

Environmental Requirements and Authentication Protocols *

Ran Canetti
IBM T.J. Watson Research
Center
POB 704
Yorktown Heights, NY 10598
canetti@watson.ibm.com

Catherine Meadows
Naval Research Laboratory
Code 5543
4555 Overlook Ave., S.W.
Washington, DC 20375

Paul Syverson
Naval Research Laboratory
Code 5543
4555 Overlook Ave., S.W.
Washington, DC 20375

meadows@itd.nrl.navy.mil 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 trade-off between putting constraints on a protocol and putting constraints on the environment in which it operates.

Keywords

authentication protocols, security, requirements engineering.

1. INTRODUCTION

Contrary to what may sometimes seem the case from observing the academic literature, protocols do not run in a ping-pong fashion in complete isolation. In reality, protocols are typically run in a complex environment that includes a collection of protocols that run under varying circumstances or possibly based on the preferences of the principals running the protocol. In addition, one must consider not only the context of other runs of the protocol but also the potential interactions of various sub-protocols of a single protocol. Perhaps most surprisingly, one must consider the application environment of a protocol, even to evaluate the security of the protocol itself (irrespective of the security of the application).

*This research was partly funded by the Office of Naval Research.

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. We first recall (in Section 2) 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. The second attack concentrates on the danger in several concurrent runs of the same protocol. In Section 3 we describe an attack that demonstrates possible weaknesses that arise from bad interactions between a key exchange protocol and an “application protocol” that uses the key. The possibility of limiting the class of protocols that may securely run together with a given protocol is examined in Section 4. We conclude with some discussion in Section 5.

2. PROTOCOL COMPOSABILITY

A first example that illustrates the danger of separately analyzing sub-protocols of a larger protocol is taken from [14] which in turn discusses an attack, first described in [2], on a very early version of SSL. The early version included an optional client authentication phase 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. 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 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. 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. In any case, vulnerability of later authentications because of earlier session key compromise has long been recognized as a problem protocols should avoid [5].

The attack is a failure of an authentication requirement called *matching histories* [6]: "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, but the server's record shows a request to use strong crypto.

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 [6]. 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. 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 T his true identity and public key (e.g., ID_a, K_a), have T verify his true identity by some (typically non-cryptographic) means, and then obtain from T his 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 $\text{Token}_{ba} = \{s_b[R_b, R_a]\}_K$ 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 [13] that this protocol is subject to attack. Specifically:

- | | | |
|-----------|-----------------------|-----------------------------------|
| Message 1 | $A \rightarrow C_B :$ | R_a |
| Message 2 | $C \rightarrow B :$ | R_a |
| Message 3 | $B \rightarrow C :$ | $R_b, \{s_b[R_b, R_a]\}_{K_{ab}}$ |
| Message 4 | $C_B \rightarrow A :$ | $R_b, \{s_b[R_b, R_a]\}_{K_{ab}}$ |
| Message 5 | $A \rightarrow C_B :$ | $\{s_a[R_a, R_b]\}_{K_{ab}}$ |

Here, message 1, 4, and 5 are a protocol run that Alice attempts to run with Bob, but that Charlie attacks. Messages 2 and 3 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, other than to say she would be wrong in holding it. 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.

3. APPLICATION ENVIRONMENTS

Until now, our discussion of composition has involved either interleaved runs of a protocol with itself or of different sub-protocols 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 [11].) 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 “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 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 $B \rightarrow A$: *challenge*

Application Message 2 $A \rightarrow B$: $\{challenge\}_{K_{ab}}$

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 $C_B \rightarrow A$: $s_c[R_a, R_b]$

Application Message 2 $A \rightarrow C_B$: $\{s_c[R_a, R_b]\}_{K_{ab}}$

the following message may be added to the Lowe attack:

Message 6 $C \rightarrow B$: $\{s_c[R_a, R_b]\}_K$

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 [13]. 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 [12] SKEME protocol, which is based on STS. We consider the implications and limitations of environmental requirements in the next section.

4. ENVIRONMENTAL REQUIREMENTS

Restricting the application environment is obviously more difficult to control or specify than the security protocol, so we should justify the need for such a move before we consider it. Are there examples of attacks on protocols that satisfy agreement? One example of such was first given by Davida long ago [4] and later generalized in [10]. Although

neither of the papers explicitly notes the connection, the idea relies on a concept similar to blinding in the sense of Chaum [3], who used it very 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 $m^{d \cdot e} = m \pmod N$, and so decrypt the message.

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

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

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 $x^{-1} \pmod N$ 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 [8] 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 [9].)

To illustrate, here is a variant on the Needham-Schroeder shared-key protocol, made extensible-fail-stop.

$$\text{Message 1} \quad A \rightarrow S : \quad (A, S, T_a, NSSK, R, 1, B), \\ \{h(A, S, T_a, NSSK, R, 1, B)\}_{K_{as}}$$

$$\text{Message 2} \quad S \rightarrow A : \quad (S, A, NSSK, R, 2), \\ \{h(S, A, NSSK, R, 2), \\ (K_{ab}, B, \{h(S, B, T_s, NSSK, R, 3), (K_{ab}, A)\}_{K_{bs}})\}_{K_{as}}$$

$$\text{Message 3} \quad A \rightarrow B : \quad (S, B, T_s, NSSK, R, 3), \\ \{h(S, B, T_s, NSSK, R, 3), (K_{ab}, A)\}_{K_{bs}}$$

$$\text{Message 4} \quad B \rightarrow A : \quad (B, A, NSSK, R, 4), \\ \{h(B, A, NSSK, R, 4), N_b\}_{K_{ab}}$$

$$\text{Message 5} \quad A \rightarrow B : \quad (A, B, NSSK, R, 5), \\ \{h(A, B, NSSK, R, 5), N_b\}_{K_{ab}}$$

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:

$$\text{Application Message 1} \quad C_B \rightarrow A : \\ h(B, A, NSSK, R, 4), N_b$$

$$\text{Application Message 2} \quad A \rightarrow C_B : \\ \{h(B, A, NSSK, R, 4), N_b\}_{K_{ab}}$$

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.

5. DISCUSSION

We have given a set of increasingly rigorous 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 are suggested by the examples we have cited.

RoTh 1. 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 $K_{ab} = PRF_K(1)$, 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 [12] and in IKE, the Internet Key Exchange protocol (described in [7]). 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 [15].

RoTh 2. Secrets used in a protocol, even potential or obsolete secrets, should not be revealed by the environment.

Such a requirement would have prevented the blinding attack noted by Davida that we described above. Also, a version of NSSK that kept R secret and whose environment satisfied this requirement would not be subject to the above attack. Note, however, that this is a strong environmental requirement, and can be somewhat offset by the protocol requirement that release of outdated secrets should have no effect on a protocol’s security. This is basically the protocol requirement of *perfect forward secrecy*, described in [6].

RoTh 3. 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. And, the above attack on NSSK would be prevented not just by this rule, but by any one of the three rules of thumb we have stated.

This observation leads 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? Etc. 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.

6. 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] J. Benaloh, B. Lampson, D. Simon, T. Spies, and B. Yee. The private communication technology protocol, October 1995. draft-benaloh-pct-00.txt.
- [3] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology—Proceedings of Crypto 82*, pages 199–203, 1983.
- [4] 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.
- [5] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.
- [6] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2:107–125, 1992.
- [7] N. Doraswamy and D. Harkins. *IPSEC: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Prentice Hall, 1999.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] J. Thayer, J. Herzog, and J. Guttman. Mixed strand spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW12)*, pages 72–82. IEEE Computer Society Press, June 1999.