Authentication and Authenticated Key Exchanges[1]

Whitfield Diffie[2]

Sun Microsystems, 2550 Garcia Ave., Mountain View, CA 94043 USA

Paul C. van Oorschot and *Michael J. Wiener*

Bell-Northern Research, P.O. Box 3511 Station C, Ottawa, Ontario K1Y 4H7 Canada

1992 March 6

Abstract. We discuss two-party mutual authentication protocols providing authenticated key exchange, focusing on those using asymmetric techniques. A simple, efficient protocol referred to as the station-to-station (STS) protocol is introduced, examined in detail, and considered in relation to existing protocols. The definition of a secure protocol is considered, and desirable characteristics of secure protocols are discussed.

1. Introduction

The goal of an authentication protocol is to provide the communicating parties with some assurance that they know each other's true identities. In an authenticated key exchange, there is the additional goal that the two parties end up sharing a common key known only to them. This secret key can then be used for some time thereafter to provide privacy, data integrity, or both. In this paper, we discuss the security of public-key based authentication protocols, with and without an associated key exchange. We restrict our attention to two-party mutual authentication, rather than multi-party and one-way authentication protocols. We assume that individual underlying cryptographic mechanisms are not vulnerable, and restrict our attention to attacks on protocols themselves. An enemy (attacker, intruder, adversary) can see all exchanged messages, can delete, alter, inject, and redirect messages, can initiate communications with another party, and can reuse messages from past communications.

Much work has been done in recent years involving identification and authentication schemes using asymmetric techniques. Identity-based schemes, as introduced by Shamir [Sham84], rely on the existence of a trusted central authority, that holds secret information from which other secrets are generated and distributed to individual users when those users join the system. Günther [Gun89] has proposed an identity-based

1 Appeared in Designs, Codes and Cryptography, 2, 107-125 (1992), Kluwer Academic Publishers

2 This work was done while Whitfield Diffie was with Northern Telecom, Mountain View, California

protocol providing authenticated key establishment, making use of the ideas of Diffie-Hellman key exchange [Diff76] and the ElGamal signature scheme [ElGa85]. The authentication is indirect and does not offer perfect forward secrecy (see Section 4), although the latter can be provided at the cost of incorporating an extra exchange of Diffie-Hellman exponentials. Okamoto and Tanaka [Okam89] have proposed an identity-based authenticated key establishment protocol based on exponential key exchange and RSA. They offer versions which provide both indirect and direct authentication, although the latter, as presented, employs timestamps (Section 4), and some of the fields in the exchange may be unnecessary or redundant. Interactive identification protocols which provide proof of identity and make use of ideas involving zero-knowledge have been proposed by Fiat and Shamir [Fiat87], and more efficient protocols have been subsequently proposed by Guillou and Quisquater [Guil88] and Schnorr [Schn89], among others. These identification protocols differ from authenticated key exchanges in that the former do not provide keys for use in subsequent communications (e.g. for data integrity or data confidentiality).

As has been pointed out by many others [Bird91], [Burr90], [Gaar91], [Mit89], [Moor88], the design of cryptographic protocols in general, and authentication protocols in particular, is extremely error prone. The literature is filled with protocols which have been found to contain security flaws ranging from minor to fatal in severity. Furthermore, aside from security issues, it is a concern in practice that many of the published protocols contain redundancies or are inefficient with respect to the number of communications required, the number of cryptographic operations required (implying high computational demands), or the number and types of fields required in the communicated messages. This motivates the search for authentication protocols that are simple, require a minimum number of communications, a small number of fields in each message or token, and a small number of cryptographic operations. These considerations motivate the present work on public-key based protocols. Similar considerations motivated Bird et al. [Bird91] in their work on symmetric authentication protocols, which helped focus our attention on the idea of *matching* protocol runs (see Section 3). Our work extends the definition of a secure protocol to public-key based protocols with optional key exchange.

We are concerned with both authentication and key exchange. It is now well accepted that these topics should be considered jointly rather than separately [Baus89]. A protocol providing authentication without key exchange is susceptible to an enemy who waits until the authentication is complete and then takes over one end of the communications line. Such an attack is not precluded by a key exchange that is independent of authentication. Key exchange should be linked to authentication so that a party has assurances that an exchanged key (which might be used to facilitate privacy or integrity and *thus keep authenticity alive*) is in fact shared with the authenticated party, and not an impostor. For these reasons, it is essential to keep key exchange in mind in the design and analysis of authentication protocols.

In the remainder of this paper, we first provide some background regarding attacks on protocols, in an effort to motivate and give context to what follows. We then proceed with a definition of a secure protocol, and discuss characteristics that we consider desirable in an authentication protocol. We introduce a protocol referred to as the *station-to-station protocol*, examine it in detail, and justify its features. Some related protocols are discussed, and the proposed protocol is considered in relation to these. We conclude with a summary of principles we feel are important in the design of authentication protocols.

2. Notation and Motivation

Before discussing protocols in more detail, we first define some notation. For historical reasons, we give the two parties involved the names Alice and Bob.

$\{\cdot\}$ Braces indicate a hash function. $\{x, y\}$ is the result when a hash function is applied to x concatenated with y.

s_A Alice's secret key for a signature scheme. $s_A(x)$ is Alice's signature on x. $s_A\{x\}$ is Alice's signature on the hashed version of x.

p_A Alice's public key for a signature scheme. If the signature scheme is a public-key cryptosystem, then we define $p_A\{x\}$ and $p_A(x)$ to be Alice's public key encryption function with and without hashing.

Cert_A Alice's certificate, containing Alice's name (and possibly other information), her public key, and a trusted authority T's signature over this information. $\text{Cert}_A = (\text{Alice}, p_A, ..., s_T\{\text{Alice}, p_A, ...\})$. Cert_A binds the name Alice to the public key p_A. If Alice sends her certificate to Bob and provides evidence that she knows the secret key s_A corresponding to p_A, then she has provided evidence to Bob that she is in fact Alice.

$E_K(\cdot)$ Encryption using a symmetric cryptosystem with key K.

To illustrate an attack on a protocol and motivate what follows, consider the following simple (but flawed) challenge-response protocol where Alice and Bob sign each other's random number challenge.

Insecure simple challenge-response:

$$
\begin{array}{ccc}
\textbf{Alice} & & \textbf{Bob} \\
 & \xrightarrow{\quad R_A \quad} & \\
 & \xleftarrow{\quad \text{Cert}_B,\ s_B(R_A),\ R_B \quad} & \\
 & \xrightarrow{\quad \text{Cert}_A,\ s_A(R_B) \quad} &
\end{array}
$$

Alice begins by sending the random challenge R_A to Bob. Bob responds with his certificate, his signature on R_A and a random challenge R_B. Alice uses Bob's public key in Cert_B to verify Bob's signature, and then responds with her certificate and signature on R_B. Finally, Bob verifies Alice's signature.

An enemy Eve can impersonate Alice in a communication with Bob by passing Bob's challenge along to Alice:

Attack on the simple challenge-response:

Alice		**Eve**		**Bob**
			$\xrightarrow{R_E}$	
			$\xleftarrow{\text{Cert}_B,\ s_B(R_E),\ R_B}$	
			Eve now needs Alice's help to sign R_B.	
	$\xleftarrow{R_B}$			
	$\xrightarrow{\text{Cert}_A,\ s_A(R_B),\ R_A}$			
Eve now has the required signature, drops this call, and continues the call with Bob.				
			$\xrightarrow{\text{Cert}_A,\ s_A(R_B)}$	

Eve begins by initiating the protocol with Bob. When Bob sends the challenge to Eve, Eve initiates another instance of the protocol with Alice and gets Alice to sign Bob's challenge. Eve can then complete the authentication with Bob and successfully impersonate Alice. The main problem here is that the challenged party has no influence over what he will sign. (As a general rule, it is better if both parties have some influence over the quantity signed.) The challenger can abuse this protocol to get a signature on any quantity he chooses.

We now turn our attention to secure protocols.

3. Definition of a Secure Protocol

A particular instantiation of an authentication protocol is referred to as a *run*. Before presenting a definition of a secure protocol, we first consider the properties of what we consider to be a "successful" run. In a successful run, two communicating parties, Alice and Bob, exchange a number of messages at the end of which they have assurances of each other's identities and furthermore, optionally share a secret key known only to them. For every completed run, each party either *accepts* or *rejects* the other's identity and optionally a secret key. In a successful run, the run is completed and both parties accept.

Property 1 of a successful run: Both Alice and Bob *accept* each other's identities. If the authentication involves key exchange, then they both *accept* the exchanged key also.

The second property of a successful run concerns the records of a protocol run (assuming the participants had each recorded the exchange). To proceed, we require definitions regarding the use of the word *match* when applied to records of a run (a slightly different definition is given by Bird et al. [Bird91]).

Matching Messages: We say that a message from one record matches a message from another if one record lists the message as incoming, the other record lists the message as outgoing, and all fields of the message relevant to authentication are the same in both records.

The qualification *relevant to authentication* is necessary to allow individual messages to *match* even if they are not bit-wise identical. The motivation here is that if a message contains unsigned fields which are cryptographically irrelevant to authentication, then discrepancies in such fields alone should not preclude a message from meeting the definition of matching.

Matching Records of Runs: We say that two records of a run match if their messages can be partitioned into sets of matching messages (each set containing one message from each record), the messages originated by one participant appear in the same order in both records, and the messages originated by the other participant appear in the same order in both records. For simplicity, we do not consider protocols in which messages need not arrive in the order in which they were sent.

Note that messages originated by distinct participants do not have to be in the same order with respect to each other. This allows the case where messages in transit cross. In such a case, each participant will record his own message as having been sent before the crossing message is received.

Property 2 of a successful run: If Alice and Bob had recorded the exchange, then their records of the run will match.

We now distinguish between a *successful* run and a *secure* run. To consider a run successful by any reasonable definition, the run must be considered secure in the intuitive sense. On the other hand, it is possible for a run to be unsuccessful even in the absence of security breaches (e.g., if both legitimate parties *reject* for some reason). It is also always possible that an enemy may delay a legitimate message of a run indefinitely. Suppose that in a particular run, Alice accepts Bob's identity, sends the last message of the protocol to Bob, and then an enemy destroys this message. Assuming Bob must receive this message before accepting Alice's identity, Bob will not accept Alice's identity. Intuitively, while this run is unsuccessful, there have been no security breaches; at the time that Alice accepted Bob's identity (before she sent the last message), Bob's record of the partial run matched Alice's record. For our purposes, such a *denial of service* attack in itself is not considered a security breach; such problems often must be dealt with by physical security and other techniques.

We are now in a position to define what it means for a run of a (symmetric or asymmetric) mutual authentication protocol to be insecure:

Definition 1: A particular run of a protocol is an *insecure run* if any party involved in the run, say Alice, executes the protocol faithfully, accepts the identity of another party, and either of the following conditions holds:

- 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 does not match Alice's record.
- The exchanged key accepted by Alice is known to someone other than the party whose identity Alice accepted. (This condition does not apply to authentication without key exchange.)

Note that under this definition a conventional key exchange protocol requiring a trusted third party [Kohl91] is not secure.

It should be clear that Alice's record, which must match that of the other party in the above definition, is the actual record she had at the point in time at which she has received enough information to carry out any computations required to reach the *accept* state; messages sent or received subsequent to this are irrelevant.

The goal of the enemy is to cause a run to be insecure. The goal of the designer of the protocol is to make the enemy's task impossible (or computationally infeasible) in all instances. Reversing Definition 1, we get a definition of a secure (symmetric or asymmetric) mutual authentication protocol:

Definition 2: A *secure protocol* is a protocol such that the following conditions hold 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.
- It is computationally infeasible for the exchanged key accepted by Alice to be recovered by anyone other than Alice and possibly the party whose identity Alice accepted. (This condition does not apply to authentication without key exchange.)

By themselves, the above definitions are not particularly helpful in deciding whether a given protocol is secure, in that they do not lead to constructive procedures to either verify or expose weaknesses of a protocol. Nonetheless, these definitions can be applied directly in deciding whether a given potential attack is a *real* attack. For example, in an authentication with key exchange, suppose an enemy merely intercepts Alice's and Bob's messages and then passes them along unchanged. Intuitively, the enemy has not compromised the system in this case; the parties have accepted each other's identities, have matching records of the run, and exclusively share a secret key. Note that by Definition 1, such a run is not insecure. In other cases a supposed attack may become quite convoluted, and it may not be obvious that the attack amounts to just passing along messages. Definition 1 can be used to distinguish such a pass-along non-attack. In Section 5, this definition serves well in identifying real attacks; in particular, the second condition, which appears trivial, is essential.

While formal analysis techniques have been successfully used to uncover weaknesses in some authentication protocols (see Section 6), proof of correctness is more difficult, and depends heavily on proper modeling of goals and assumptions. Another technique available for uncovering weaknesses is that of exhaustive search with respect to interleaving attacks [Bird91]. Unfortunately, since there are as yet no absolute proofs of correctness, confidence in a protocol develops only over time as experts conduct analysis of the protocol and fail to find flaws.

4. Desirable Protocol Characteristics

In addition to being secure, there are other desirable characteristics for a protocol.

Perfect Forward Secrecy

An authenticated key exchange protocol provides perfect forward secrecy if disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs. The property of perfect forward secrecy does not apply to authentication without key exchange.

Direct Authentication

In some authenticated key exchange protocols, authentication is not complete until both parties prove knowledge of the shared secret key by using it in subsequent communications. Such a protocol is called *indirect*. When authentication is established by the end of each protocol run, the protocol is *direct*. An indirect protocol can be modified to be direct by adding an exchange of known messages or messages with redundancy encrypted with the exchanged key. For authentication without key exchange, an indirect protocol provides no security because neither party can accept the other's identity.

No Timestamps

While timestamps are convenient for administrative and documentation purposes, it is desirable in practice to avoid relying on their use for security in authentication protocols. Difficulties, precautions, and objections to timestamps are well-documented in the literature [Bell90], [Bird91], [Gaar91]. For convenience, we summarize the more notable issues below.

To use timestamps for authentication, all parties must maintain local clocks that are periodically synchronized in a secure manner with a reliable source of time. Between synchronizations with the reliable time source, local clocks may drift. Two parties, Alice and Bob, must allow a time window for timestamps to compensate for local clock drift and the fact that messages take time to cross a network. Alice will accept any timestamp from Bob that is within a window around the time on Alice's local clock as long as Bob has not used this particular time value before. Alice can either store all time values used by all other parties that are within her current window (which is impractical in some communications environments) or she can store the latest time used by each party and insist on strictly increasing time values from each party. However, in the strictly increasing time values case, if Bob uses a time *t* far into the future for some reason (e.g. severe clock drift or improper synchronization with the reliable time source), then Bob will not be able to communicate with Alice until time *t* is within her window. To prevent this problem, Alice would have to store time *t* and not update her record of the latest time value used by Bob. This could potentially lead to a choice among storing large quantities of data, sacrificing communications availability, or sacrificing security. Concerning communications availability, if two parties' local clocks are too far out of synchronization, then the parties cannot communicate. This tends to make those concerned with communications availability want wide time windows which increases storage requirements. While timestamps are convenient from a theoretical point of view, they present a number of practical problems. Protocols based on random numbers do not suffer from these difficulties.

Recently formal analysis has been used in the verification of authentication protocols [Burr90], [Gaar91]. Starting with a list of initial formal beliefs, the objective is to logically derive the stated protocol goal by consuming the list of protocol steps. One of the basic assumptions on which such analysis is typically based is that the parties involved have the ability to check the *freshness* of timestamps. In fact, one of the main results of the work by Gaarder and Snekkenes is the identification of the security requirement that time clocks be trustworthy in certain protocols. This means that in practice, the security of timestamp-based protocols relies heavily on the proper implementation of synchronized and secure time clocks. Unfortunately, despite much discussion in the literature regarding timestamp-based protocols (e.g. [Den81], [Habe91]), when it comes to actually implementing such a protocol, the significance of the security of time clocks is easily lost, and furthermore, the costs associated with a proper implementation can be significant.

5. Station-to-Station Protocol

We now introduce a simple, efficient authenticated key exchange protocol called the *station-to-station* (STS) *protocol*. The STS protocol has evolved over time; an early version of this work was described at the 1987 International Switching Symposium [OHig87]. The STS protocol reached its present form in 1989. We believe that it is secure according to Definition 2 and has a number of other desirable properties. In the remainder of this section, we describe the protocol, discuss its properties, and justify its subtle details by showing how variants of it are vulnerable.

5.1 Basic STS Protocol

The STS protocol consists of Diffie-Hellman key establishment [Diff76], followed by an exchange of authentication signatures. In the basic version of the protocol, we assume that the parameters used for the key establishment (i.e., the specification of a particular cyclic group and the corresponding primitive element α) are fixed and known to all users. While we refer to the Diffie-Hellman operation as *exponentiation*, implying that the underlying group is multiplicative, the description applies equally well to additive groups (e.g., the group of points of an elliptic curve over a finite field). We also assume in this section that Alice knows Bob's authentic public key, and vice versa; this assumption is dropped in the following section.

The protocol begins with one party, Alice, creating a random number x and sending the exponential α^x to the other party, Bob (see diagram below). Bob creates a random number y and uses Alice's exponential to compute the exchanged key $K = \alpha^{xy}$. Bob responds with the exponential α^y and a token consisting of his signature on the exponentials, encrypted with K using a suitable symmetric encryption algorithm E (i.e., $E_K(s_B\{\alpha^y, \alpha^x\})$). Alice computes K, decrypts the token using K, and verifies Bob's signature using Bob's public key. Alice sends to Bob her corresponding encrypted signature on the exponentials, $E_K(s_A\{\alpha^x, \alpha^y\})$. Finally, Bob similarly verifies Alice's encrypted signature using K and Alice's public key. The security of the exponential key exchange relies on the apparent intractability of the discrete logarithm problem [Odly91].

Basic STS Protocol:

$$
\begin{array}{ccc}
\textbf{Alice} & & \textbf{Bob} \\
 & \xrightarrow{\qquad\qquad \alpha^x \qquad\qquad} & \\
 & \xleftarrow{\qquad \alpha^y,\ E_K(s_B\{\alpha^y, \alpha^x\}) \qquad} & \\
 & \xrightarrow{\qquad E_K(s_A\{\alpha^x, \alpha^y\}) \qquad} &
\end{array}
$$

It is possible to create a more symmetric version of this protocol where the parties exchange exponentials first and then exchange encrypted signatures in separate messages. This would make it permissible for the exponential messages to cross, and then the encrypted signature messages to cross. In such a case, neither Alice nor Bob need know who initiated the call. This is desirable, as situations exist in practice (e.g. in both voice telephony and X.25 data transfer) in which at certain implementation levels, it is not known which party initiated a call. This explains why each party forms his signature with his own exponential listed first. If the exponentials were in the same order in both signatures, then Alice and Bob would have to find a way to agree on whose exponential should be listed first (such as by basing the decision on which party initiated the call).

At this point, consider what assurances the STS protocol provides to the participants. From Bob's point of view, as a result of the Diffie-Hellman key exchange, he shares a key known only to him and the other participant who may or may not be Alice. Our assumption in this section is that Bob knows Alice's public key (this is achieved in the section below through use of certificates). Because Alice has signed the particular exponentials associated with this run, one of which Bob himself has just created specifically for this run, her signature is tied to this run of the protocol. By encrypting her signature with K, Alice demonstrates to Bob that she was the party who created x. This gives Bob assurance that the party he carried the key exchange out with was, in fact, Alice. Alice gets a similar set of assurances from Bob.

The STS protocol has the desirable characteristics discussed in section 4. Rather than using timestamps, challenges are used. Because the parties demonstrate knowledge of the exchanged key by encrypting their signatures, the authentication is direct. The STS protocol also offers perfect forward secrecy. The only long-term secret keying material stored by users is their secret keys for the signature scheme. If a secret key is compromised, the security of exchanged keys from earlier runs is not affected because Diffie-Hellman key exchange is used; Diffie-Hellman key exchange has no long-term keying material. There are two other desirable properties of the STS protocol. The first is that public key techniques are used to make key management simpler and more secure than is possible using conventional cryptography. If parties generate their own secret keys, these keys need never be disclosed (to anyone, including any supposedly trusted party), even during initialization. The second is that there is no need for communicating parties to contact a central facility on a per-call basis. If certificates are used for distributing public keys (see Section 5.2), once a party has its own certificate and the trusted authority's public key, it can exchange keys with, and authenticate other parties without consulting a central facility. The protocol appears to strike an elegant and difficult balance, being simple and secure without utilizing unnecessary or redundant elements.

To illustrate the need for the features of the STS protocol, it is now demonstrated how the protocol is weakened when the following modifications are made: removing the encryption of the signatures, signing only one's own exponential, signing only the other party's exponential, or uncoupling authentication from key exchange.

Removing encryption on signatures

Consider a modified STS protocol where the signatures on the exponentials are not encrypted with the exchanged key K. Because the exponentials are public information, any other party could sign them as well. Suppose that in the last message of the protocol, an enemy Eve substitutes her own signature on the exponentials for Alice's signature. (If the parties exchange public keys using certificates, Eve would have to substitute her own certificate for Alice's certificate.) This may not seem like a serious attack, as Eve does not know the exchanged key. However, if Bob were a bank, Eve could get credit for a deposit Alice might make. Interestingly, even though Bob has been misled here, Alice is the party who may be hurt.

Having informally discussed why the above run is insecure, we now apply Definition 1. Bob executed the protocol faithfully and accepted Eve's identity, but the exchanged key is known to a different party, Alice. By Definition 1, the run is insecure. Because an insecure run is possible, the modified protocol is insecure.

Signing only one's own exponential

Consider the variant of the STS protocol where each party signs only his own exponential (i.e. Alice's encrypted signature is $E_K(s_A\{\alpha^x\})$ and Bob's is $E_K(s_B\{\alpha^y\})$). We know of no general attack which applies to this case, but there is an attack which applies when the signature scheme is RSA [Riv78], the hash function is the identity function, and Diffie-Hellman key exchange is carried out over GF(p). In this case, Eve can impersonate Alice in a run with Bob by using x=0 as the exponent in the key exchange. Eve's exponential is $\alpha^0 = 1$, and the exchanged key is $K = \alpha^{xy} = 1$. Eve requires the following encrypted signature

$$
\begin{aligned}
E_K(s_A\{\alpha^x\}) &= E_1(s_A\{1\}) \\
&= E_1(s_A(1)) && \text{because the hash function is the identity function} \\
&= E_1(1) && \text{because signing in RSA is exponentiation and } 1^z = 1 \text{ for all } z
\end{aligned}
$$

Eve can compute $E_1(1)$, and hence can impersonate Alice. Although this attack applies only to a specific case, it illustrates a more general problem in signing only one's own exponential: if Eve can obtain a quantity for which she can acquire or compute the discrete logarithm, and can acquire or compute Alice's signature on the quantity, then Eve can use (and reuse) this quantity as an exponential to impersonate Alice. By introducing the second exponential into the data to be signed, an adversary is forced to solve a different instance of the problem in real time each time impersonation is attempted.

Signing only the other party's exponential

Consider the variant of the STS protocol where each party signs only the other party's exponential (i.e. Alice's encrypted signature is $E_K(s_A\{\alpha^y\})$ and Bob's is $E_K(s_B\{\alpha^x\})$). Again, we know of no general attack which applies to this case, but there are some concerns.

In principle, it is imprudent to sign arbitrary text supplied by a potential adversary. In the case at hand, in order for an adversary to recover the signature, he would have to know the key K. To compute K, the adversary would need to know the discrete logarithm of the quantity being signed. While an adversary would not in general know the logarithm of a particular fixed quantity he might desire signed, it is trivial to produce such quantities by preselecting logarithms, and it would appear undesirable to allow an adversary the freedom to acquire signatures on any quantities whose logarithms are known.

In Section 5.3, it is shown that the STS protocol can be reduced to an authentication-only protocol by replacing exponentials with random numbers and removing the encryption on the signatures. If each party were to sign only the other party's exponential, then the authentication-only variant would be subject to the attack on the simple challenge-response outlined in Section 2. Similarly, signing only one's own exponential does not result in a protocol which reduces to a secure authentication-only variation.

Note that even should it turn out that signing both exponentials does not provide more security than simply signing a single exponential, the only added cost in doing the former is additional hashing, which in general is relatively minor. No additional operations involving the signature scheme, symmetric cryptosystem operations, or data transmission are introduced by signing both exponentials rather than one only.

Uncoupling authentication from key exchange

If the STS protocol is modified so that authentication is uncoupled from key exchange by having the parties sign some quantity that is independent of the exponentials, the resulting protocol is subject to the classical *intruder-in-the-middle* attack (e.g., [Riv84]) on Diffie-Hellman key exchange:

Alice		**Eve**		**Bob**
	$\xrightarrow{\quad\alpha^{x}\quad}$		$\xrightarrow{\quad\alpha^{x'}\quad}$	
	$\xleftarrow{\quad\alpha^{y'}\quad}$		$\xleftarrow{\quad\alpha^{y}\quad}$	
	$K_A = \alpha^{xy'}$		$K_B = \alpha^{x'y}$	

Eve substitutes her own exponentials for Alice's and Bob's exponentials. This results in Alice and Bob calculating two different keys, both of which can be calculated by Eve. Eve shares key K_A with Alice, and key K_B with Bob. During the authentication phase of the run, Eve can pass Alice's encrypted messages to Bob and vice versa by decrypting the messages with one key and re-encrypting with the other. After authentication, Eve is free to passively eavesdrop or to inject her own messages. By Definition 1, this modified protocol is insecure because while Alice executed the protocol faithfully and accepted Bob's identity, the exchanged key is shared with a different party, Eve. There is a similar problem from Bob's point of view.

5.2 STS Protocol in Practice

We now describe the use of the STS protocol in practice, for the specific case where the key exchange is carried out in the multiplicative group of a finite field. For clarity, we focus on prime fields $GF(p)$. Two parameters are required then for Diffie-Hellman key exchange: a primitive element α in $GF(p)$, and a suitable prime p. The prime p should be chosen to preclude Pohlig-Hellman type attacks [Poh78]. In light of recent work on the discrete logarithm problem ([Odly91] for prime fields; [Odly84] for fields of characteristic two), it is prudent to use a distinct field for each user (i.e., for $GF(p)$, a distinct prime p, chosen by the user himself). The best known attacks on Diffie-Hellman key exchange over finite fields are the index-calculus techniques involving a massive pre-computation which yields a database specific to a particular field. The database then allows computation of individual logarithms in that field relatively quickly. If a single field is used for an entire network, a single database allows the compromise of *all* key exchanges – providing great incentive to attempt to construct the database.

To facilitate the distribution of user's public keys and user-specific Diffie-Hellman parameters, certificates may be used. In addition to these items, a certificate should contain the user's name and the signature of the trusted authority over these data items. The reason for the inclusion of the (α, p) pair in the certificate is explained below. The STS protocol is then as follows.

STS Protocol in practice:

Alice **Bob**

$$\xrightarrow{\quad \alpha,\ p,\ \alpha^x \quad}$$

$$\xleftarrow{\quad \alpha^y,\ \mathrm{Cert}_B,\ E_K(s_B\{\alpha^y, \alpha^x\}) \quad}$$

$$\xrightarrow{\quad \mathrm{Cert}_A,\ E_K(s_A\{\alpha^x, \alpha^y\}) \quad}$$

$$\mathrm{Cert}_A = (\mathrm{Alice}, p_A, \alpha, p, s_T\{\mathrm{Alice}, p_A, \alpha, p\})$$

The differences here are as follows. Alice sends her Diffie-Hellman parameters along in the first message; Bob uses these instead of fixed network-wide parameters. Upon receiving the third message, Bob verifies that the Diffie-Hellman parameters sent in the first message agree with those actually in Alice's certificate. In the second message, Bob sends Alice his certificate, from which Alice can extract his authentic public key; Alice verifies authenticity by checking the signature of the trusted authority on Bob's certificate. Similarly, in the third message, Alice sends Bob her certificate, allowing Bob to extract her authenticated public key, after similarly verifying the trusted authority's signature on her certificate. Note that Bob does not need Alice's certificate until the third message, and in fact may not wish to receive it earlier, since this may require having to allocate storage to save the certificate until needed upon receipt of the third message. A further reason for Alice to delay sending her certificate until the third message is to allow both Alice and Bob the option to encrypt their certificates with the exchanged key. Although certificates are, in theory, public information, it may be desirable in some applications to prevent an eavesdropper from seeing them in order to prevent a passive eavesdropper from learning Alice and Bob's identities.

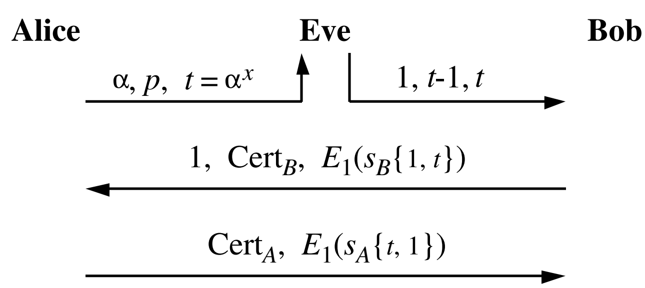
Note that knowledge of the other party's public key is not required to construct and send or to receive and process the first message. If the public key were required at this stage, then introducing certificates would necessitate an additional preliminary message to make the certificate available earlier.

As discussed in Section 5.1, it may be desirable in some cases to allow both parties to send the initial message simultaneously. In this case, some method must be used to establish one of the parties as the *dominant* party (i.e. the party whose α, p pair will be used). The nondominant party would then continue the protocol with the second message. An example of a simple method would be to choose the party with the larger prime p to be dominant.

It is now shown that the protocol is weakened if Diffie-Hellman parameters are not included in certificates.

Removing Diffie-Hellman parameters from certificates

Without Diffie-Hellman parameters in certificates, the enemy, Eve, has the freedom to modify α and p in Alice's first message. Let Alice's exponential be $t = (\alpha^x \bmod p)$. Suppose that Eve changes α to be 1 and p to be t-1 (see diagram below). Then Bob's exponential is $1^y \bmod (t\text{-}1) = 1$, and Bob calculates the exchanged key to be $t^y \bmod (t\text{-}1) = 1$. Alice calculates the exchanged key to be $1^x \bmod p = 1$. Because Eve does not modify the exchanged exponentials and Alice and Bob calculate the same exchanged key, Alice and Bob will accept each other's encrypted signatures.



Eve knows the exchanged key and after authentication, she is free to both eavesdrop and inject her own messages. Note that Alice and Bob accepted each other's identities, but their records of the run do not match, and the exchanged key is known to a third party; the modified protocol is thus insecure by our definitions, as well as intuitively.

While it may appear that the above-described substitution is trivial and easily detected by special checks, the potential for compromise remains. More sophisticated or disguised related attacks appear possible, including the possible use of Pohlig-Hellman-weak primes. The fundamental concern is that in order to rely on the believed intractability of the Diffie-Hellman problem, it must be ensured that suitable Diffie-Hellman parameters are in fact used.

5.3 Authentication-Only version of STS Protocol

It is possible to turn the STS protocol into an authentication-only protocol by replacing the exponentials with random numbers and removing the encryption on signatures:

Authentication-only STS Protocol:

Alice **Bob**

$$\xrightarrow{\quad R_A \quad}$$

$$\xleftarrow{\quad \mathrm{Cert}_B,\ R_B,\ s_B\{R_B, R_A\} \quad}$$

$$\xrightarrow{\quad \mathrm{Cert}_A,\ s_A\{R_A, R_B\} \quad}$$

This simplified protocol is essentially the same as the three-way authentication protocol currently proposed by ISO [9798-3]. This is discussed further in the following section.

6. Discussion of Other Protocols

From the intruder-in-the-middle attack on unauthenticated Diffie-Hellman key exchange to spoofs in the spirit of the well-known “grandmaster postal-chess” problem,[1] attacks on authentication protocols are numerous and well-documented in the literature. Burrows, Abadi and Needham analyzed eight protocols and found six to contain redundancies, and four to contain flaws [Burr90, Table I], including both redundancies and flaws in the CCITT X.509 mutual authentication protocols [X509]. To get a flavour of the concerns we have with many of the currently proposed protocols, we briefly discuss two of the four protocols analyzed by Burrows et al.: Kerberos, and one of the X.509 protocols. We also discuss a related ISO protocol.

Kerberos protocol. The popular Kerberos protocol [Kohl91], based on symmetric cryptosystems, has several features which make it somewhat undesirable in various applications. These include the use of timestamps (discussed earlier), the requirement of an on-line authentication server, and redundancies in the protocol itself. These and further issues are raised by Bellovin and Merritt [Bell90].

---

1 A novice who engages in two simultaneous chess games with two distinct grandmasters, playing white pieces in one game and black in the other, can take his opponents’ moves in each game and use them in the other to guarantee himself either two draws or a win and a loss, and thereby unfairly have his chess rating improved.

Three-pass CCITT X.509 authentication protocol. The CCITT X.509 recommendation [X509] is a very widely known internationally standardized authentication protocol based on public-key cryptography. The one and two-pass X.509 protocols require timestamps, while timestamps are redundant in the three-pass protocol; the specification allows that the timestamp field may be zero in this latter case (making the three-pass protocol practical, although it would be preferable if no field at all had to be allocated for timestamps). Some concerns regarding the protocol are now summarized. The final message of the protocol is Alice's signature on both Bob's challenge and Bob's identity: $s_A\{R_B, \text{Bob}\}$.[1] This allows Bob to obtain the signature of Alice on a quantity over which Bob has control. This is undesirable, although it is not clear how to use this to mount a direct attack. A second concern involves the suggested use of the optional encrypted data field in the protocol to accomplish key exchange; this use does not guarantee perfect forward secrecy.[2] A further issue with the use of this field is that there is no guarantee that the sender of the encrypted data actually knows the encrypted data itself, and in fact an adversary can pass off another party's encrypted data as his own [Burr90], [Gaar91]. A third concern [IAns90] is the restriction that the signature system used must be capable of both signing and encrypting data, which rules out many candidate signature schemes including the proposed NIST Digital Signature Algorithm [DSA91].

ISO three-way protocol. As noted in Section 5.3, the authentication-only version of the STS protocol is essentially the same as the three-way protocol currently proposed by ISO [9798-3]. The differences are that the ISO protocol allows redundant copies of the random numbers, optional fields for the identity of the intended recipient of a message, and optional fields for arbitrary text. Due to limitations of authentication-only protocols as discussed earlier, in most applications it is expected that the key establishment functionality of the ISO protocol (provided by the optional text fields both within and outside the signed portion of each message) will be employed. Recalling the concern noted above in X.509, care must be taken in the use of these fields; furthermore, note that their use to transfer encrypted session keys does not guarantee perfect forward secrecy.

Attack on a specific authentication protocol.

To augment the literature documenting attacks on specific protocols, and to further emphasize how easily flaws can be introduced and overlooked, we now consider the following (flawed) variation of the ISO authentication exchange. In fact, this variation was a preliminary version of the protocol. Here, Alice is allowed to use a new random number R_A' in place of R_A in the third message; R_A' is then also sent along as an additional cleartext field in the third message. In this modified protocol, an enemy Eve can authenticate herself as Alice to an unsuspecting party Bob as follows (see diagram below). Eve calls Bob, pretending to be Alice, sending a challenge to Bob; Eve responds

[1] In an early version of X.509, the final message was simply $s_A\{R_B\}$; the recommendation has since been formally updated.

[2] Note that the use of RSA [Riv78] in the obvious manner to achieve key exchange similarly does not guarantee perfect forward secrecy.

to Bob's counter-challenge by calling Alice and getting her to respond correctly to the challenge; Eve then drops the call with Alice and passes the correct response along to Bob, thus completing the authentication from Bob's point of view. Note that this attack is successful even if the identity of the intended recipient of each message is incorporated within the signed portion of each authentication token, as is optionally permissible in the formal definition of the related ISO protocol. To emphasize this, these principals' identities are included, and annotated with asterisks, in the attack detailed below. For simplicity, certificates are not shown.

Alice		**Eve**		**Bob**
		Choose R_A. Send message to Bob pretending to be Alice.		
			R_A →	
			← R_B, Alice*, $s_B\{R_B, R_A, \text{Alice*}\}$	
		Use R_B. Send message to Alice pretending to be Bob.		
	← R_B			
	R_A', Bob*, $s_A\{R_A', R_B, \text{Bob*}\}$ →			
			R_A', Bob*, $s_A\{R_A', R_B, \text{Bob*}\}$ →	
		Eve drops the call with Alice. Now Bob believes that Eve is Alice. The attack has succeeded.		

Regarding other attacks documented in the literature, we note that Bird et al. ([Bird91], section 4) detail an attack on a specific protocol. This is a specific case of the general class of *reflection attacks* in which a challenger is tricked into providing answers to his own questions [Mit89].

7. Concluding Remarks

Below are some general principles that appear prudent to follow in the design of authentication protocols. While many of these have been previously observed, we find it convenient to collect them here.

1. Authentication and key exchange should be linked.

If authentication and key exchange are independent, then an attacker could allow two parties to carry out authentication unhindered, and could take over one party's role in key exchange. This would allow the attacker to impersonate a valid party after authentication and key exchange are completed.

2. Asymmetry in a protocol is desirable.

Symmetries in a protocol should be used with caution, due to both the possibility of reflection attacks, and attacks in which responses from one party can be re-used within a protocol. As an obvious illustrative example, the authentication responses of each of two parties should not be identical.

3. Messages within a particular protocol run should be logically linked or "chained" in some manner, to prevent the re-use of previous messages or the introduction of messages from a parallel run.

The objective here is to preclude replay attacks and interleaving attacks. Messages should also be linked to the current time frame (e.g., through incorporation of recently generated random numbers). The specific attack detailed in Section 6 is possible due to a lack of such chaining of messages; similarly, the middleperson attack discussed by Bengio et al. [Beng91] is possible in protocols which fail to address this principle.

4. A party carrying out a cryptographic operation (serving as a signature) should be able to incorporate into the data being operated on a reasonable amount of data which he himself randomly selects.

In other words, a protocol should not require a party to carry out a cryptographic operation on inputs which may be entirely under the control of an adversary. This "add your own salt" principle is aimed at preventing an adversary from obtaining responses to specific questions he himself may not be able to answer. This should also prevent so-called chosen-ciphertext attacks [Bras88, p. 27]. Related to this principle, we note the following principle paraphrased from Moore [Moor88, section II]:

5. Valid signatures should result from the transformation of a message from a message space which is a sparse subset of the domain of the signature function.

For example, requiring redundancy, or some other expectation, in the data to be signed, may thwart attacks whereby an adversary attempts to forge new signatures by combining previously obtained valid signatures. For the STS protocol, the hash function selected to hash the exponentials should produce a result smaller than the maximum size of input allowed to the signature process, to allow redundancy to be added to the hash result before signing.

The proposed station-to-station protocol satisfies the above principles, as well as the desirable properties noted in Section 4 (perfect forward secrecy, direct authentication, no requirement of timestamps). Its compatibility with the emerging ISO authentication protocol, and its ability to provide key establishment within this framework, add to its appeal. Furthermore, the station-to-station protocol uses the minimum number of messages required for random-number-based challenge-response mutual authentication (three), and requires only one signature generation, one signature verification, and two encryption operations by each party (with an additional signature verification if certificates are used on a per-run basis to bind a user's identity and public key).

Any appropriate signature scheme may be used in the STS protocol, including the Digital Signature Algorithm (DSA) recently proposed by NIST [DSA91]. For reasons of practical efficiency, an obvious candidate signature scheme is RSA [Riv78]. Similarly, any appropriate symmetric encryption algorithm may be used. In some applications it may be desirable to avoid the use of an encryption algorithm. One method to consider for avoiding the need for an encryption algorithm E_K is as follows: replace the encrypted signature by a signature plus a message authentication code (MAC) over the signature; i.e., replace $E_K(s)$, where $s = s_B\{\alpha^y, \alpha^x\}$ (as in Section 5.1), by $(s, M_K(s))$, where M_K is a MAC with key K. The receiving party would then verify both the signature and the MAC over the signature. While allowing one to avoid the requirement of an encrypt/decrypt capability (which e.g., both Kerberos and the X.509 protocols require), a disadvantage of this approach is the additional data transfer it entails.

References

[9798-3] *Information Technology - Security Techniques. Entity Authentication Mechanisms - Part 3: Entity Authentication Using a Public-Key Algorithm (CD 9798-3)*, Nov. 1991 (ISO/IEC JTC1/SC27 Committee Draft #4).

[Baus89] F. Bauspieß and H.-J. Knobloch "How to keep authenticity alive in a computer network", *Advances in Cryptology - Eurocrypt 89,* J.-J. Quisquater and J. Vandewalle (eds.), *Lecture Notes in Computer Science* **434**, 38-46, Springer-Verlag (1990).

[Bell90] S.M. Bellovin and M. Merritt, "Limitations of the Kerberos authentication system", *ACM Computer Communication Review* **20** No.5 (1990), 119-132.

[Beng91] S. Bengio, G. Brassard, Y.G. Desmedt, C. Cloutier, and J.-J. Quisquater, "Secure implementation of identification system", *J. Cryptology* 4 (3), 1991, pp. 175-183.

[Bird91] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, M. Yung, "Systematic design of two-party authentication protocols", *Advances in Cryptology - Crypto '91*, Springer-Verlag (to appear).

[Bras88] G. Brassard, *Modern Cryptology, Lecture Notes in Computer Science* 325. Berlin/New York: Springer-Verlag, 1988.

[Burr90] M. Burrows, M. Abadi, R. Needham, "A logic of authentication", *ACM Transactions on Computer Systems* **8** No.1 (Feb. 1990), 18-36.

[Den81] D.E. Denning and G.M. Sacco, "Timestamps in key distribution protocols", *Comm. ACM* **24** No. 8 (Aug. 1981), 533-536.

[Diff76] W. Diffie and M.E. Hellman, "New directions in cryptography", *IEEE Trans. Info. Theory* **IT-22** No.6 (Nov. 1976), 644-654.

[DSA91] (proposed U.S. FIPS) Digital Signature Standard (DSS), *Federal Register*, vol.56, no.169 (Aug. 30 1991), 42980-42982.

[ElGa85] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Trans. Info. Theory* **IT-31** No.4 (July 1985), 469-472.

[Fiat87] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems", *Advances in Cryptology - Crypto 86,* A. Odlyzko (ed.), *Lecture Notes in Computer Science* **263**, 186-194, Springer-Verlag (1987).

[Gaar91] K. Gaarder and E. Snekkenes, "Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol", *J. Cryptology* **3** No.2 (1991), 81-98.

[Guil88] L.C. Guillou and J.-J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessing minimizing both transmission and memory", *Advances in Cryptology - Eurocrypt '88*, C.G. Günther (ed.), *Lecture Notes in Computer Science* **330**, 123-128, Springer-Verlag (1988).

[Gun89] C.G. Günther, "An identity-based key-exchange protocol", *Advances in Cryptology - Eurocrypt 89,* J.-J. Quisquater and J. Vandewalle (eds.), *Lecture Notes in Computer Science* **434**, 29-37, Springer-Verlag (1990).

[Habe91] S. Haber and W.S. Stornetta, "How to time-stamp a digital document", *J. Cryptology* **3** No.2 (1991), 99-111.

[IAns90] C. I'Anson and C. Mitchell, "Security defects in CCITT Recommendation X.509 - The Directory Authentication Framework", *Computer Communication Review* **20** No.2 (Apr. 1990), 30-34.

[Kohl91] J. Kohl and B.C. Neuman, "The Kerberos network authentication service", MIT Project Athena Version 5, 1991.

[Mit89] C. Mitchell, "Limitations of challenge-response entity authentication", *Electronics Letters* **25** No.17 (Aug. 1989), 1195-1196.

[Moor88] J.H. Moore, "Protocol failures in cryptosystems", *Proc. of the IEEE* **76** No.5 (May 1988), 594-602.

[OHig87] B. O'Higgins, W. Diffie, L. Strawczynski, R. de Hoog, "Encryption and ISDN - A Natural Fit", *1987 International Switching Symposium* (ISS 87).

[Okam89] E. Okamoto and K. Tanaka, "Key distribution system based on identification information", *IEEE J. Selected Areas in Comm.* **7** No.4 (May 1989), 481-485.

[Odly84] A.M. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance", *Advances in Cryptology - Eurocrypt 84,* T. Beth, N. Cot and I. Ingemarsson (eds.), *Lecture Notes in Computer Science* **209**, 224-314, Springer-Verlag (1985).

[Odly91] B.A. LaMacchia and A.M. Odlyzko, "Computation of Discrete Logarithms in Prime Fields", *Designs, Codes and Cryptography* **1** No.1 (May 1991), 47-62.

[Poh78] S.C. Pohlig and M. Hellman, "An Improved Algorithm for Computing Logarithms over $GF(p)$ and its Cryptographic Significance", *IEEE Transactions on Information Theory* **IT-24** (1978), 106-110.

[Riv78] R.L. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Comm. ACM* **21** (1978), 120-126.

[Riv84] R.L. Rivest and A. Shamir, "How to expose an eavesdropper", *Comm. ACM* **27** No.4 (Apr.1984), 393-395.

[Schn89] C.P. Schnorr, "Efficient identification and signatures for smart cards", *Advances in Cryptology - Crypto 89,* G. Brassard (ed.), *Lecture Notes in Computer Science* **435**, 239-251, Springer-Verlag (1990).

[Sham84] A. Shamir, "Identity-based cryptosystems and signature schemes", *Advances in Cryptology - Crypto 84,* G.R. Blakley and D. Chaum (ed.), *Lecture Notes in Computer Science* **196**, 47-53, Springer-Verlag (1985).

[X509] *CCITT Blue Book Recommendation X.509*, *The Directory-Authentication Framework*, Geneva, March 1988; also *ISO 9594-8*.