Protocol Interactions and the Chosen Protocol Attack

John Kelsey Bruce Schneier
Counterpane Systems
101 E. Minnehaha Parkway
Minneapolis, MN 55419
{kelsey,schneier}@counterpane.com

David Wagner
U.C. Berkeley
C.S. Div., Soda Hall
Berkeley, CA 94720-1776
daw@cs.berkeley.edu

Abstract. There are many cases in the literature in which reuse of the same key material for different functions can open up security holes. In this paper, we discuss such interactions between protocols, and present a new attack, called the chosen protocol attack, in which an attacker may write a new protocol using the same key material as a target protocol, which is individually very strong, but which interacts with the target protocol in a security-relevant way. We finish with a brief discussion of design principles to resist this class of attack.

1 Introduction

One of the most difficult engineering aspects of designing any secure system—cryptographic protocol, cryptographic primitive, etc.—is identifying all of the assumptions that may affect security. Most of the time, when designing cryptographic protocols, we silently assume that the only access anyone has to the keys involved is through the protocol steps, or other steps much like them.

However, the real world is not that clean: Tamper-resistant tokens can hold only a few public-key/private-key key pairs at one time; users are somtimes lucky to have even one certified public key. Given these sorts of design constraints, it's likely that the same private key(s) will be used for several different systems running on the same device. These systems will probably be designed by different people and fielded together without any analysis of their potential interactions.

A protocol may be quite secure alone, but may lose its security when another protocol exists that can be carried out with the same key pair. In fact, it is always possible, in principle, to defeat the security of protocol $\mathcal{P}$, if we are able to choose another protocol, $\mathcal{Q}$, to be run by the same participants in parallel with the target protocol. A general construction for protocol $\mathcal{Q}$ (the "chosen-protocol" attack) will be given below. In some cases, the chosen-protocol attack is not practical. However, on many systems such as smart cards and users' PCs, the attack can be both practical and effective.

In this paper, we discuss protocol interactions which can weaken the security of one or both protocols. We then describe a new attack, the "chosen protocol" attack, in which a new protocol is designed to interact with an existing protocol, to create a security hole. After discussing generalities, we give several specific examples of this attack. We then look at accidental protocol interactions. Finally, we discuss protocol design rules that appear to render the chosen protocol attack impossible.

2 Protocol Interactions

Let $\mathcal{P}$ and $\mathcal{Q}$ be two different protocols, both of which use the same key material, but which do different things. These protocols are said to **interact** whenever some information derived from $\mathcal{P}$ allows an attacker to successfully mount some attack on $\mathcal{Q}$. For example, suppose that both protocols rely for their replay-resistance on a digitally-signed timestamp from one party of the protocol. Then the protocols interact, since an attacker can now use his observations of the execution of $\mathcal{P}$ to mount a replay attack on $\mathcal{Q}$.

When $\mathcal{P} = \mathcal{Q}$, the possible protocol interactions reduce to a subset of possible attacks on the protocol. A replay attack (replaying messages from one instance of a protocol to attack another instance of the same protocol) is an example of an interaction between a protocol and itself, as is a standard man-in-the-middle attack.

There are a great many ways in which two different protocols can interact. For example, it may be possible for a third party to simply observe the messages in $\mathcal{P}$, and then mount an attack on $\mathcal{Q}$. Alternatively, it may be possible for an attacker to mount a variant of the man-in-the-middle attack: Alice believes she is executing protocol $\mathcal{P}$ with Bob, but instead, she is executing protocol $\mathcal{P}$ with Mallory, who uses the information derived to execute protocol $\mathcal{Q}$ with Bob in Alice's name. In still other cases, it may be possible for Mallory to execute protocol $\mathcal{P}$ in his own name with Alice, and simultaneously execute protocol $\mathcal{Q}$ with Bob in Alice's name. It is not necessary for the legitimate execution of $\mathcal{P}$ to complete successfully, so long as the attack on $\mathcal{Q}$ is made possible.

2.1 Sharing Keys Among Many Different Protocols

The one attribute common to all protocol interactions is that they involve the use of shared key material, either public/private key pairs or symmetric keys, between different protocols and applications. Most systems today use a different symmetric key or public/private key pair for each different function or application. This generally reduces the number of protocols having access to a given key or key pair to a manageable number. However, there are three forces pushing us toward a world in which different applications share common key material:

1. Certification. Certification of public keys is a costly process. Relatively few users are likely to want to maintain twenty certificates, paying a few dollars per year for each.
2. Cryptographic APIs. As cryptographic APIs become widespread, and more non-cryptographic applications make limited use of cryptography, it becomes increasingly likely that many different protocols will emerge, all of which may make use of the user's certified public key by default.

3. Smart cards. Smart cards and other cryptographic tokens often have limits on the number of key pairs they can store. If many applications are allowed to use the same smart card, then there will likely be some use of keys for different applications.

Already VeriSign certificates are used to provide security for PEM [Lin93, Ken93, Bal93, Kal93, Sch95, Sch96], S/MIME [RSA96, Dus96], SSL [FKK96], and SET [VM96]. Entrust Technologies [Cur96, Oor96] markets technology to provide for a common key archetecture over multiple applications.

3 Accidental Interactions

There are several examples of very simple known-protocol attacks. (These attacks are generally much simpler than the chosen protocol attacks.)

3.1 RSA signing and encryption

It has long been known that carelessly providing RSA signatures with an RSA key used for encryption could lead to a simple attack: The attacker intercepts a public-key encrypted block intended for Alice, and then presents it to her, requesting a signature. If Alice complies, then the attacker ends up with the decryption.

3.2 RSA signing and zero knowledge

A standard way of doing zero-knowledge proofs using RSA also provides an RSA signing oracle for a clever attacker.[And97a, MOV97]

3.3 A Banking Protocol Interaction

In [AK97], Ross Anderson describes a potentially disasterous interaction between two mechanisms in a bank's system for managing ATM cards. The bank requested a program to update all its PIN numbers for a new systemwide key, and the hardware/software vendor provided such a program, with the warning that it should be run immediately and then deleted. This program could be used to recover the PIN numbers of anyone's card using a protocol interaction.

3.4 Blind and Regular Signatures

Using the same RSA modulus for both regular and blinded signatures can, naturally, allow various attacks. In most blind signature schemes, the signer does not know what he is signing; he might be signing anything. Therefore, the blind signer can be used as a RSA signing oracle to sign arbitrary messages, which can then be used to defraud the protocol using regular signatures.

3.5 Interactions Among Different Applications

One of the most interesting places we can get a protocol interaction is when an identical protocol is used for different purposes in different applications, and no signed or authenticated statements bind an execution of the protocol to its specific application.

One example threat is that a user may be authorized to execute one application on a secure server, but not another application. If both use the same protocol to establish the user's identity, then the user may be able to access functions from which he should be restricted. Another threat is that the user may voluntarily run one protocol with an outsider, who then can use this information to run another protocol with some other entity in the original user's name.

One fairly serious threat can come in the form of electronic mail packages that may be set up to automatically send a digitally-signed reply of a message. (For example, a person might be sent some large binary, with a note in the subject line saying "hit reply to be removed from our mailing list.")

4 The Chosen-Protocol Attack

The "chosen-protocol attack" is an attack in which some attacker convinces one or more intended victims to accept and start using a new, tailor-made protocol, called the "chosen protocol." This protocol is designed specifically to interact with some already-running protocol, called the "target protocol." The chosen protocol should have no obvious weaknesses, but must allow an attack on the target protocol.

A chosen protocol can always be built to interact the security of any given target protocol, if there are no restrictions on what the protocol steps are allowed. This can be shown by example. Suppose there is a target protocol which uses private key material only for signing and decrypting. If the chosen protocol gives the attacker a decryption oracle and a signing oracle, then the attacker is trivially able to defeat the target protocol. By installing an oracle of this type, we can trivially break any other protocol that shares the same key material. (For that matter, the first step of the chosen protocol could, in principle, simply be to send all the private key material to the attacker.)

It is more interesting to consider chosen protocols that aren't obviously dangerous or weak, but that still allow the attack on the target protocol. Below, we give a method for constructing chosen protocols which seems to satisfy this requirement. However, without a rigorous definition of what a reasonable-looking protocol is, it isn't possible to prove that reasonable-looking chosen protocols of this kind actually exist. One interesting definition of "reasonable-looking" is that the protocol is not susceptible to attacks based only on the messages that occur in this protocol. However, it is certainly not clear how to prove that any given protocol has this property.

We can give a more general argument for why chosen protocol attacks will generally exist for any target protocol. Here, we consider a protocol P and a protocol Q which are intended to be run between Alice and Bob. In the attack, Eve will run an instance of Q with Alice in Bob's name, and use information from this to run an instance of P with Bob, in Alice's name.

Consider a normal execution of P. Each time Alice needs to generate a protocol message, she must have the information necessary to do so. Thus, that information must be contained in messages she has received during P's execution, or information she already has. Each time Alice is supposed to receive some information from a protocol message, again, she must have the information to do so, and that information, once again, must come either from previous messages received during P, or from information Alice already has. Now, consider an instance of Eve executing P with Bob in Alice's name, and executing Q with Alice in Bob's name. The first protocol message of Q that Alice sends to Eve must contain the information needed to make the first message in P that is supposed to come from Alice. Whatever response Eve gets from Bob, the next couple of steps in Q can be used to recover that information from Alice and get it back to Eve, and to form the right response to Bob's messages. (Note that during these steps of Q, Eve can give Alice things to put into this response message, to accomplish her purposes.) This can continue for as long as is needed. Since Alice has all the information Eve needs, if she is willing to use that information as Eve wants it used, even without giving her a signing or decryption oracle, Alice will wind up allowing Eve to attack the other protocol.

This leads to the way to foil this attack: Make certain that Alice never gives Eve correct signatures, decryptions, MACs, etc., for any other protocol in the chosen protocol she runs with Eve. This will be discussed further below.

4.1 Justification

At first glance, the chosen protocol attack may look like a purely academic attack. However, we can provide some realistic scenarios for chosen-protocol attacks:

1. The same cryptographic keys may be reused by different products. For example, if the user's certified keypair is used both in a home-banking product and in a video game to confirm high scores, the lower-security product's protocols may be chosen to interact with the higher-security product's protocols.
2. Infiltration of lower-security products. We might expect a company designing an application to securely transmit financial or medical records over the internet to be very careful in the design of its protocols. However, the company may reuse the same key material in some lower-security protocols used in the same product. The design of these lower security protocols may not be as carefully overseen as that of the high-security protocols.
3. Protocols used in commercial products can often be strongly influenced by requests from important customers. Such requests could be used to install a chosen protocol for this class of attack. Such protocols may wind up being adopted later as worldwide standards.

5 Constructing Chosen Protocols

In this section, we present three chosen protocols which we believe to be plausible. Each is designed to work with the target protocol, so that sharing of keys makes some degree of sense. This is not a necessary trait of chosen protocols, although may provide plausable deniability to an attacker deliberately designing a chosen protocol.

5.1 Agora

In [GS96], the authors introduce Agora, a simple electronic payment protocol designed specifically for pay-per-view web pages. The protocol for making a payment can be described as follows. (In this description, Alice is buying something from Bob.)

Protocol A: Agora (the Target Protocol):

1. Alice (the customer) sends to Bob (the merchant):
   $M_0 =$ Request for a price quotation from the merchant.
2. Bob forms:
   $N =$ a running sequence number kept by the merchant,
   $P =$ the price,
   $X_1 = MerchantCert_{Bob}, N, P,$
   and sends to Alice:
   $M_1 = X_1, SIGN_{SK_B}(X_1).$
   (He must also immediately increment N.)
3. If Alice wants to make the purchase, she verifies the certificate (including expiration date) and signature, forms:
   $X_2 = UserCert_{Alice}, N, P,$
   and sends to Bob
   $M_2 = X_2, SIGN_{SK_A}(X_2).$
4. Bob verifies Alice's certificate, her signature, and the values of N and P. If all is well, he delivers whatever was paid for. (In this context, this is probably a pay-per-view web page.)

Our aim is to build a special class of man-in-the-middle attack. In our attack, Alice thinks she's carrying out Protocol B (the chosen protocol) with Bob. Unfortunately for her, Mallory is sitting in the middle, using her protocol steps to allow him to carry out Protocol A (the target protocol) with Bob, in Alice's name.

For each step in the target protocol, we determine what information Mallory needs from Alice and from Bob in order to carry out the protocol. We can then design the chosen protocol to allow Mallory to get that information.

For example: In Step 3, Mallory needs

$$M_2 = X_2, SIGN_{SK_A}(UserCert_{Alice}, N, price)$$

from Alice. If he has this, he can successfully impersonate Alice to Bob. Therefor, Mallory also needs a plausable story behind a protocol designed to get him this information.

In light of concerns about children viewing inappropriate web pages, we might imagine another protocol, used with the same keys and certificates, for verification of adulthood with servers that don't charge for viewing their pages. We will design this chosen protocol so that it can be used by attacker Mallory to charge Alice for the web pages he views on Bob's page.

This age-verification protocol uses the same certificates as Agora, and works as follows:

Protocol B: Age-Verification Protocol (the Chosen Protocol):

1. Alice sends a request to Bob to view his page:
   $M_0 =$ Request to view page.
2. Bob responds by forming:
   $R_1 =$ A random challenge.
   $M_1 = R_1$.
   (Note that R_1 is designed to be the same size as the concatenation of the N and *price* in the Protocol A.)
3. Alice responds by forming:
   $X_2 = UserCert_A, R_1$.
   and sending
   $M_2 = X_2, SIGN_{SK_A}(X_2)$.

This protocol is secure when executed on its own, even though it has been designed specifically to subvert Protocol A.

To make this into a man-in-the-middle sort of attack, we put Mallory in Bob's place when Alice executes Protocol B (the age-verification protocol). Here is how Mallory uses his man-in-the-middle status with Alice in Protocol B to impersonate her in Protocol A:

1. Alice completex step 1 of Protocol B.
2. Mallory intercepts the message Alice sent to Bob in Step 1 of Proctocol B.
3. Mallory executes Step 1 of Protocol A with Bob.
4. Bob executes Step 2 of Protocol A with "Alice."
5. Mallory intercepts Bob's reply to Alice in Step 2 of Protocol A.
6. Mallory recovers N and P from M_1 of Protocol A and sends them to Alice as R_1, in Step 2 of Protocol B.
7. Alice responds with Step 3 of Protocol B.
8. Mallory allows to pass through to Bob.

At this point, Bob thinks he completed Protocol A with Alice, while Alice thinks she completed protocol B with Bob. Mallory is now free to intercept whatever information he forced Alice to buy from Bob.

This man-in-the-middle attack can work again and again. Each time Alice verifies her age to Mallory, he can now use her information to view web pages on an arbitrary merchants' pages, and stick her with the bill.

5.2 The Wide Mouth Frog Protocol

The Wide Mouth Frog protocol [BAN90, Sch96] is a well-known protocol for exchanging a symmetric encryption key using a trusted third party with whom each party shares a secret symmetric key. Its simplicity makes it an ideal example of how the chosen-protocol attack can work on symmetric, as well as public-key, protocols.

The key-exchange protocol, which will be our target protocol, works as follows[1]:

Protocol C: Wide Mouth Frog Protocol (the Target Protocol)

1. Alice wants to establish a session key with Bob. She begins by forming:
   K = a random 192-bit triple-DES key,
   T_A = current timestamp,
   and sends to Trent, the trusted third party,
   $M_0 = ID_A, E_{K_A}(T_A, ID_B, K)$.
2. Trent looks up the right secret key, K_A, and then decrypts the message and verifies the validity of the timestamp and ID_B. If all is well, he forms
   T_B = current timestamp (may be different than T_A),
   and sends to Bob
   $M_1 = E_{K_B}(T_B, ID_A, K)$.
3. At this point, Bob decrypts the message and verifies the timestamp and ID_A. If all is well, he now has a shared key with Alice, which he knows is authentic and fresh.

Our chosen protocol will be built to use this trusted third party and infrastructure of shared keys to allow secure logins. Now, we have a user, Alice, and a host, Mallory. This is how the basic protocol works:

Protocol D: Secure Login Protocol (the Chosen Protocol)

1. Mallory forms:
   R_0 = A 64-bit random number.
   T_x = Current Timestamp.
   and sends to Alice:
   $M_0 = LoginChallenge, T_x, R_0$.
2. Alice responds by sending Trent:
   $M_1 = LoginRequest, ID_A, E_{K_A}(T_A, R_0, hash(passphrase), ID_M)$.
   where T_A is the current timestamp, R_0 is the random number sent by Mallory, and ID_M and ID_A are Mallory's and Alice's IDs.
3. Trent verifies the timestamp and IDs, and then sends to Mallory:
   $M_2 = LoginMessage, E_{K_M}(T_M, hash(R_0, hash(passphrase)), ID_A)$.
4. At this point, Mallory verifies the timestamp and the hash. Note that Mallory only has access to the hash of the passphrase, and that outsiders never see even that.

Now, the chosen-protocol attack works as follows:

[1] We are filling in some specific values left open by the protocol's designers.

1. Mallory sends ID_B as R_0 in the first step of Protocol D.
2. Mallory catches Alice's response to Trent, strips away the *LoginRequest* header, and sends the rest of the message off to Trent as the first step of Protocol C.
3. Trent treats this as a valid request for a secure session with Bob from Alice. He sends the message to Bob.
4. Mallory now intercepts all messages from Bob to Alice, and impersonates her. Bob is convinced.

5.3 The DASS Public-Key Protocol

DASS (Distributed Authentication Security Service) is a protocol for mutual authentication and key exchange developed by Digital Equipment Corporation and marketed in a product called SPX [TAP90, TA91].

Protocol E: DASS (the Target Protocol)

– Protocol steps deleted for space considerations. See [Sch96].

To build a chosen protocol, we have to add some additional functionality to the system. In this example, we add a new protocol to have Trent generate random public keys for us, as needed.

Protocol F: Protocol for Requesting a Temporary Public Key (the Chosen Protocol)

1. Alice forms
   $R_0 =$ a random number the same length as an ID,
   and sends to Trent
   $M_0 = RequestForPK, ID_A, R_0$.
2. Trent generates a new public key, PK_T, forms
   $K_1 =$ a random encryption key,
   and sends back
   $M_1 = PKE_{PK_A}(K_1), E_{K_1}(SK_T), SIGN_{SK_T}(R_0, PK_T)$.

With this, we can build a chosen-protocol attack based on being the man-in-the-middle between Bob and Trent. We choose R_0 to be some person's ID, and then use the signed block to convince Bob that it's the right public key. Thus, this protocol looks like:

1. Mallory selects
   $R_0 = ID_A$,
   and sends to Trent
   $M_0 = RequestForPK, ID_M, R_0$.
2. Trent generates a new public key, PK_T, forms
   $K_1 =$ a random encryption key,
   and sends back to Mallory
   $M_1 = PKE_{PK_M}(K_1), E_{K_1}(SK_T), SIGN_{SK_T}(R_0, PK_T)$.

3. Mallory now encrypts a random session key under Bob's public key, including the timestamp, key lifetime, and Alice's ID. All this is exactly as appears in the third message of DASS.
4. Bob sends ID_A to Trent.
5. Mallory intercepts this request. He sends back $SIGN_{SK_T}(R_0, PK_T)$, recovered from the second message in the chosen protocol.
6. Bob decrypts this message, and uses PK_T to verify the signature on the first message sent to Bob. Bob is now convinced he shares a secret symmetric key with Alice, when in fact, he shares it with Mallory instead.

6 Design Principles for Avoiding Weakening Interactions

To prevent protocol interactions, we impose a few requirements on all protocols implemented with a given key or key pair.[2] If these rules of thumb are followed, then we believe a chosen protocol attack cannot work.

1. The first and most important rule to follow is to limit the scope of each key. A key should typically have only a small number of closely related functions. There is sometimes a temptation to reuse keys for related applications—this should be avoided whenever possible. If there is only one certified key pair, it should be used to sign (and perhaps even derive) other single-use public keys, as in [And97b, SH97]. This eliminates the overwhelming majority of possible protocol interactions.
2. Each application, protocol, version, and protocol step or operation that can be performed using a given key must have its own unique identifier, which must have a fixed length, and be used in a standard way in all protocol steps and operations. Note that it is very important that different versions of a protocol, or the same protocol running in two different applications, be differentiated here. The goal here is that each different use of a given key has a different unique identifier, and that this identifier is involved in whatever cryptographic operations are done using that key.
   This guideline is similar to the design approach, fail-stop protocols, advocated in [GS95]. There they suggested signing every protocol message, and including in each message a header containing the sender's name, the recipient's name, the protocol identifier and version number, a sequence number, and a nonce or timestamp. The point is to ensure that, if an active attacker injects a fake message, protocol execution will immediately halt. This allows one to consider only passive attacks when verifying the confidentiality of protocol secrets, which makes it possible to prove that secrets remain secret; this, in turn, allows one to apply BAN logic to verify the security of the protocol. In particular, their work on extensible fail-stop (and fail-safe) protocols provides significant progress towards ensuring composibility of protocols, even when private keys are reused. We conclude that the fail-stop design approach is

[2] Some of these rules were inspired by [And95].

valuable in the context of chosen-protocol attacks, and that including unique identifiers in each message is a powerful defensive design technique.

3. In message authentication and signing operations, simply including the fixed unique identifier a fixed place in the authentication operation will prevent chosen-protocol attacks from other messages that follow the same guidelines.
4. In public- and secret-key encryption operations and key-derivation operations, the unique identifier should be used in a way that makes the message impossible to decrypt without using the right unique identifier. There are several straightforward ways to do this: The simplest one conceptually is to use the protocol-identifier as a symmetric encryption key, and use it to encrypt whatever value is to be decrypted by the receiver in a way that an attacker cannot undo. For example:
   (a) When a public-key encryption is being used to send a symmetric encryption key, then that symmetric encryption key is encrypted under the protocol-identifier first, and then under the public key.
   (b) When symmetric encryption is being used, the message is first encrypted under the protocol-identifier, and then under the real symmetric encryption key. (Note that this will not work when the order of encryption is interchangeable, as in an OFB stream cipher.)
   (c) When symmetric encryption is being used that is order independent, or when the above guideline is too computationally expensive, the message is encrypted under a random symmetric key, and that key is encrypted and authenticated (in an order-dependent way, such as by using a block cipher) first with the protocol-identifier, and then with the actual secret symmetric key that would otherwise have encrypted the message.
   (d) When a symmetric key is being derived from shared secret information (such as might result from a Diffie-Hellman key agreement operation), the symmetric key is derived by hashing together the shared secret information and the protocol identifier.
5. Smartcards should include support for, and enforcement of, these mechanisms. Smartcards should be aware of public key reuse across protocols and applications.

The basic guideline is that every time some key is used, there must be a cryptographic binding between the message produced and the unique identifier of that message. This binding ensures that a message in one protocol cannot be substituted for some message in another protocol adhering to the same guidelines. In authentication and signing operations, simply including the unique identifier in the authenticated or signed operation is sufficient to prevent inter-protocol reuse of the signed message. In encryption operations, trying to decrypt a message with the wrong protocol or step identifier simply gives us an incorrect decryption.

7 Conclusions

The chosen-protocol attack resonates with a fairly common theme in security: security does not necessarily compose. Two protocols can each be secure on their own, but when they are implemented together the composed system may no longer be secure.

The chosen protocol attack brings to mind Bob Morris's comments at Crypto '96; he discussed building systems that are secure "even when they contain a John Walker" [Mor96] or, more generally, a nameless insider intent on attacking or subverting the system. What do you do if you've got an attacker on the design team for one of the (many) cryptographic protocols you rely on? How do you compartmentalize for robust security? The Internet owes its success to its mass decentralization, and its clever uses of existing infrastructure components in new and useful ways; we need to ensure that Internet security can survive in this environment.

8 Acknowledgements

The authors wish to thank Ross Anderson, Susan Langford, Mark Lomas, James Riordan, Paul Syverson, and all those who made comments during and after the presentation at SPW'97 for their invaluable help in improving this paper.

References

[And95] R. Anderson, "Robustness Principles for Public Key Protocols," *Advances in Cryptology — CRYPTO '95,* Springer-Verlag, 1995, pp. 236-247.

[And97a] R. Anderson personal communication, 1997.

[And97b] R. Anderson, "Perfect Forward Secrecy", presented at the rump session of Eurocrypt '97, 1997.

[AK97] R. Anderson, M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," these proceedings.

[Bal93] D. Balenson, "Privacy Enhancement for Internet Electronic Mail: Part III—Algorithms, Modes, and Identifiers," RFC 1423, Feb 1993.

[BAN90] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," *ACM Transactions on Computer Systems*, v. 8, n. 1, Feb 1990, pp. 18–36.

[Cur96] I. Curry, "Entrust Overview, Version 1.0," Entrust Technologies, Oct. 96. `http://www.entrust.com/downloads/overview.pdf`

[Dus96] S. Dusse, "S/MIME Message Specification: PKCS Security Services for MIME," IETF Networking Group Internet Draft, Sep 1996. `ftp://ietf.org/internet-drafts/draft-dusse-mime-msg-spec-00.txt`

[FKK96] A. Freier, P. Karlton, and P. Kocher, "The SSL Protocol Version 3.0", `ftp://ftp.netscape.com/pub/review/ssl-spec.tar.Z`, March 4 1996, Internet Draft, work in progress.

[GS95] L. Gong and P. Syverson, "Fail-Stop Protocols: An Approach to Designing Secure Protocols," *Fifth International Working Conference on Dependable Computing for Critical Applications*, Sept. 1995.

[GS96] E. Gabber and A. Silberschatz, "Agora: A Minimal Distributed Protocol for Electronic Commerce," *The Second USENIX Workshop on Electronic Commerce Proceedings*, USENIX Association, 1996, pp. 223–232.

[Kal93] B.S. Kaliski, "Privacy Enhancement for Internet Electronic Mail: Part IV—Key Certificates and Related Services," RFC 1424, Feb 1993.

[Ken93] S.T Kent, "Privacy Enhancement for Internet Electronic Mail: Part II—Certificate Based Key Management," RFC 1422, Feb 1993.

[Lin93] J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I—Message Encipherment and Authentication Procedures," RFC 1421, Feb 1993.

[Mor96] R. Morris, invited talk at Crypto '96.

[MOV97] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography,* p. 418, CRC Press, 1997.

[Oor96] P.C. van Ooorschot, "Standards Supported by Entrust, Version 2.0," Entrust Technologies, Dec 1996. `http://www.entrust.com/downloads/standards.pdf`

[RSA96] RSA Data Security, Inc., "S/MIME Implementation Guide Interoperability Profiles, Version 2," S/MIME Editor, Draft, Oct 1996. `ftp://ftp.rsa.com/pub/S-MIME/IMPGV2.txt`

[Sch96] B. Schneier, *Applied Cryptography, Second Edition*, John Wiley & Sons, 1996.

[Sch95] B. Schneier, *E-Mail Security,* John Wiley & Sons, 1995.

[SH97] B. Schneier and C. Hall, "An Improved E-Mail Security Protocol," in preparation.

[TA91] J. Tardo and K. Alagappan, "SPX: Global Authentication Using Public Key Certificates," *Proceedings of the 1991 IEEE Computer Society Symposium on Security and Privacy,* 1991, pp. 232–244.

[TAP90] J. Tardo, K. Alagappan, and R. Pitkin, "Public Key Based Authentication Using Internet Certificates," *USENIX Security II Workshop Proceedings*, 1990, pp. 121–123.

[VM96] Visa and MasterCard, "Secure Electronic Transaction (SET) Specification, Books 1-3" June 1996, `http://www.visa.com.cgi-bin/vee/sf/set/intro.html` or `http://www.mastercard.com/set/set.htm`.

This article was processed using the LaTeX macro package with LLNCS style