using the protocol;
3. the functions (e.g. encryption, generation of random numbers, etc.) on which the protocol depends, and;
4. other similar protocol executions with which the current execution may or may not interact.

It is known that, for any cryptographic protocol, it is possible to construct another protocol with which it interacts insecurely [18], given that the two protocols share some secret information. Such general "attack protocols" are not always realistic, but the authors of [18] are able to construct many examples that are, and to develop several design principles that can help to prevent these attacks. The aim of our paper is to extend this approach further by showing that even the strongest requirements on a protocol are not enough to guarantee the security of a protocol interacting with even a "reasonable" environment unless we put restrictions on the environment as well as the protocol. We will do this by presenting as examples protocols that satisfy increasingly strong requirements, each time showing how they could be subverted by seemingly benign aspects of the environment that are exploited by a hostile intruder.

We begin by describing (in Section 2) the notion of a threat environment, which comprises the assumptions made about the capabilities and goals of a possible attacker. We describe two well-known attacks on cryptographic protocols, both of which came about as the result of changing assumptions about the threat environment. We then present the current standard definition of the threat environment, and argue that it is not adequate to prove a protocol's security. In Section 3 we extend our argument by recalling two known attacks. Even though these attacks focus on a protocol run in isolation, they make our point that when analyzing one protocol (or sub-protocol) one must keep in mind the environment in which this protocol is expected to run. The first attack demonstrates that separately analyzing sub-protocols of a given protocol, without making sure that the composition of the two sub-protocols is secure, is dangerous. It also demonstrates our point that the threat environment can vary according to the protocol and its intended application. The second attack concentrates on the danger in several concurrent runs of the same protocol. We discuss requirements on the environment that could have prevented this problem, as well as the protocol requirement known as "matching histories" which requires that principals have identical message histories after a protocol completes. Then, in Section 4 we describe an attack on a protocol satisfying matching histories that demonstrates possible weaknesses that arise from bad interactions between a key exchange protocol and an "application protocol" that uses the key. This also motivates the definition of a stronger requirement, "agreement," which requires that principals not only have identical message histories but agree on who they are talking to. The necessity of protecting secrets, even malformed or immature ones, is discussed in Section 5. In this section we first show how a protocol satisfying agreement can be foiled by an intruder who sends messages that have different meanings to sender and receiver, causing the receiver to respond with an inappropriate message. We then show how a protocol that satisfies the even stronger requirement of extensible-fail-stop, which would outlaw such type of behavior, can be tricked when it interacts with a seemingly harmless challenge-response protocol. We discuss our findings in Section 6 and give some recommendations for environmental requirements, as well as a discussion of open problems.

2 Threat Environments

One of the most obvious features of an environment is the set of assumptions that are made about the capabilities of the intruder who is trying to subvert the protocol. As a matter of fact, many of the attacks that have been found on published protocols are found as a result of changing the threat environment model; a protocol that was designed to be secure against one type of threat may not be secure against another. We give here two examples, discussed by Pancho in [24].

Both of these attacks are on early protocols due to Needham and Schroeder in [23]. These protocols were some of the earliest proposed for distributing keys over a network, so it is not surprising that the threat model was still evolving at the time. In particular, Needham and Schroeder made two assumptions which are not commonly made today. The first was that old session keys would not be vulnerable to compromise, and the second was that principals would only try to communicate securely with other principals that they trusted.

Both of these protocols used an architecture that has become standard among cryptographic protocols today. In this architecture, principals possess long-term, or master, keys which are used to distribute shorter-term, or *session*, keys. The master keys may possibly be distributed by some non-electronic means, while the session keys will be distributed electronically over the network. The rationale behind this is manifold: First, in networks of any size it is infeasible for keys to be generated and assigned so that all principals share a unique key with each other principal. Second, limited use of the master keys makes them less vulnerable to compromise. Third, this limited use makes it possible to restrict more expensive forms of encryption (such as public key) to encryption with the master key and to use cheaper encryption for the session keys, which will encrypt the highest volume of traffic.

The first of these protocols involves two parties who wish to communicate with each other using a key server. Each party shares a pairwise key with the server, and the protocol proceeds as follows:

Message 1 $A \to S$: A, B, R_A

A requests a key from the server S to communicate with B. It includes a random nonce R_A so that it can verify the recency of any message it gets from the server. If it receives an encrypted message from the server containing R_A, it will know that the message could only have been generated after the server saw R_A.

Message 2 $S \to A$: $\{R_A, B, k_{AB}, \{k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}$

The server creates a key k_{AB}. It encrypts the key together with A's name using the key B shares with S. Then it uses k_{AS}, the key S shares with A, to encrypt R_A, B's name, the new key k_{AB}, and the encrypted message for B.

Message 3 $A \to B$: $\{k_{AB}, A\}_{k_{BS}}$

After A receives and decrypts the message from S, it forwards the key to B.

Message 4 $B \to A$: $\{R_B\}_{k_{AB}}$

B can verify the the key is from the server and is intended for communication with A, but it still needs to verify that the message is not a replay. It does this by creating its own nonce, encrypting it with k_{AB}, and sending it to A.

Message 5 $A \to B$: $\{R_B - 1\}_{k_{AB}}$

A decrypts the nonce, subtracts one, reencrypts it, and sends it to B. When B receives the message and decrypts it, it can verify that it can only have come from someone who knows k_{AB}. Since only A (and the server) is assumed to know k_{AB}, and the server is assumed to be honest, B can assume both that the key is recent and that it is talking to A.

Denning and Sacco found the following attack, based on the assumption that old session keys can be compromised. In this attack, the intruder not only compromises the old session key, but has been able to record the communication by which the old session key was distributed:

Message 3 $\quad I_A \to B: \quad \{k_{AB}, A\}_{k_{BS}}$

An intruder, pretending to be A, resends an old encrypted message containing an old compromised key for A and B.

Message 4 $\quad B \to A: \quad \{R_B\}_{k_{AB}}$

B responds according to the rules of the protocol.

Message 5 $\quad I_A \to B: \quad \{R_B - 1\}_{k_{AB}}$

The intruder intercepts B's message to A. Since it knows the old key, it can decrypt B's message and produce the appropriate response.

The other protocol involves two principals who use public key encryption to set up a session with each other. This protocol has two parts, one in which the principals obtain each other's public keys from a server, and one in which they use the public keys to set up a session with each other. Since the attack does not involve the first part, we present only the second part here:

Message 1 $\quad A \to B \ : \ \{R_A, A\}_{k_B}$

A sends a nonce R_A, together with A's name to B, encrypted with B's public key. B decrypts to get A's name.

Message 2 $\quad B \to A \ : \ \{R_A, R_B\}_{k_A}$

B generates a nonce R_B and sends it together with R_A to A, encrypted with k_A. A decrypts the message. If it finds R_A, it assumes that this is a message from B in response to its original message.

Message 3 $\quad A \to B \ : \ \{R_B\}_{k_B}$

A encrypts R_B with k_B and sends it to B. B decrypts the message. If it finds R_B, it assumes that this is a message from A in response to its original message.

The attack, as presented by Lowe in [20], proceeds as follows. Note that this attack only works if one of the principals attempts to communicate with an untrusted principal first:

Message 1 $\quad A \to I: \quad \{R_A, A\}_{k_I}$

A initiates communication with I.

Message 1′ $\quad I_A \to B: \quad \{R_A, A\}_{k_B}$

I initiates communication with B, using R_A.

Message 2′ $\quad B \to A: \quad \{R_A, R_B\}_{k_A}$

B responds to A. A decrypts and finds R_A.

Message 3 $A \to I$: $\{R_B\}_{k_I}$

Thinking that the previous message is a response from I, A responds in kind. I decrypts R_B and can now use it to impersonate A to B.

Message 3′ $I_A \to B$: $\{R_B\}_{k_B}$

I completes the protocol with B.

Over the years, the community has come to agree on a standard intruder model which has come to be so widely accepted that it is sometimes assumed that Needham and Schroeder were attempting to design their protocols according to it but failed (see Pancho [24] for a discussion of this). In this model we assume that the network is under the control of a hostile intruder who can read, modify, and delete message traffic, and who may be in league with corrupt principals in the network. We assume that honest principals have no way of telling corrupt principals from other honest principals, and thus may be willing to initiate communication with these dishonest principals. Finally, we assume that old session keys may be compromised, although we usually do not assume that the master key that is used to distribute the session key will be compromised.

This is accepted as the standard intruder model, but is it the last word? We argue that it is not.

First of all, threat models can change. For example, the standard threat model alluded to above takes into account the compromise of old session keys, but does not concern itself with the threat of session keys being compromised while the session is still current. In the next section we will discuss a protocol in which the real-time compromise of session keys is a realistic threat.

Secondly, the model does not take into account other, possibly benign aspects of the environment which may affect the security of the protocol. These include other, similar, protocols that the protocol could be tricked into interacting with, as well as applications that make use of the protocol in ways not anticipated by the designer. We will discuss these aspects of the environment in the remainder of the paper.

3 Protocol Composability and Matching Histories

A first example that illustrates the danger of separately analyzing sub-protocols of a larger protocol is taken from [22] which in turn discusses an attack, first described in [5], on a very early version of SSL. SSL negotiates a key between a client and a server. The early version included an optional client authentication phase, achieved via a digital signature on the key and a nonce provided by the server, in which the client's challenge response was independent of the type of cipher negotiated for the session, and also of whether or not the authentication was being performed for a reconnection of an old session or for a new one. Instead, this information was authenticated by the key that was negotiated between the client and the server. Moreover, this version of SSL allowed the use of cryptographic algorithms of various strength (weak algorithms for export and stronger ones for domestic use), and it was not always clear by inspection of the key whether weak or strong cryptography was being used. This allowed the following attack (note that in this version of SSL, session keys were supplied by the client):

1. A key k is agreed upon for session A using weak cryptography.
2. Key k is broken by the intruder in real time.
3. The client initiates a reconnection of session A.
4. The intruder initiates a new session B, pretending to be the client, using strong cryptography together with the compromised key k.
5. As part of the connection negotiations for session B, the server presents a challenge to the client. The client should return a digital signature of both k and the challenge. The intruder can't do this itself, but it can pass the server's request on to the client, who will take it to be part of the reconnection negotiations for session A, and produce the appropriate response. The intruder passes the response on to the server as part of the session B negotiations, and the protocol completes.
6. If the client would have been given access to special privileges as a result of using strong cryptography, this could lead to the intruder gaining privileges that it should not be able to have by breaking the key k.

This attack involves two environmental features. One involves a confusion of the reconnection protocol with the connection protocol. Thus, it is an example of a failure of composition which would not have been found if the two protocols had been analyzed separately. The second involves the use of strong and weak cryptography. Authentication protocol analysis often assumes that cryptography is "perfect" (i.e., modeled as a black box) and that keys are not broken directly. However, this protocol explicitly distinguishes between strong and weak cryptography, thus explicitly addresses relative invulnerability to key compromise. Although vulnerability of later authentications because of earlier session key compromise has long been recognized as a problem protocols should avoid [12], the use of weak cryptography also raises the possibility of a compromise *during* a session, which can then be used to attack other, concurrent sessions.

The attack is a failure of an authentication requirement called *matching conversations* [4] or *matching histories* [6, 13]. We cite the definition from [13]: "in all cases where one party, say Alice, executes the protocol faithfully and accepts the identity of another party: at the time that Alice accepts the other party's identity (before she sends or receives a subsequent message), the other party's record of the partial or full run matches Alice's record." Specifically the (composed) protocol fails to satisfy matching histories because the client's record of the protocol shows a request to use weak crypto via a reconnection, but the server's record shows a request to use strong crypto via a new connection.

Later versions of SSL fixed the above problem by including signed hashes of all messages previously sent in a given protocol round. This prevents an attack on the histories of the runs. Note the subtlety here. If the hashes were authenticated by using the session key, then the protocol would appear to satisfy matching histories. However, an adversary strong enough to break k under weak crypto could then spoof an authentication of k for the client using strong crypto, thus violating matching histories.

A protocol that was designed to meet the matching histories requirement was given in [13]. In this station-to-station (STS) protocol, a publicly known appropriate prime p and primitive element α in $GF(p)$ are fixed for use in Diffie-Hellman key exchange. It is not necessary to understand the underlying mathematics to follow the protocol. All that is necessary is to know the following: A has a public exchange key, R_A, and a private exchange key, x, while B has public exchange key R_B, and private exchange key y. They can form a session key, k_{AB}, because the public-private key pairs are chosen so that ${R_A}^y = {R_B}^x = k_{AB}$. (All arithmetic is *modulo p*.) Parties A and B use a common signature scheme: $s_U[\bullet]$ indicates the signature on the specified argument using the private signature key of party U. $\{\bullet\}_k$ indicates the symmetric encryption of the specified argument under key k. Public key certificates are used to make the public signature keys of A and B available to each other. In a one-time process prior to the exchange between A and B, each party must present to a trusted certificate server, T, her true identity and public key (e.g., ID_A, k_A), have T verify the true identity by some (typically non-cryptographic) means, and then obtain from T her own certificate. The protocol is as follows. (Here the public parameters are a finite group of large prime order, and a generator, α, of this group. All exponentiations are done in the group arithmetic.)

1. A generates a random positive integer x, computes $R_A = \alpha^x$ and sends R_A to a second party, B.
2. Upon receiving R_A, B generates a random positive integer y, computes $R_B = \alpha^y$ and $k_{AB} = (R_A)^y$.
3. B computes the authentication signature $s_B[R_B, R_A]$ and sends to A the encrypted signature $\text{Token}_{BA} = \{s_B[R_B, R_A]\}_{k_{AB}}$ along with R_B and his certificate Cert_B. Here ‘,’ denotes concatenation.
4. A receives these values and from R_B computes $k_{AB} = (R_B)^x$.
5. A verifies the validity of B's certificate by verifying the signature thereon using the public signature-verification key of the trusted authority. If the certificate is valid, A extracts B's public verification key, k_B from Cert_B.
6. A verifies the authentication signature of B by decrypting Token_{BA}, and using k_B to check that the signature on the decrypted token is valid for the known ordered pair R_B, R_A.
7. A computes $s_A[R_A, R_B]$ and sends to B her certificate Cert_A and $\text{Token}_{AB} = \{s_A[R_A, R_B]\}_{k_{AB}}$.
8. A sets k_{AB} to be the shared key with B in this exchange.
9. Analogously, B checks Cert_A. If valid, B extracts A's public verification key k_A and proceeds.
10. Analogously, B verifies the authentication signature of A by decrypting Token_{AB}, and checking the signature on it using k_A and knowledge of the expected pair of data R_A, R_B.

11. Analogously, B sets k_{AB} to be the shared key with A in this exchange.

Lowe argued in [21] that this protocol is subject to attack. Specifically:

Message 1 $\quad A \to C_B: \; R_A$

Message 1′ $\quad C \to B: \; R_A$
Message 2′ $\quad B \to C: \; R_B, \{s_B[R_B, R_A]\}_{k_{AB}}$

Message 2 $\quad C_B \to A: \; R_B, \{s_B[R_B, R_A]\}_{k_{AB}}$
Message 3 $\quad A \to C_B: \; \{s_A[R_A, R_B]\}_{k_{AB}}$

Here, messages 1, 2, and 3 are a protocol run that Alice attempts to run with Bob, but that Charlie attacks. Messages 1′ and 2′ are a (partial) protocol run that Charlie initiates with Bob. This is an attack according to Lowe because Alice believes (correctly) that she is running the protocol with Bob. But, Bob believes that he is running the protocol with Charlie. Lowe considers this to be an attack "since A ends up holding incorrect beliefs". For reasons such as this, Lowe considers matching histories to be an insufficiently strong requirement. He proposes a requirement he later called *agreement*: whenever an agent A completes a run of the protocol, apparently with B, then B has recently been running the protocol, apparently with A, and the two agents agree upon who initiated the run, and agree upon all data values used in the run; further there is a one-one relationship between the runs of A and the runs of B. (If we take the one-one requirement as implicit in the matching-histories definition, then agreement is just matching histories where the protocol initiator and responder must always be specified in the record of all principals.)

Whether or not the above is correctly called an attack may be argued. The failure of agreement is clear. However, Alice ends up holding incorrect beliefs only if she forms the incorrect belief that Bob believes he has been running the protocol with Alice. There is no specific reason to attribute such a belief to Alice here, since she has received no authenticated information to that effect. Nonetheless, if we consider the protocol in composition with its environment, then an attack may be possible. We consider the effect of composing protocols with different environments in the next section.

4 Application Environments

Until now, our discussion of composition has involved either interleaved runs of a protocol with itself or of different subprotocols of a protocol. But what about other distinct protocols that may be running alongside of the one being considered? This concern may potentially be "brushed aside" by requiring that the protocol in question be the only protocol of its kind (authentication, key-exchange, etc.) to be run in the system. This way, one may not have to consider the potential interactions of different protocols because only one protocol (possibly with subprotocols) is permitted to run in a given context. (An exception to this assumption and analysis of the resulting implications was given in [18].) This may be a reasonable assumption to make in some cases. However, even if we can assume our protocols to be running in isolation we must still contend with the applications that make use of the authenticated keys that were established using the protocol in question.

Consider the following attack, described by Shoup in [25]. It involves the use of an "application protocol" that uses shared keys. Here principals authenticate by encrypting a challenge. To be a bit more vivid, consider an environment in which monitoring devices establish authenticated communications with a server. Perhaps these are used like watch keys to indicate that an actual person is present at the location of the device; someone must physically engage the device in some way for it to operate. The server might send out a nonce challenge, and the monitoring devices would then encrypt the challenge in a return message to prove that a person was present to operate the monitor. Assume that monitors initiate contact with the server. So the application would be something like.

Application Message 1 $\quad B \rightarrow A: \quad \mathit{challenge}$

Application Message 2 $\quad A \rightarrow B: \quad \{\mathit{challenge}\}_{k_{AB}}$

Now, suppose that this application protocol uses STS to establish session keys. Also suppose that monitors are not expected to recognize or test the format of the challenge in any way. This is perhaps reasonable since the challenge should be unpredictable and the authentication gives the monitor credit rather than responsibility [1]. Let us now consider the STS protocol and the putative attack given above. The attacker C has access to R_A and R_B as plaintext. Thus, once the protocol has completed with A, C_B could send to A the challenge $s_c[R_A, R_B]$. To which A would respond with $\{s_c[R_A, R_B]\}_{k_{AB}}$. That is, after the application challenge and response of

Application Message 1 $\quad C_B \rightarrow A: \quad s_c[R_A, R_B]$

Application Message 2 $\quad A \rightarrow C_B: \quad \{s_c[R_A, R_B]\}_{k_{AB}}$

the following message may be added to the Lowe attack on STS:

Message $3'$ $\quad C \rightarrow B: \quad \{s_c[R_A, R_B]\}_k$

This way, the above debatable attack by Lowe is turned into a clear attack with more significant consequences. That is, C can now use the resulting message to complete the protocol run begun with B. Consequently, from now on, anytime B issues a challenge, credit for encryption of it with k_{AB} will be given to C rather than to A. Whether or not the Lowe attack indicates an inadequacy of matching histories to capture practical authentication goals, this attack would seem to do so—in the presence of such an application.

One possible solution would be to strengthen the protocol to include the name of the intended recipient of each message within the signature. In fact, this is the revision suggested by Lowe in [21]. STS so strengthened appears to satisfy agreement.

Another possible solution is to restrict the application environment in some way. For example, in the case of the STS protocol, we could require that any protocol that makes use of a key generated by an instance of the STS protocol would need to apply some transformation to it first, such as a hash function. This is indeed what is done in Krawczyk's [19] SKEME protocol, which is based on STS. We consider the implications and limitations of environmental requirements in the next section.

5 Environmental Requirements

The application environment is obviously more difficult to control or specify than the security protocol, so we should justify the need to restrict the environment before we consider how to do so. Are there examples of attacks on protocols that satisfy agreement? One example of such was first given by Davida long ago [11] and later generalized in [17]. Although neither of the papers explicitly notes the connection, the idea relies on a concept similar to blinding in the sense of Chaum [10], who used it effectively in the design of anonymous payment protocols. A public-key encryption is blinded so that the intended recipient gets unrecognizable garbage upon decrypting a message and discards the result as worthless. If the attacker is able to obtain this discarded message, he can then apply the blinding factor to obtain the plaintext.

The attack applies to the RSA cryptosystem, which encrypts a message m by raising it to a public exponent e modulo a public modulus $N = p \cdot q$, where p and q are two primes. Only someone who knows the secret factorization is able to compute d so that $m^{d \cdot e} = m \bmod N$, and so decrypt the message.

The attacker A proceeds by finding an encrypted message, $m^e \bmod N$, that may have previously been sent to a principal B using B's public key e and N. The attacker first computes $x^e \bmod N$ for some x and proceeds as follows:

$A \rightarrow B: \quad m^e \cdot x^e = (m \cdot x)^e \bmod N$

B receives the message and decrypts it to obtain $m \cdot x$. Since this looks like garbage to B, he discards it. If the message is discarded in such a way that the attacker A can find it, A can then multiply it by $x^{-1} \bmod N$ to obtain the secret m.

In one sense, this protocol satisfies agreement, since A and B agree on the messages that were sent between the two parties, and on who they were sent to. In another sense, it does not, since A and B disagree on the meaning of the messages sent. Since B "responds" to a nonsense message by putting it in a place where A can find it, this semantic form of disagreement can have serious consequences.

We can prevent problems like this by requiring that people secure their garbage bins—to use the Joye-Quisquater term. If we did this and required protocols to satisfy, not just agreement in Lowe's sense, but also that principals never send anything in response to an improper protocol message, it would seem that we could be free of such questions.

Such an approach was taken in [15] with the introduction of extensible-fail-stop protocols. These effectively require that agreement can be checked on each message as it is received. Active attacks on a message therefore cause any later messages not to be sent. Also, protocols that are extensible-fail-stop can be arbitrarily composed. (Similar concepts were discussed in [16].)

To illustrate, here is a variant on the Needham-Schroeder shared-key protocol presented in Section 2. It has been modified to be extensible-fail-stop.

Message 1	$A \rightarrow S:$	$(A, S, T_A, NSSK, R, 1, B),$
		$\{h(A, S, T_A, NSSK, R, 1, B)\}_{k_{AS}}$
Message 2	$S \rightarrow A:$	$(S, A, NSSK, R, 2),$
		$\{h(S, A, NSSK, R, 2),$
		$(k_{AB}, B, \{h(S, B, T_s, NSSK, R, 3), (k_{AB}, A)\}_{k_{BS}})\}_{k_{AS}}$
Message 3	$A \rightarrow B:$	$(S, B, T_s, NSSK, R, 3),$
		$\{h(S, B, T_s, NSSK, R, 3), (k_{AB}, A)\}_{k_{BS}}$
Message 4	$B \rightarrow A:$	$(B, A, NSSK, R, 4),$
		$\{h(B, A, NSSK, R, 4), R_B\}_{k_{AB}}$
Message 5	$A \rightarrow B:$	$(A, B, NSSK, R, 5),$
		$\{h(A, B, NSSK, R, 5), R_B\}_{k_{AB}}$

For each message, a hash of the message parameters is encrypted together with message data using a key shared between the sender and receiver. Parameters indicate the sender and receiver, possibly a timestamp, a protocol identifier (we assume for convenience that there is only one version of $NSSK$), a unique and unpredictably generated protocol round identifier, R, a message sequence identifier, and relevant data. Thus in the first message, Alice tells the server that she would like to establish a session key with Bob. The server is assumed to keep a list of previously used round identifiers within the lifetime of the timestamp, T_A and to check that R is not on that list. It should be clear that this protocol is extensible-fail-stop, i.e., that the recipient of each message can completely determine that s/he is receiving the next appropriate message in the protocol. Thus, there is no possibility either of using this protocol as an oracle to generate messages or of inserting a message from another protocol (or from an earlier part of the same protocol) and having it accepted as legitimate. This is therefore about as strong a requirement as one could impose on a protocol itself. Nonetheless, if the application environment is not also restricted in some way, the protocol is subject to attack.

Suppose that an application allows "eager" use of keys before the protocol has been completed. Only if the authentication protocol does not complete within some reasonable timeout is there an alarm or noting of anomaly in the logs. This eagerness might be all the more reasonable if the protocol distributing the keys is extensible-fail-stop and as explicit as this one. In this case, there would seem to be no possibility of mistake about who the session key is for, who the relevant principals are, or the roles they each play (i.e., initiator or responder). But, allowing eager use of keys in an application such as the monitor example described above could be used to attack the protocol.

Specifically, when Alice begins NSSK for a session with Bob, the attacker prevents the third message from arriving. Then, for the application challenge-response he produces:

Application Message 1	$C_B \to A:$	$h(B, A, NSSK, R, 4), R_B$
Application Message 2	$A \to C_B:$	$\{h(B, A, NSSK, R, 4), R_B\}_{k_{AB}}$

The attacker uses the response from Alice for the fourth message in NSSK, and intercepts the final message from Alice to Bob. Alice will now be spoofed into thinking she has completed a handshake with Bob when Bob was never present, with all the potential implications previously discussed.

Note that the original Needham-Schroeder shared-key protocol is just as vulnerable to this attack (and others as well). We have strengthened it to be extensible-fail-stop to show that even such a strong requirement on protocol authentication may not be adequate to preclude failure if the environment is not somehow restricted.

6 Discussion

We have given a set of increasingly stringent definitions of security and shown how they can be subverted by interaction with a carelessly designed environment. The obvious question arises: are there conditions that we can put on an environment so that we can guarantee that it will not subvert the goals of a reasonably well-behaved protocol? Some rules of thumb, in the spirit of Abadi and Needham [2], Anderson and Needham [3], and Kelsey and Schneier [18], and expanding on those we presented in [9], are suggested by the examples we have cited.

RoTh 1. Know your threat environment, and the threat environment for which your protocol was designed. If a protocol is being transferred from one threat environment to another, make sure that it is secure in the new environment as well as the old one.

As we saw from our discussion of the Needham-Schroeder protocols threat environments can change, and a protocol which was secure with respect to one set of threats may be insecure with respect to another. And, as we saw from our discussion of the SSL protocol, there is no such thing as a “one-size-fits-all” threat environment. Threats such as the real-time compromise of keys which may be silly under one set of assumptions may be realistic under another set.

RoTh 2. The environment should not use protocol keys or other secrets in unaltered form.

Thus, in the attack on STS above, the session key k_{AB} should not be the same key as the key k used internally in the STS protocol. In fact, the two keys should be "cryptographically independent". For instance, let $k_{sts} = PRF_k(0)$ and $k_{AB} = PRF(1)$, where PRF is a pseudorandom function. Now, use k_{sts} to encrypt the STS messages, and use k_{AB} as the session key. We remark that this technique for guaranteeing the "virginity" of the session key is used in SKEME [19] and in IKE, the Internet Key Exchange protocol (described in [14]). The concept underlying this principle has not only been used in the design of protocols such as IKE and SKEME but also in proving theoretical results about protocol composition [27, 26]. It is also similar in spirit to the design principle given in [18] which recommends limiting the scope of keys; indeed, what it does is give a practical means of accomplishing just that. More recent theoretical work along these lines can be found in [8], which proposes a specification for key-exchange protocols, and demonstrates that a key exchanged using a protocol that satisfies the specification can be used within standard shared-key "secure channel" mechanisms, without fear of bad interferences between protocols. Further, [7] provides a general framework for writing specifications for protocols, with an attempt to guarantee that security is maintained in a large set of computational environments.

RoTh 3. A protocol should be designed as much as possible to be resilient against the release of secrets, even potential or obsolete secrets.

We note that the protocol used in the blinding attack failed to satisfy this requirement, as well as the fail-stop version of NSSK, although in the latter case the requirement could have been satisfied with the use of RoTh 2.

RoTh 4. Values established during a protocol run should not be used in applications by a principal before that principal completes his run.

This one might seem obvious, but as we saw above, such eager use might involve a key that has been released at most to valid principals, and so appear harmless. Also, this might be an onerous requirement if applications have tight real-time constraints that would be hard to meet if they must wait for the authentication protocol to finish. In that case, we may want to make use of some of the other Rules of Thumb such as RoTh 2 or RoTh 3 to guarantee security.

RoTh 5. Insist, as much as possible, on only interacting with protocols whose messages contain unique protocol identifiers. More generally, try to make certain that a message intended for use in one protocol, or protocol component, can not be mistaken for a message for use in one protocol, or protocol component.

This use of unique protocol identifiers is recommended in [18] and has also been used in the design of extensible-fail-stop protocols. We note that, in the attack on the extensible-fail-stop protocol in Section 5, if both protocols involved had followed this principle instead of just one, then the attack would not have been possible. Again, this may be onerous, e.g., if the environment requires the use of off-the-shelf applications that are not extensible-fail-stop. This leads us to

RoTh 6. Any fix to the environment should be made as close to the protocol as possible.

To illustrate what we mean by this, we consider the use of challenge-and-response protocols to illustrate environmental attacks. All of these attacks could have been prevented if each time the principal responded to the challenge it signed, instead of the challenge itself, a hash of the challenge and some text indicating what it believed the challenge was for. Indeed, this was how the weakness in the SSL protocol was ultimately fixed. However, in the case of SSL, the challenge-and-response protocol was directly under the control of the protocol designer, since it was part of the SSL protocol suite. In our other examples this was not necessarily the case. Thus, in the case of the Station-to-Station protocol, it made more sense to require that keys distributed by the protocol be transformed before they were used. This was a feature that could be implemented, for example, by putting a wrapper around the protocol that transformed the keys, a feature that would only have to be added once for every time the protocol was implemented. Thus this was an easier requirement to satisfy than assuring that all challenge-and-response protocols that used keys generated by the protocol were implemented properly.

These observations lead to further questions: Is there a minimal set of requirements that we can put on an environment such that the various types of protocol requirements that we have described will guarantee security? If so, is it unique? If so, what is it? If not, what are the possibilities? Clearly, it is possible for any protocol to generate an environment that will subvert its security goals; the environment that releases all secrets will do the trick. But it seems reasonable to expect, that by using some combination and augmentation of the rules of thumb that we have already described, we should be able to come up with requirements for environments in which the different definitions of protocol security in isolation would also guarantee security in combination with the environment. Moreover, although in this paper we have limited ourselves to attacks involving interactions between protocol messages, we realize that the environment that a protocol depends on includes much more, including sound random or pseudo-random number generation, secure access control for keys, and the behavior and assumptions of users interacting with the system, which could also be brought into play. In summary, we believe that the study of the interaction of a security protocol with its environment, and the interaction of protocol requirements with environmental requirements is a potentially rewarding one of which we have only scratched the surface in this brief study.

7 Acknowledgements

Portions of this work originally appeared in [9]. This research was partly funded by the Office of Naval Research.

References

1. M. Abadi. Two facets of authentication. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CFW11)*, pages 25–32. IEEE Computer Society Press, June 1998.
2. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.
3. Ross Anderson and Roger Needham. Robustness principles for public key protocols. In *Proceedings of Crypto 96*, pages 236–247. Springer-Verlag, LNCS 0963, 1996.
4. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology - CRYPTO 93*. Springer-Verlag, 1994.
5. J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee. The private communication technology protocol, October 1995. draft-benaloh-pct-00.txt.
6. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In *Advances in Cryptology - Proceedings of CRYPTO 91*. Springer-Verlag, 1991.
7. R. Canetti. A unified framework for analyzing security of protocols, 2000. available at http://eprint.iacr.org/2000/067.
8. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Proceedings of Eurocrypt 01*. LNCS, May 2001.
9. R. Canetti, C. Meadows, and P. Syverson. Environmental requirements and authentication protocols. In *Symposium on Requirements Engineering for Information Security*, March 2001.
10. D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology–Proceedings of Crypto 82*, pages 199–203, 1983.
11. G. Davida. Chosen signature cryptanalysis of the RSA (MIT) public key cryptosystem. Technical Report TR-CS-82-2, Dept. of EECS, University of Wisconsin-Milwaukee, October 1982.
12. D.E.R. Denning and G.M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.
13. W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2:107–125, 1992.
14. N. Doraswamy and D. Harkins. *IPSEC: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Prentice Hall, 1999.
15. L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In R. K. Iyer, M. Morganti, W. K. Fuchs, and V. Gligor, editors, *Dependable Computing for Critical Applications 5*, pages 79–100. IEEE Computer Society Press, 1998.
16. N. Heintze and J. D. Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.
17. M. Joye and J.-J. Quisquater. On the importance of securing your bins: The garbage-man-in-the-middle attack. In *4th ACM Conference on Computer and Communications Security*, pages 135–141. ACM Press, April 1997.

18. J. Kelsey, B. Schneier, and D. Wagner. Protocol interactions and the chosen protocol attack. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *Security Protocols 1997*, volume 1361 of *LNCS*, pages 91–104. Springer-Verlag, April 1997.
19. H. Krawczyk. SKEME: A versatile secure key exchange mechanism for Internet. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security (NDSS)*, February 1996.
20. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software - Concepts and Tools*, 17:93–102, 1996.
21. G. Lowe. Some new attacks upon security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop (CSFW9)*, pages 162–169. IEEE Computer Society Press, June 1996.
22. C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *DISCEX 2000: Proceedings of the DARPA Information Survivability Conference and Exposition*, volume I, pages 237–250. IEEE Computer Society Press, January 2000.
23. R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.
24. S. Pancho. Paradigm shifts in protocol analysis: Needham and Schroeder again? In *Proceedings of the 1999 New Security Paradigms Workshop*. ACM Computer Society Press, September 1999.
25. V. Shoup. On formal models for secure key exchange (version 4). Available at `http://shoup.net/papers/`, November 1999. Revision of IBM Research Report RZ 3120 (April 1999).
26. F.J. Thayer Fábrega and J.D. Guttman. Protocol independence through disjoint encryption. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW13)*, pages 24–34. IEEE Computer Society Press, June 2000.
27. F.J. Thayer Fábrega, J.C. Herzog, and J.D. Guttman. Mixed strand spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW12)*, pages 72–82. IEEE Computer Society Press, June 1999.