8 Key Exchange and Authentication Protocols

8.1 Introduction

Introduction
In this section, we are going to explore protocols that solve two general problems:

- Entity Authentication
- Key-Exchange

An entity authentication protocol allows an entity to ensure that 1 or more other entities are, in fact, who they say they are.
A key exchange protocol allows 2 or more entities to agree on a common key or keys that can used for some communication.
We will describe security protocols in isolation, but in practice they will be part of larger communications protocols that achieve some useful function (e.g., share files).

Introduction
In a typical scenario we might have 2 entities who wish to use a communications protocol to perform some task.
This communications protocol would have a number of phases:

1. Authentication Phase during which each entity securely establishes the identity of the other entity.
2. Key-Exchange Phase during which the entities agree on a symmetric session key to be used to encrypt the data during the remainder of the communication.
3. Data Transfer Phase in which the application data is encrypted using the agreed key(s) and exchanged between the 2 entities.

Introduction
To define a security protocol we need:

1. A description of the objectives of the protocol, including:
   - The entities involved.
   - The types of cryptography to be used and the details of existing keys that entities may possess.
   - The security objectives to be achieved.
2. A rigorous description of the steps making up the protocol.
   - For each step we will define the entities involved and the data exchanged.
   - We will also describe what computations the entities need to perform.
3. An analysis of the security of the protocol, including:

- The general behaviour of the protocol.
  - Here we will assume perfect cryptography.
- The key usage properties.
- What type(s) of cryptographic attacks does the protocol permit.
  - We will not look at these attacks in any detail, but they may be important when we are picking an actual cipher.

8.2 Notation

Notation
When describing protocols, we use the following abstract notation to represent data in messages sent during protocol exchanges:

$A, B, \ldots$	The parties or entities taking part in a protocol.
K	Symmetric Key
$K_{A,S}$	Symmetric Key shared between A and S
$d_1, d_2, \ldots, d_n$	Concatenated Data
$\{d_1, d_2, \ldots, d_n\}_K$	Encrypted Data
$H(d_1, d_2, \ldots, d_n)$	Hashed Data
N	Nonce

Notation

T	Timestamp
P	Password
K^-	Private Key
K_A^-	Private Key for entity A
K^+	Public Key
K_A^+	Public Key for entity A
$\{\vert d_1, d_2, \ldots, d_n \vert\}_{K_A^-}$	Signature
$C[S, I]$	S's certificate issued by I i.e. $\{\vert S, I, K_S^+, \ldots \vert\}_{K_I^-}$

Notation
Each step in a protocol is represented as a message sent from one entity to another:

$$N.\ A \rightarrow B : Message$$

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, N_B, \{N_A\}_{K_{A,B}}$

8.3 Confidentiality

Confidentiality
Before looking at authentication and key-exchange protocols, we will look at some simple protocols for the confidential transfer of data.
Protocol 1
Objectives:

- A (Alice) wishes to send a message M confidentially to B (Bob) using a symmetric cipher.
- We assume that A and B share a secret key $K_{A,B}$.

Protocol Steps:

1. $A \rightarrow B : \{M\}_{K_{A,B}}$

Confidentiality
General Analysis:

- In addition to A and B, we must consider other entities:
  - E (Eve) - a malicious active attacker who is both reading and modifying messages being transferred.
  - T (Trevor) - a trusted third party not directly involved in the exchange.
- The only way that E can read or modify messages is by knowing the key $K_{A,B}$.
- However, E can attack the protocol in a number of other ways:
  - She can delete messages - denial of service.
  - She can replay messages, i.e., she can capture a message and send it again.
  - If the plaintext of messages does not contain a certain amount of redundancy, she can send arbitrary data as a message.
- If B receives ciphertext $\{M\}_{K_{A,B}}$, and B did not produce it, then it must have been produced by A.

Confidentiality
Does this provide entity authentication?

- It does provide message origin authentication.
- It does not provide entity authentication as E can replay messages.

Does this provide message non-repudiation?

- There is no way that B can prove to T that a particular message came from A.
- B could always have constructed the message and encrypted it himself.

Confidentiality
Key Usage Analysis:

- We assume that Alice and Bob have a shared key $K_{A,B}$.
  - How exactly, do Bob and Alice agree on a shared key?
  - They need to use an out-of-band exchange or use some suitable key-exchange protocol.
- If the key $K_{A,B}$ is compromised then all messages encrypted with $K_{A,B}$ can be decrypted and messages can be falsified.
  - This includes messages that were sent in the past.
- Protocols that use symmetric keys normally require separate keys for each pair of users.
  - If there are N users, we need $\frac{N(N-1)}{2}$ keys.
  - As N grows, it becomes extremely difficult to manage all these keys.

Confidentiality
Cryptographic Analysis:

- This protocol allows a ciphertext only attack.
- However, if Eve can persuade Alice to reveal some messages and/or send particular messages, then various plaintext attacks may be possible.

Confidentiality
Protocol 2
Objectives:

- A (Alice) wishes to send a message M confidentially to B (Bob) using an asymmetric cipher.
- We assume that B's public key K_B^+ is available to A.

Protocol Steps:

1. $A \rightarrow B : \{M\}_{K_B^+}$

Confidentiality
General Analysis:

- The only way that Eve can read messages is by knowing Bob's private key K_B^-.
  - Even though Eve knows Bob's public key K_B^+, she still cannot read messages.
- As before, Eve can delete messages, replay messages etc..
- However, Eve can create new messages using Bob's public key K_B^+.
  - Therefore, we have no authentication properties as anyone can produce valid messages.
- It should also be noted that asymmetric cryptography is orders of magnitude slower that symmetric cryptography.

Confidentiality
Key Usage Analysis:

- Public keys are intended to be known by everyone.
- However, we still need to ensure that the correct public key is associated with an entity.
  - If we don't, E may be able to mount a man-in-the-middle attack.
  - We can avoid this by using a PKI.
- With this protocol each entity must have a key-pair, but since every other entity can use the same public key to communicate with (say) B, the key management problem is much simpler.
  - If there are N users we only need N public keys.

Cryptographic Analysis:

- Since we are using an asymmetric cipher, this protocol allows an adaptive chosen plaintext attack.

8.4 Entity Authentication (Identification)

Entity Authentication (Identification)
The problem is for one entity (the verifier) to be sure that another entity (the claimant) is who they say they are and hence prevent impersonation.
With Message Origin Authentication, a user can simply digitally sign data so that its origin can be determined, but there is no way of knowing when the data was signed.
Here, however, we are interested in real-time communications and ensuring that we have a valid channel to the other entity as we execute the protocol.

- In particular, we require the entity that is authenticated to be an active participant in the protocol as it is executed.

Entity Authentication (Identification)

In general, authentication is based on the claimant supplying information to the verifier which only the claimant knows or can compute.

- The obvious approaches would be for the claimant to use a password or to compute information using a cryptographic key.

One-way or unilateral authentication is when only one entity authenticates itself to the other entity.
With two-way or mutual authentication, each entity authenticates itself to the other entity.

Entity Authentication (Identification): Goals

Consider an honest entity A authenticating itself to B and a dishonest entity E trying to impersonate A:

1. A should be able to authenticate itself to B, i.e. B should accept A's claim to be A.
2. It should be computationally infeasible for E to have B accept its claim to be A, i.e. B should reject the claim and terminate the protocol.
3. No matter how many exchanges between A and B that E witnesses, it should remain computationally infeasible for E to impersonate A.
4. The information supplied to B by A should not allow B to impersonate A to a third party T, i.e. authentication should not be transferable.
5. Any authentication protocol needs to exhibit computational and communication efficiency.

Attacks on Authentication Protocols

The main purpose of attacking an authentication protocol is to allow a dishonest entity to impersonate a legitimate user.
We will consider a number of different kinds of attacks and see examples of such attacks on the protocols we describe.

1. Replay attacks in which information used in one instance of the protocol is captured and replayed at a later stage.
2. Reflection attacks in which an adversary sends information back to the originator of the information.
3. Interleaving attacks in which an adversary combines information from a number of interleaved instances of the protocol.
4. Chosen Plaintext cryptographic attacks in which an adversary can get one of the entities to encrypt arbitrary pieces of text. This gives a cryptanalyst more opportunity to discover a key.
5. Man-in-the-Middle attacks in which an adversary sits and (potentially) manipulates all communications.

Attacks on Authentication Protocols

There are a number of other issues that we need to consider.

1. Typically, authentication occurs as the first steps of a larger protocol and is only valid at a given instant of time. If ongoing assurance of identity is required, then additional techniques (e.g. encrypting all subsequent data with a session key) are required.

2. Since entity authentication takes place in real-time, the amount of computing power available to an attacker is more limited than with an off-line attack. However, an adversary might be able to gather information during a successful authentication and compute information off-line that can be used later.

3. An attacker may be able to mount a denial of service attack by simply delaying protocol messages.

Weak Authentication using Passwords

The simplest way to implement authentication is to use passwords, i.e. knowledge of the password is proof of identity.
A claimant simply supplies their identity and password, and the verifier compares this with a stored copy of the password to authenticate the claimant.
Protocol 3
Objectives:

- A client C wishes to authenticate itself to a server S using a password.

Protocol Steps:

1. $C \rightarrow S : C, P_C$

Weak Authentication using Passwords

Analysis: This protocol is extremely weak.

1. The password is passed as plaintext, therefore, unless we use a secure channel, anyone can observe the password and capture it for future use.

2. Therefore, this protocol is subject to replay attacks.

3. S must maintain a table of passwords and these need to be protected from disclosure. This is a problem with all password-based schemes.

Weak Authentication using Passwords

There are situations where simple passwords are widely used:

1. For logging onto computers.

2. For logging onto websites; typically using a confidential SSL connection.

Both of these uses are acceptable, provided some care is taken.

1. Passwords need to be chosen so as to avoid dictionary attacks.
2. The computer or website should only permit a small number of logon attempts or take progressively longer to respond to logon attempts.
3. Users should change their passwords frequently.
4. Users should not use the same password with different computers and websites.
    - If a user uses the same password with a number of computers or sites, then a dishonest administrator could impersonate the user.

Weak Authentication using Passwords
Typically, passwords are used as one-way authentication mechanisms.

- A user (client) authenticates itself to a computer (server) and they are then granted access rights on that computer (server).

In theory, we could extend the use of passwords to supply two-way authentication.

1. $A \rightarrow B : A, P_A$
2. $B \rightarrow A : B, P_B$

In situations where passwords are used, it is normally too difficult for both parties to maintain lists of passwords.

Weak Authentication using Passwords
Instead of exchanging a password as plaintext, it is possible for the claimant to send a digest of the password.
The verifier can then compute its own copy of the digest and compare it to the value sent from the claimant.

1. $C \rightarrow S : C, H(P_C)$

Of course, this is no more secure as an adversary could copy $H(P_C)$ and mount a replay attack.
`x.509` defines an authentication protocol which computes a new digest for every authentication exchange. Freshness is guaranteed by using a timestamp and/or a nonce.

Nonces and Timestamps
A nonce is a value that is used only once.
The main use of a nonce is to distinguish one instance of a protocol message from another.
In particular, if a message does not naturally contain information that varies from instance to instance, then a nonce can be added as an extra field.
The value of a nonce does not matter; it only needs to be different from instance to instance.
However, we may need some way of testing that a nonce has not been used before. In particular, when we use a nonce to prevent a replay attack, we need some way of knowing that we have handled a message with a particular nonce value already.

Nonces and Timestamps

There are a number of different ways of generating nonces:

1. Picking a random number.
   - If we pick random numbers from a sufficiently large space (e.g. $0..2^{64}$), then the chances of using the same value twice are very small.
   - However, if we need to detect reuse, then all previous nonces would need to be stored.
2. Using a monotonically increasing sequence number, e.g. $0, 1, 2, \ldots$.
   - Monotonically increasing ensures that each new nonce is greater than the previous nonce.
   - To detect reuse, we only need to remember the last sequence number used.
3. Using the time of day to a suitable accuracy.
   - This requires the entities to have synchronized clocks.
   - Times are monotonically increasing and detecting replays is relatively easy.

Nonces and Timestamps

In addition to nonces, we sometimes use timestamps.
Unlike a nonce (and indeed time used as a nonce), timestamps convey information about when an event occurred, so the value is important.
Protocol 4: `X.509` Protected Password Authentication
Objectives:

- A client C wishes to authenticate itself to a server S using a password.

Protocol Steps:

1. $C \rightarrow S : C, T, N, H(C, T, N, P_C)$

Nonces and Timestamps

Analysis:

- Given this exchange, S computes the digest $H(C, T, N, P_C)$ using its copy of P_C and compares it with the value sent in the protocol message. In particular, S uses the values T and N sent in the protocol message when computing its copy of the digest.
- This protocol uses both a timestamp T and nonce N. Using only a timestamp requires accurately synchronized clocks and using only nonces may require S to store large numbers of old nonces.
- C must ensure that the timestamp T and nonce N are chosen in such a way as to avoid the possibility of a replay.

- An adversary capturing the digest $H(C,T,N,P_C)$ is unable to replay the message as S will detect the duplicated nonce or a timestamp that is too old.
- This protocol can easily be extended to support two-way authentication.

Authentication Using Cryptography
In the previous protocols a claimant's ability to produce a password was used to authenticate the claimant.
An alternative approach is to have the claimant demonstrate that they have access to a cryptographic key. This is achieved by requiring the claimant to compute some value using the key.
This can be done using a challenge-response protocol.
Protocol 5
Objectives:

- A client C wishes to authenticate itself to a server S.

Protocol Steps:

1. $S \rightarrow C : N$
2. $C \rightarrow S : C, \{N\}_K$

Authentication Using Cryptography
Analysis:

- The verifier S sends a challenge (nonce) N to the claimant and the claimant returns $\{N\}_K$, i.e. the challenge encrypted with some key K.
  - Provided that C is the only user with access to K, then S knows that C is participating in the protocol.
- Is there any need for C and/or S to record nonces so as to prevent a replay attack?
- How can the nonce be generated?
- Is this protocol subject to a man-in-the-middle attack?
- This basic challenge-response protocol can be extended to implement two-way authentication.

Authentication Using Symmetric Ciphers
Protocol 6
Objectives:

- A client C wishes to authenticate itself to a server S using a shared secret key $K_{S,C}$.

Protocol Steps:

1. $S \rightarrow C : N$
2. $C \rightarrow S : C, \{N\}_{K_{S,C}}$

Authentication Using Symmetric Ciphers

Analysis:

- Since S shares a key with C, it can decrypt the ciphertext received from C and ensure it is valid.
- Like password-based schemes, S may need to maintain a database of shared keys. As with passwords, this is vulnerable to disclosure.
- Again, like password-based schemes, if C was foolish enough to use the same shared key with a number of servers, then S could impersonate C to other servers.

Authentication Using Symmetric Ciphers

Cryptographic Analysis:

- This protocol exposes the cipher to an adaptive chosen plaintext attack.
- Adaptive chosen plaintext attacks are a problem for challenge-response protocols, i.e. the verifier can pick any nonce N that they want.
  - An attacker could repeatedly perform steps 1 and 2 of the last protocol above.
  - Each time they would pick a value for N that would assist in attacking the shared key.

Authentication Using Symmetric Ciphers

There are two ways of reducing this vulnerability:

- The claimant can add additional information to the data to be encrypted. This means that the verifier has only partial control over what data is encrypted.
  1. $S \rightarrow C : N_S$
  2. $C \rightarrow S : C, \{N_S, N_C, S\}_{K_{S,C}}$
- The claimant can select the nonce. For example, the protocol presented earlier for a protected password can be extended to use symmetric keys.
  1. $C \rightarrow S : C, T, N, \{C, T, N, S\}_{K_{S,C}}$

Authentication Using Symmetric Ciphers

We can extend this protocol to allow mutual authentication.

Protocol 7

Objectives:

- Entities A and B wish to mutually authenticate each other using a shared secret key $K_{A,B}$.

Protocol Steps:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, N_B, \{N_A\}_{K_{A,B}}$
3. $A \rightarrow B : \{N_B\}_{K_{A,B}}$

Authentication Using Symmetric Ciphers
Analysis:

- Unfortunately, this protocol is subject to a reflection attack.
- Consider, a dishonest entity E who wishes to impersonate A to B.
  - First E starts one instance of the protocol with B.
    1. $E(A) \rightarrow B : A, N_A$
    2. $B \rightarrow E(A) : B, N_B, \{N_A\}_{K_{A,B}}$
  - At this stage, E is stuck as she cannot encrypt N_B.
  - However, E can start a second instance of the protocol and get B to encrypt N_B.
    3. $E(A) \rightarrow B : A, N_B$
    4. $B \rightarrow E(A) : B, N_{B'}, \{N_B\}_{K_{A,B}}$
  - E abandons the second instance and resumes the first instance.
    5. $E(A) \rightarrow B : \{N_B\}_{K_{A,B}}$
- It is important to note that this is an attack on the protocol and not the cryptography.

Authentication Using Symmetric Ciphers
There are a number of ways of preventing this attack.

- A and B could use different keys.
  - Therefore, a nonce encrypted by B could not be sent back to B as it would be encrypted with the wrong key.
- The initiator could prove its identity first:
  1. $A \rightarrow B : A$
  2. $B \rightarrow A : B, N_B$
  3. $A \rightarrow B : N_A, \{N_B\}_{K_{A,B}}$
  4. $B \rightarrow A : \{N_A\}_{K_{A,B}}$

  In this case B does not encrypt anything until it has authenticated A.

Authentication Using Symmetric Ciphers

- The nature of the challenges produced by A and B could be different, i.e. a challenge encrypted by B could not be confused as a challenge to be encrypted by A.
- For example, this can be achieved by including different information in the ciphertext sent between the parties:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, N_B, \{N_A, A\}_{K_{A,B}}$
3. $A \rightarrow B : \{N_B\}_{K_{A,B}}$

Authentication Using Asymmetric Ciphers

A challenge-response protocol can also be used with asymmetric ciphers. There are two basic modes of operation:

- The claimant encrypts the challenge with their private key.
- The verifier S encrypts a challenge using the claimant's public key K_C^+ and the claimant proves that they have the private key by decrypting the challenge and returning it to the verifier.

Protocol 8
Objectives:

- A client C wishes to authenticate itself to a server S using its public key K_C^+.

Protocol Steps:

1. $S \rightarrow C : N$
2. $C \rightarrow S : C, \{N\}_{K_C^-}$

Authentication Using Asymmetric Ciphers

Analysis:

- Since S knows C's public key K_C^+, it can decrypt the ciphertext received from C and ensure it is valid.
- If we use a PKI, then key management becomes relatively simple, i.e. S just needs to obtain a certified copy of C's public key from the PKI.
- If required, certificates can be exchanged as part of the protocol.
- It is unwise to use key-pairs for more than one purpose.
  - For this protocol, if S challenges C with a value $H(m)$, then C will encrypt it with its private key and hence produce a digital signature for m.
  - Thus, if the same key-pair is used for producing digital signatures, C has unwittingly signed some unknown document m.

Authentication Using Asymmetric Ciphers
Cryptographic Analysis:

- This protocol would allow an attacker E to mount an adaptive chosen plaintext attack against encryption with a private key.
- We can use some of the techniques as described for symmetric ciphers to prevent such attacks.

Authentication Using Asymmetric Ciphers
Protocol 9
Objectives:

- A client C wishes to authenticate itself to a server S using its public key K_C^+.

Protocol Steps:

1. $S \rightarrow C : \{N\}_{K_C^+}$
2. $C \rightarrow S : N$

Authentication Using Asymmetric Ciphers
Analysis:

- S encrypts the nonce N with C's public key. Therefore if N is returned in step 2, S knows that C must have participated in the protocol.

Cryptographic Analysis:

- This protocol would allow an attacker E to mount an adaptive chosen ciphertext attack against encryption with a public key.

Needham-Schroeder Public-Key Protocol
There are two protocols due to Needham-Schroeder; this one is based on the use of asymmetric encryption, the other one (which we will look at later) is based on symmetric encryption.
Protocol 10: Needham-Schroeder Public-Key (NSPK) Protocol
Objectives:

- Entities A and B wish to mutually authenticate each other using their public keys K_A^+ and K_B^+.

Protocol Steps:

1. $A \rightarrow B : \{N_A, A\}_{K_B^+}$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A^+}$
3. $A \rightarrow B : \{N_B\}_{K_B^+}$

Needham-Schroeder Public-Key Protocol
Analysis:

- Since the ciphertext sent in step 1 is encrypted by B's public key, only B can discover N_A.
- Therefore, if A finds N_A in the message sent in step 2, she must be communicating with B.
- Similarly, only A can discover N_B and return it in step 3.
- If necessary, this protocol can be extended so that A and B exchange certificates.

Needham-Schroeder Public-Key Protocol
Cryptographic Analysis:

- In 1995, Gavin Lowe discovered an interleaving attack on the NSPK Protocol by using a model checking tool called FDR.
- If A and B can be trusted, this protocol appears to be secure.
- However, E can interleave two runs (instances) of this protocol and transfer authentication, i.e. she can impersonate A to B.

$$\begin{array}{llllll}
1. & A \rightarrow E: & \{N_A, A\}_{K_E^+} & & & \\
 & & & 1'. & E(A) \rightarrow B: & \{N_A, A\}_{K_B^+} \\
 & & & 2'. & B \rightarrow E(A): & \{N_A, N_B\}_{K_A^+} \\
2. & E \rightarrow A: & \{N_A, N_B\}_{K_A^+} & & & \\
3. & A \rightarrow E: & \{N_B\}_{K_E^+} & & & \\
 & & & 3'. & E(A) \rightarrow B: & \{N_B\}_{K_B^+}
\end{array}$$

Needham-Schroeder Public-Key Protocol

- So at this point, A believes that she is communicating with E and that only she and E share the nonces N_A and N_B.
- In fact, she is communicating with E, but the nonces N_A and N_B are also known to B.
- B believes he is communicating with A, but he is in fact communicating with E.
- Note that this attack was achieved without attacking the cryptography, it is achieved by simply manipulating the protocol.

Needham-Schroeder Public-Key Protocol
This protocol is fixed by Lowe to prevent this attack as follows:

1. $A \rightarrow B : \{N_A, A\}_{K_B^+}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A^+}$
3. $A \rightarrow B : \{N_B\}_{K_B^+}$

- The message modified by this new version of the protocol is the only one which cannot be decrypted by E in the previous attack.
- So E will not be able to change B to E.
- A will now be able to see that the received message does not come from E but from B, so will not send N_B to E.
- Thus, E is unable to discover the value of N_B.

8.5 Key-Exchange

Key-Exchange
The Problem:

- Symmetric cryptography is much more efficient than asymmetric cryptography.
  - Therefore, when exchanging large amounts of data we prefer to use symmetric cryptography to provide confidentiality.
- Handling large numbers of symmetric keys is problematic.
- Therefore, we would like some way of establishing or exchanging session keys that we can use for a limited period of time and then destroy.
  - With real-time protocols we will not even want to store the session keys.
- Clearly we will want to protect the key agreement or exchange so that an observer cannot determine the session keys.

The Diffie-Hellman Key-Agreement Protocol
For Diffie-Hellman, we select a large prime p and a generator g, then:

1. A chooses a random x such that $1 < x < p-1$.
2. $A \rightarrow B : g^x \pmod{p}$
3. B chooses a random y such that $1 < y < p-1$.
4. $B \rightarrow A : g^y \pmod{p}$
5. A computes $K = (g^y)^x \pmod{p}$.
6. B computes $K = (g^x)^y \pmod{p}$.
7. A and B now share the secret K.

User-Generated Keys
An alternative to key-agreement, is for one entity A to generate a session key K, which is then transferred confidentially to another entity B.

- Of course, for this to work there must be existing keys that can be used to encrypt this session key.

With an asymmetric cipher A can simply encrypt K with B's public key.

1. $A \rightarrow B : \{K\}_{K_B^+}$

User-Generated Keys
With symmetric ciphers the situation is somewhat more complex.

- In particular, each entity will need existing symmetric keys and we have a potential key management problem.
- As we will see, this problem can be overcome by using a trusted server.

Allowing one entity to generate the session key is a potential weakness.

- The entity may use an existing key or generate a weak key.
- With the symmetric protocols we will consider, this problem is overcome by having the session key generated by a trusted server.

8.6 Authentication and Key-Exchange

Authentication and Key-Exchange
Earlier we noted that an authentication protocol gave assurance of identity at a single instant in time.
For a communications protocol we would like some sort of ongoing authentication that lasts for the lifetime of a connection.
An obvious approach is to combine authentication and key exchange into a single protocol.

- Not only are the entities authenticated, but the session key is bound to the authentication process.
- Then, provided future exchanges are encrypted with the session key, we have ongoing authentication.

Symmetric Ciphers
Earlier we looked at how two entities A and B could perform mutual authentication using a shared secret key.
This approach requires each distinct pair of users to share a separate key and with a large community of users this becomes unmanageable. An alternative approach is to use a trusted server S to authenticate users and issue session keys. With this approach:

- Each user need only maintain a single shared key with the server.
- The server generates the session key and is trusted to behave properly.

There a number of protocols that are based on this basic structure, we will consider the Needham-Schroeder Secret-Key Protocol and Otway-Rees.
We will also look briefly at Kerberos, a complete system based on the Needham-Schroeder Secret-Key Protocol.

Needham-Schroeder Secret-Key Protocol
Let us assume that:

- We have a trusted server S.
- Two users A and B who share secret keys $K_{A,S}$ and $K_{B,S}$ respectively with S.

Protocol 11: Needham-Schroeder Secret-Key (NSSK) Protocol
Objectives:

1. To allow A and B to be authenticated.
2. To allow S to generate a session key $K_{A,B}$ that is bound to the authentication process and is only known to A, B and S.

Needham-Schroeder Secret-Key Protocol
Protocol Steps:

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A : \{N_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\}_{K_{A,S}}$
3. $A \rightarrow B : \{K_{A,B}, A\}_{K_{B,S}}$
4. $B \rightarrow A : \{N_B\}_{K_{A,B}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{A,B}}$

Needham-Schroeder Secret-Key Protocol

1. A sends its identity, the identity B of the entity it wishes to communicate with and a nonce N_A to the server S.
2. S performs the following steps:
   - It generates a session key $K_{A,B}$ for use by A and B.
   - It constructs B's token $\{K_{A,B}, A\}_{K_{B,S}}$ that only B can decrypt.
   - It constructs A's token $\{N_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,S}}\}_{K_{A,S}}$ that contains B's token and that only A can decrypt.
   - It sends A's token to A.

3. A performs the following steps:
   - It decrypts its token received from S using the key $K_{A,S}$ that it shares with S.
   - It checks the nonce N_A and name B in the token are the same as those sent to S in step 1.
   - It records the session key $K_{A,B}$ for future use.
   - It relays B's token to B.

Needham-Schroeder Secret-Key Protocol

4. B performs the following steps:
   - It decrypts its token received from S via A using the key $K_{B,S}$ that it shares with S.
   - It records the session key $K_{A,B}$ for future use.
   - It challenges A by sending a nonce N_B encrypted using the session key $K_{A,B}$.
5. A performs the following steps:
   - It decrypts the message sent from B using the session key $K_{A,B}$ and checks that the result is the nonce N_B. Note that this requires N_B to be redundant.
   - It then encrypts the value $N_B - 1$ using the session key and sends it to B.

Finally, B decrypts the message sent by A in step 5 and checks that the result is $N_B - 1$.

Needham-Schroeder Secret-Key Protocol
Analysis:

- This protocol provides mutual authentication of A and B, and allows a session key generated by S to be shared by A and B.
- The server S must be trusted; it has access to all keys.
- In step 1 we use a nonce N_A to prevent a replay attack i.e. A checks that N_A appears in the token returned by S in step 2.
- In step 1, A identifies itself and the entity B it wishes to communicate with. Since this message is not protected in any way, an attacker could substitute its own identity for B. Provided the attacker could also intercept messages 3-5, it could impersonate B to A.
  - This attack is prevented by S including the identity of the other party in A's token i.e. A checks that S has supplied information that allows A to authenticate B.
- After step 2, A knows that S has participated in this run of the protocol and that $K_{A,B}$ is fresh.

Needham-Schroeder Secret-Key Protocol

- After step 3, A and B have a shared session key $K_{A,B}$. Clearly, S knows this key, but because it was protected using $K_{A,S}$ and $K_{B,S}$, no other entity knows the key. We trust S not to misbehave.
- After step 3, B knows that $K_{A,B}$ is a key shared by A and itself. However, B does not know if $K_{A,B}$ is fresh.
- In steps 4 and 5, A and B perform mutual authentication i.e. they prove to each other that they have the session key and that they are participating in this run of the protocol.
  - The nonce N_B needs to be redundant so an attacker cannot replace the message in step 4 with some arbitrary ciphertext.
  - In step 5, A encrypts $N_B - 1$ since the encryption of N_B is already available.
- Note that A can repeat steps 3-5 as often as it wants i.e. it can use the same session key for multiple communications.

Needham-Schroeder Secret-Key Protocol
The Needham-Schroeder Secret Key Protocol was first published in 1978.
A significant feature is that it does not use timestamps and it is the basis for many of the server-based authentication and key distribution protocols developed after 1978.
It has been shown that the double encryption of B's token in step 2 is unnecessary. However, this is not a big issue.
However, serious weaknesses with the freshness of keys have been uncovered.

Needham-Schroeder Secret-Key Protocol
Attack 1: Compromised Session Keys

- Assume that an adversary E observes a legitimate protocol exchange between A and B.
- E can capture B's encrypted token in step 3 of the protocol.
- If E can compromise the session key $K_{A,B}$, then E can follow steps 3-5 of the protocol and impersonate A to B.
  - Note that E can take as long as it wants to compromise the session key.
  - So even though determining the session key may be computationally hard, the design of this protocol gives an adversary an unnecessarily long time to attempt to find the key.

Needham-Schroeder Secret-Key Protocol

Attack 2: Compromised Server/User Keys

- If the key $K_{A,S}$ that a user A shares with the server S is compromised, then an adversary E can use steps 1-2 of the protocol to obtain session keys and tokens for as many other users as it wishes.
- E can then impersonate A to one of these users at any time by completing steps 3-5 of the protocol.
- Even if A determines that the key $K_{A,S}$ has been compromised and generates a new key, E can still use the session keys and tokens that it obtained earlier.

Needham-Schroeder Secret-Key Protocol

- The problem with this protocol is that B has no way of knowing if the session key is fresh.
- There are two ways of resolving this problem:
  1. Have B interact with the server S so that it knows that the server is participating in the protocol.
     - In the existing protocol, steps 1-2 effectively authenticate S to A since S is encrypting fresh data with the shared secret key $K_{A,S}$.
     - This was done in an expanded version of the Needham-Schroeder protocol. However, we will look at an improved protocol due to Otway and Rees.
  2. Include timestamps so that B can ascertain the freshness of the session key.
     - Kerberos does this by adding a validity period or lifetime for each session key.

The Otway-Rees Protocol

Otway-Rees gives freshness guarantees without using synchronized clocks.
Again, let us assume that:

- We have a trusted server S.
- Two users A and B who share secret keys $K_{A,S}$ and $K_{B,S}$ respectively with S.

Protocol 12: Otway-Rees
Objectives:

1. To allow A and B to be authenticated.
2. To allow S to generate a session key $K_{A,B}$ that is bound to the authentication process and is only known to A, B and S.

The Otway-Rees Protocol

Protocol Steps:

1. $A \rightarrow B : N, A, B, \{N_A, N, A, B\}_{K_{A,S}}$
2. $B \rightarrow S : N, A, B, \{N_A, N, A, B\}_{K_{A,S}}, \{N_B, N, A, B\}_{K_{B,S}}$
3. $S \rightarrow B : N, \{N_A, K_{A,B}\}_{K_{A,S}}, \{N_B, K_{A,B}\}_{K_{B,S}}$
4. $B \rightarrow A : N, \{N_A, K_{A,B}\}_{K_{A,S}}$

Otway-Rees Protocol

Analysis:

- In this protocol both A and B perform mutual authentication with the server S and S returns the session key $K_{A,B}$ to each.
- In the case of A we have:
  - S can authenticate A by decrypting $\{N_A, N, A, B\}_{K_{A,S}}$ it receives in step 2.
  - A can authenticate S and obtain its copy of the session key by decrypting $\{N_A, K_{A,B}\}_{K_{A,S}}$ in step 4.
  - How does S ensure that the message from step 1 is not a replay?
    - Actually it can't!
    - However, this is okay, because an attacker replaying the message will be unable to recover the session key $K_{A,B}$ in step 4.
  - In this protocol, B relays messages to/from A. However, since these messages are encrypted using $K_{A,S}$, B cannot decrypt or modify them.
- In the case of B we have a similar argument with $\{N_B, N, A, B\}_{K_{B,S}}$ and $\{N_B, K_{A,B}\}_{K_{B,S}}$

Otway-Rees Protocol

- In step 1, A generates two nonces:
  - N_A which is used to prevent replay of S's message back to A and guarantee the freshness of the session key received from S.
  - The nonce N_B serves the same purpose for B.
  - N which acts as a transaction identifier that binds the information sent by A and B together:
    - S can check that $\{N_A, N, A, B\}_{K_{A,S}}$ and $\{N_B, N, A, B\}_{K_{B,S}}$ have the same value for N and therefore belong to the same transaction.
  - After the protocol has completed, A and B will share the key $K_{A,B}$.
    - If A and B use $K_{A,B}$ to encrypt a communication, then they both have ongoing mutual authentication.

Kerberos

Kerberos is a complete server-based authentication and key distribution system based on the Needham-Schroeder protocol. The main differences lie in the use of timestamps and lifetimes.

It was developed as part of project Athena at MIT and has been widely used in Unix networks. The current version is 5.1.

It has been incorporated into Windows 2000.

As with Needham-Schroeder, entities share secret keys with a trusted Kerberos server.

Kerberos

Kerberos uses its own terminology:

- The trusted server S is known as a Key Distribution Centre (KDC).
- When a client A wishes to use a computing service B, the KDC issues a ticket that can be used to access the services of B.
  - As we will see, this is essentially B's token in the original Needham-Schroeder protocol.
- Tickets have a specific lifetime, but they can be renewed, post-dated, etc.

A significant use of Kerberos is to allow human users to logon to computer networks and access computing services distributed throughout that network.

Kerberos also defines the precise syntax of the various protocols, but since we are only interested in an abstract view of Kerberos, we will ignore such detail.

Kerberos

The following is a much simplified view of the basic Kerberos protocol.

Here A is a client, B is a computer service and S is the Kerberos Server.

The client A wishes to use the service B. It does so by getting a ticket from the Kerberos server S. This ticket is then presented to the service B.

1. $A \rightarrow S : A, B, N_A$
2. $S \rightarrow A :$ $ticket_B, \{K_{A,B}, N_A, L, B\}_{K_{A,S}}$
   where $ticket_B = \{K_{A,B}, A, L\}_{K_{B,S}}$
   and L is the validity period (lifetime) of the key $K_{A,B}$.
3. $A \rightarrow B :$ $ticket_B, authenticator_A$
   where $authenticator_A = \{A, T_A\}_{K_{A,B}}$
4. $B \rightarrow A : \{T_A\}_{K_{A,B}}$

Kerberos

1. This step is the same as Needham-Schroeder.
    - As with Needham-Schroeder, S generates a session key $K_{A,B}$ for use by A and B.
    - It encrypts a copy of this key for A using their shared key.
    - It also encrypts a copy for B and this produces a so-called ticket. This is essentially the token used in Needham-Schroeder.
    - Note that the shared key has a lifetime L which is included in the encrypted information sent to both A and B.
2. This step is similar to Needham-Schroeder except that the ticket $ticket_B$ is conveyed separately and not as part of the ciphertext sent to A. A performs the following steps:
    - It decrypts its token using the key $K_{A,S}$.
    - It checks that the nonce N_A and name B in the token are the same as those sent to S in step 1.
    - It records the session key $K_{A,B}$ for future use.
    - It records the ticket $ticket_B$ so that it can send it to B whenever it wants to use the service offered by B.
    - A can use this ticket multiple times, but only during its lifetime L.

Kerberos

3. To access the services offered by B, A presents its ticket and an authenticator
    - Since the ticket $ticket_B$ is encrypted by the key shared between B and S, B can decrypt it, recover the session key $K_{A,B}$, the lifetime L during which the session key can be used and the identity A of the owner of the ticket.
    - The authenticator $authenticator_A$ allows B to authenticate A, i.e. that the entity has access to the shared key $K_{A,B}$.
    - The timestamp provides a freshness guarantee.
4. Finally, A can authenticate B.

Kerberos
Notes:

- After completing this exchange, A and B use the session key $K_{A,B}$ to keep their communications confidential.
- Since Kerberos is using timestamps and lifetimes, there is a need for secure, synchronized clocks. On a local area network (Unix, Windows 2000, Netware, ...) this is relatively easy to implement.
- As you would expect, the authenticator $authenticator_A$ and the ciphertext in step 4 have different formats.

Asymmetric Ciphers
The Needham-Schroeder Public-Key Protocol can be used to achieve both mutual authentication and key exchange.
This can be done by exchanging key material instead of nonces.
Here we look at a Diffie-Hellman exchange embedded in the protocol.

1. $A \rightarrow B : \{g^x \pmod{p}, A\}_{K_B^+}$
2. $B \rightarrow A : \{g^x \pmod{p}, g^y \pmod{p}\}_{K_A^+}$
3. $A \rightarrow B : \{g^y \pmod{p}\}_{K_B^+}$

Signing and Public-Key Encryption
An obvious approach is to use digital signatures for authentication and public-key encryption to protect session keys.
There are a number of ways this can be achieved.

i. The session key can be signed and then encrypted.

   1. $A \rightarrow B : \{K_{A,B}, T_A, \{\!|B, K_{A,B}, T_A|\!\}_{K_A^-}\}_{K_B^+}$

   The timestamp T_A ensures freshness and including B in the signed data prevents B from impersonating A to a third party.

ii. The session key can be encrypted and signed separately.

   1. $A \rightarrow B : \{K_{A,B}, T_A, \}_{K_B^+}, \{\!|B, K_{A,B}, T_A|\!\}_{K_A^-}$

   This only works if the signature scheme does not allow recovery, e.g. it uses a hash function.

iii. The session key can be encrypted and then signed.

   3. $A \rightarrow B : T_A, token, \{\!|B, T_A, token|\!\}_{K_A^-}$
      where $token = \{K_{A,B}\}_{K_B^+}$

These protocols can be extended to implement two-way authentication.

X.509

- X.509 defines a number of strong authentication protocols.
- X.509 uses slightly different terminology. In particular, it uses the terms one-way, two-way and three-way to specify the number of protocol steps.
- Like Kerberos, X.509 defines the detailed syntax of the protocol using ASN.1. We will ignore such detail here.
- The X.509 protocols support the optional exchange of certificates, but we will ignore this feature.

X.509 One-Way Authentication

Protocol 13: X.509 One-Way Authentication

Objectives:

1. To allow B to authenticate A, i.e. provide unilateral authentication.
2. To allow A to share a session key $K_{A,B}$ with B.

Protocol Steps:

1. $A \rightarrow B : token_A, \{|token_A|\}_{K_A^-}$

   where $token_A = T_A, N_A, B, data_A, \{K_{A,B}\}_{K_B^+}$

Notes:

- $data_A$ is some optional signed data that A can send to B.
- This protocol uses both timestamps and nonces to ensure freshness and prevent replays.

X.509 Two-Way Authentication

Objectives:

1. To allow A and B to mutually authenticate each other.
2. To allow A and B to establish or exchange session keys.

Protocol Steps:

1. $A \rightarrow B : token_A, \{|token_A|\}_{K_A^-}$

   where $token_A = T_A, N_A, B, data_A, \{X_A\}_{K_B^+}$

2. $B \rightarrow A : token_B, \{|token_B|\}_{K_B^-}$

   where $token_B = T_B, N_B, A, N_A, data_B, \{X_B\}_{K_A^+}$

X.509 Two-Way Authentication

Notes:

- This protocol allows two shared values X_A and X_B to be exchanged.
  - These could be symmetric keys to be used in different directions on the connection.
  - They could be combined into a larger key.
  - They could be used with Diffie-Hellman.
- Note that $token_B$ contains the nonce N_A sent by A.

X.509 Three-Way Authentication

Objectives:

1. To allow A and B to mutually authenticate each other.
2. To allow A and B to establish or exchange session keys.

Protocol Steps:

1. $A \rightarrow B : token_A, \{|token_A|\}_{K_A^-}$
2. $B \rightarrow A : token_B, \{|token_B|\}_{K_B^-}$
3. $A \rightarrow B : token'_A, \{|token'_A|\}_{K_A^-}$

   where $token'_A = N_B, B$

X.509 Three-Way Authentication

Notes:

- This also provides mutual authentication, but does so without the need for timestamps.
- Steps $1-2$ are the same as for two-way authentication but the timestamps may be set to zero and ignored.
- This protocol is now essentially a challenge-response protocol.
  - In particular, both entities exhibit ownership of a private key by signing data containing a nonce produced by the other entity.