Lecture 12: Key Establishment and Certificates

TTM4135

Relates to Stallings Chapters 15 and 16

Spring Semester, 2025

Motivation

- So far, we have looked at cryptographic primitives (symmetric key encryption, public key encryption, signature schemes)
- Now, we are going to look at how to use these to build more advanced cryptographic protocols
- Key management is an essential part of setting up secure communications
- Key establishment is the process of setting up cryptographic keys to protect a subsequent communication session
- Key establishment in TLS uses public keys to allow clients and servers to share a new communication key
- Digital certificates are a vital tool for key establishment based on public key cryptography

Outline

Key establishment principles

Key establishment types

- Session Key Distribution using Symmetric Keys
- Session Key Distribution using Asymmetric Keys

Key establishment using public key encryption

- Certificates and PKI

Key establishment using symmetric key encryption

- Needham-Schroeder protocol
- Kerberos

Key management

Key management is concerned with generation, distribution, protection, destruction, of cryptographic keys. It is a critical aspect of any cryptographic system and includes several phases:

- Key generation: ideally keys are random
- Key distribution: keys must be distributed in a secure fashion
- Key storage: keys must be accessible for use but not to unauthorised users
- Key destruction: removing a key from memory is not always easy

Types of keys

Long-term keys: (or static keys) intended to be used for a long period – depending upon the application this may be a few hours or a few months or a few years

Ephemeral keys: generated for single use and then deleted, Diffie–Hellman being a typical example

Session keys: intended to be used for one communication session – depending upon the application this may be a few seconds, a few hours or a day

Types of keys and usage

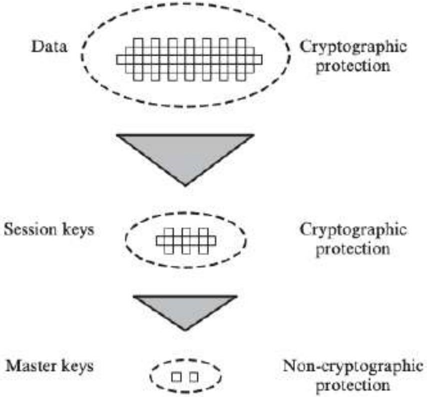


Figure 14.2 The Use of a Key Hierarchy

Stallings book, *seventh edition*. See Figure 15.2 in the *eight* edition.

Types of keys and usage

- Session keys are usually symmetric keys used to protect communications in a session with ciphers such as AES for authenticated encryption
- Session keys should make different sessions *independent* in the sense that compromise of one session key should not affect other sessions
- Long-term keys can be either symmetric or asymmetric keys depending on how they are used
- Typically long-term keys and ephemeral keys are used in establishment of session keys

Adversary capabilities

When analysing the security of key establishment protocol we assume an adversary that knows the details of the cryptographic algorithms involved (why?) and is able to:

- eavesdrop on all messages sent in a protocol
- alter all messages sent in a protocol using any information available
- re-route any message (including fabricated messages) to any user
- **obtain the value of the session key K_{AB} used in any previous run of the protocol**

Security goals

The security of key establishment protocols is defined by two properties

authentication: if a party A completes the protocol and believes (is convinced) that the session key K_{AB} is shared with party B, then K_{AB} should not be shared with a different party C

confidentiality: an adversary is unable to obtain the session key accepted by a particular party

- Authentication can be *mutual* (both parties achieve it) or *unilateral* (only provided on one side)
- Many real-world key establishment protocols achieve only unilateral authentication

Categories of key establishment protocols

- Three common approaches are:
  1. key pre-distribution where keys are set in advance
  2. key transport where one party chooses the key and distributes it
  3. key agreement where two or more parties contribute to the session key
- Each of these approaches can use symmetric-key cryptography or public-key cryptography

Key pre-distribution (pre-shared keys)

- A trusted authority (TA) generates and distributes long-term keys to all users when they join the system
- Simplest scheme assigns a secret key for each pair of users
  - But then the number of keys grows quadratically resulting in poor scalability
- TA only operates in the pre-distribution phase and need not be online afterwards

Session key transport with an online server

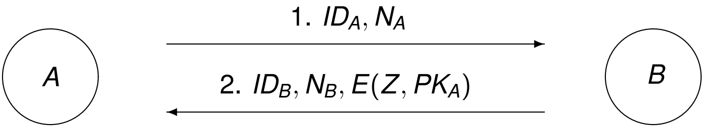
- TA shares a long-term shared key with each user
- TA generates and sends session keys to users when requested and protected by the long-term keys
- TA must be trusted and is a single point of attack
- Scalability can be a problem
- Kerberos (next few slides) is an example of this type of key establishment

- Typical advantages of using an online server are:
  - No need for computationally expensive public key algorithms.
  - Only a small amount of secure storage is required by each user to store the long-term key.
  - No certificate management overheads (distribution, validation, etc).
- Typical disadvantages include:
  - Requires that the TA be available online.
  - The TA is highly trusted and is a single point of attack. The security of the whole network depends on it.
  - Scalability can be a problem.

Key transport with asymmetric cryptography

- One user chooses key material and sends it encrypted with the other party's public key
- Each party can include a random *nonce* value to ensure that the key is new
- A key derivation function (KDF) binds the secret key material with other protocol elements to prevent certain attacks
- A standard KDF uses HMAC — think of a KDF as a hash function
- TLS up to version 1.2 includes this type of key establishment

Key transport protocol



- PK_A is A's public encryption key
- Z is a random value generated by B
- The session key can be

$$K_{AB} = KDF(Z, ID_A, ID_B, N_A, N_B)$$

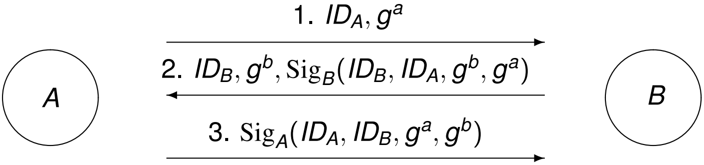
Key agreement

- Two parties each provide input to the keying material
- Usually add authentication with public keys, for example by signing the exchanged messages
- Diffie–Hellman protocol is a widely used key agreement protocol
- TLS includes Diffie–Hellman which is today the usual method of key establishment in TLS

Signed Diffie-Hellman

- A and B are two parties with identities ID_A, ID_B, who want to share a session key
- Computation takes place in a group G with generator g
- a, b are random values chosen by A and B in the range up to the order of G
- $\mathrm{Sig}_A(m)$ is a digital signature on message m by A
- $\mathrm{Sig}_B(m)$ is a digital signature on message m by B
- Both parties need each other's public signature verification keys

Signed Diffie–Hellman protocol



1. ID_A, g^a

2. $ID_B, g^b, \mathrm{Sig}_B(ID_B, ID_A, g^b, g^a)$

3. $\mathrm{Sig}_A(ID_A, ID_B, g^a, g^b)$

- *A* checks the signature received in step 2 and if it is valid *A* computes the shared secret: $Z = (g^b)^a = g^{ab}$
- Similarly *B* checks the signature received in step 3 and, if valid, computes the shared secret: $Z = (g^a)^b = g^{ab}$
- The session key can be

$$K_{AB} = KDF(Z, ID_A, ID_B, g^a, g^b)$$

Forward secrecy

- What happens when a long-term key is compromised?
- The attacker can now act as the owner of the long-term key
- Previous session keys may also be compromised

Forward secrecy definition

A key agreement protocol provides *(perfect) forward secrecy* if compromise of long-term private keys does not reveal session keys previously agreed using those long-term keys

- Remember that the attacker can see (and record) protocol messages
- Some use the term *perfect forward secrecy* while others simply say *forward secrecy* for the same thing

Post-compromise security (PCS)

- Sometimes we can recover after a long-term key is compromised – also known as *self-healing* protocols
- Only works when attacker is passive
- The long-term key must evolve over time so that the attacker becomes locked out with long-term key updates
- A way to do this is to send a new Diffie–Hellman share with every message and change the session key also after every message
- Used today in secure messaging in protocols such as Signal

Forward secrecy examples

- The protocol on slide 15 does **not** provide forward secrecy
- The protocol on slide 18 **does** provide forward secrecy, because the long-term (signing) keys are only used for authentication
- Neither of these two protocols provides post-compromise security

Key establishment using public keys

Can be achieved in several ways.

- Public announcement – Alice can broadcast her public key at large
  - Anyone can forge this – by the time Alice realises, forger has done some damage
- Publicly available directory
  - More secure, but still has vulnerabilities. Single point of failure
- Public-key authority
  - Central autority maintains a dynamic directory. Still some drawbacks: user must appeal to the authority for every other user it wants to contact, so it can be a bottleneck
- Public-key certificates
  - What is used in practice. A certificate is: a public key, an identifier of the key owner, and all this is signed by a trusted third party

Digital certificates

- When using a public key to encrypt a message or to verify a digital signature, it is essential to be confident of the correct binding between a public key and its owner
- Normally this is achieved through use of *digital certificates* which contain the public key and owner identity, and usually other information such as signature algorithm and validity period
- The certificate is digitally signed by a party trusted by the certificate verifier, normally called a *certification authority* or CA
- Certificates play a central role in key management for public key infrastructures

Public key infrastructure (PKI)

- In the NIST Special Publication 800-57 a public key infrastructure is defined as:
  “A framework that is established to issue, maintain, and revoke public-key certificates”
- A number of different legal or business entities may be involved including:
  - registration authorities which manage identities
  - naming authorities which manage domain naming
  - certfication authorities (CAs)
- We focus on technical issues in CAs

Certification Authority (CA)

- Creates, issues, and revokes certificates for users and other CAs
- Have a Certification Practice Statement (CPS) covering issues such as:
  - checks performed before certificate issue
  - physical, personnel and procedural security controls for the CA
  - technical and key pair protection and management controls
  - certificate revocation management procedures
  - audit procedures for the CA
  - accreditation information
  - legal and privacy issues and liability limitations

X.509 standard

- Widely used standard allowing flexible *extensions*
- Originally an ITU standard, now available as RFC 5280
- Important fields in X.509 digital certificates are:
  - Version number
  - Serial number (set by the CA)
  - Signature algorithm identifier (Algorithm used for dig sigs)
  - Issuer (Name of the CA)
  - Subject (Name of entity to which certificate is issued)
  - Public key information
  - Validity period
  - Digital signature (of the certificate, signed by the CA)

Using a certificate

- Certificates are verified by checking that the CA signature is valid and that any conditions set in the certificate are correct
- In order to verify a certificate, the user of the certificate must have the correct public key of the CA (why?)
- Users may obtain certificates in different ways, including:
  - sent by owners during a protocol run
  - distributed with web browsers
  - in public directories
  - as part of a DNS record

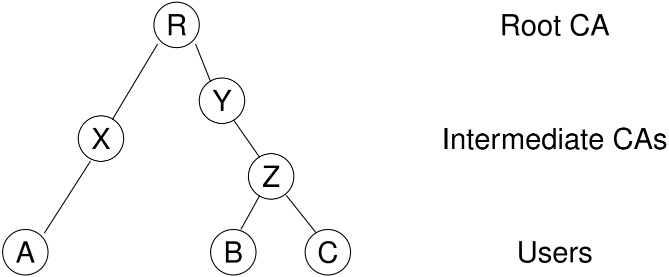
Certification paths

- If the public key of a CA, say $\mathtt{CA}_0$, is not already known and trusted, it can itself be certified by a different CA, say $\mathtt{CA}_1$
- In turn it is possible that the public key of $\mathtt{CA}_1$ is certified by $\mathtt{CA}_2$
- In this way a chain of trust can be set up, which is known as a *certification path*

$$\mathtt{CA}_n \rightarrow \mathtt{CA}_{n-1} \rightarrow \ldots \mathtt{CA}_1 \rightarrow \mathtt{CA}_0$$

- If an entity has a trusted copy of the public key of $\mathtt{CA}_n$, the chain of trust can be used with certificates for all the *intermediate CAs* to obtain a trusted copy of the public key of $\mathtt{CA}_0$

Hierarchical PKI



- CAs certify the public key of the entity below
- In a non-hierarchical PKI, certification can be done between any CAs: X can certify Y's public key or Z can certify Y's public key, for example

Browser PKI

- Multiple hierarchies with preloaded public keys as root CAs
- Other CAs and intermediate CAs can be added
- Your own certificates can also be added - you will be doing this in the lab
- Web servers send their public key and certificate to the browser at the start of a secure communication using the TLS protocol
- Root certificates are *self-signed* — the CA for the root CA is the root CA itself!

Revocation

- Sometimes it may be required to declare a certificate invalid even though its validity period is current
- In order to make this work the user must check to see which certificates have been revoked
- Two widely deployed mechanisms:
  1. Certificate revocation lists (CRL) – each CA periodically issues a list of revoked certificates which can be downloaded and then checked by clients
  2. Online certificate status protocol (OCSP) – a server maintains a current list of revoked certificates and responds to requests about specific certificates

Key distribution using symmetric key encryption

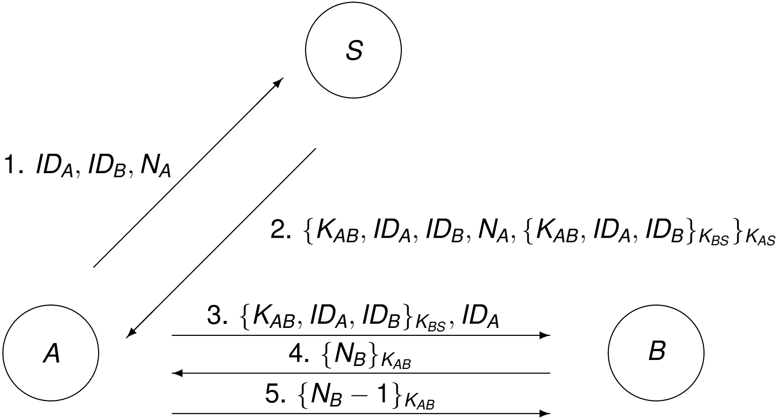
Options are as follows.

- Alice picks a key, physically delivers it to Bob
  - Need to meet physically
- A third party selects a key and physically delivers it to Bob
  - Need to meet physically
- Alice and Bob previously have shared a key – use that to update the key
  - But if an attacker gains access to the key at any point, it can decrypt all subsequent messages!
- Alice and Bob have an encrypted connection to (trusted third party) Charlie – Charlie can deliver new key to Alice and Bob

Needham–Schroeder protocol: notation

- Parties/Principals: A, B, S
  - A and B are two parties who want to establish a session key S is the trusted authority
- Shared Secret Keys: K_{AS}, K_{BS}, K_{AB}
  - A and S share long-term key K_{AS}
  - B and S share long-term key K_{BS}
  - K_{AB} is the new session key generated by S
- Nonces: N_A, N_B
  - Randomly-generated for one-time use (*n*-once)
- $S \to A : M$
  - S sends message M to A
- $\{X\}_K$
  - Authenticated encryption of message X using the shared secret key K

Needham–Schroeder protocol


S
1. ID_A, ID_B, N_A
2. $\{K_{AB}, ID_A, ID_B, N_A, \{K_{AB}, ID_A, ID_B\}_{K_{BS}}\}_{K_{AS}}$
A
3. $\{K_{AB}, ID_A, ID_B\}_{K_{BS}}, ID_A$
4. $\{N_B\}_{K_{AB}}$
5. $\{N_B - 1\}_{K_{AB}}$
B

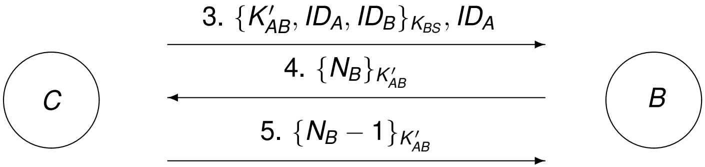
Needham–Schroeder protocol

One of the most widely known key establishment protocols

- Published by Needham and Schroeder in 1978
- The basis for many related protocols, including the Kerberos protocol we look at below
- The protocol is vulnerable to a *replay attack* found by Denning and Sacco in 1981
- An attacker is able to replay old protocol messages and the honest party accepts an old session key

Replay attack on Needham–Schroeder protocol

- Assume an attacker C obtains a session key K'_{AB} previously established between A and B
- In the attack, C masquerades as A and is thus able to persuade B to use the old key K'_{AB}

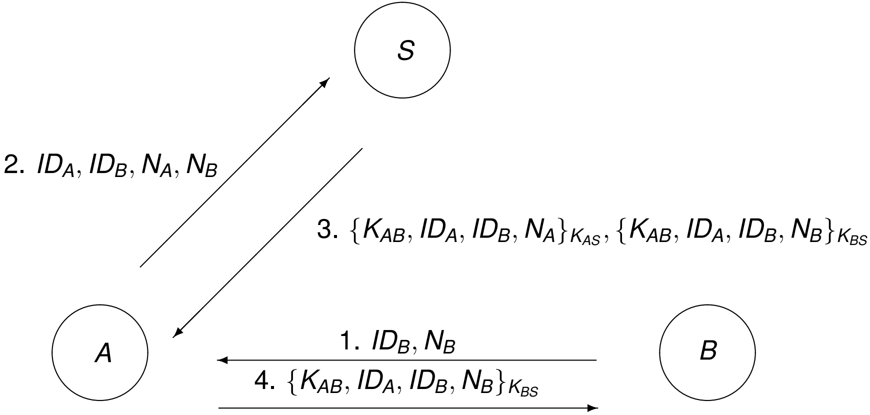


3. $\{K'_{AB}, ID_A, ID_B\}_{K_{BS}}, ID_A$

4. $\{N_B\}_{K'_{AB}}$

5. $\{N_B - 1\}_{K'_{AB}}$

Repaired Needham–Schroeder protocol


S
2. ID_A, ID_B, N_A, N_B
3. $\{K_{AB}, ID_A, ID_B, N_A\}_{K_{AS}}, \{K_{AB}, ID_A, ID_B, N_B\}_{K_{BS}}$
A
1. ID_B, N_B
4. $\{K_{AB}, ID_A, ID_B, N_B\}_{K_{BS}}$
B

Freshness

- To defend against replay attacks, it is critical that the key established be *fresh* (new) for each session
- There are three basic mechanisms that can be used to achieve freshness:
    1. random challenges (nonces)
    2. timestamps (a string on the current time)
    3. counters (increased for each new message)
- The repaired Needham–Schroeder protocol above uses random challenges to provide freshness
- Timestamps or counters can also be used

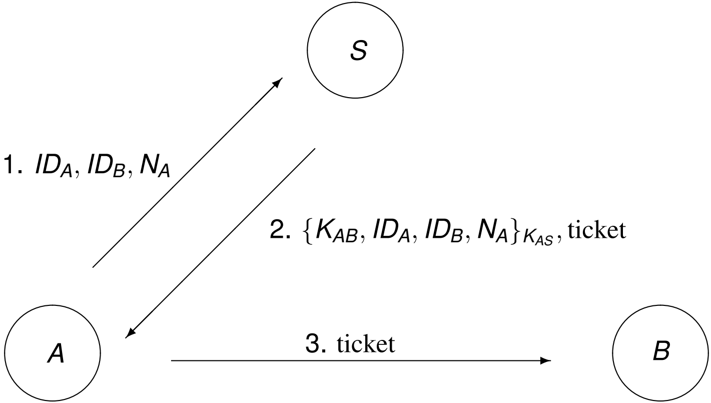
Tickets

- An alternative way to fix the Needham–Schroeder protocol uses a key with a validity period
- Suppose entity *A* wishes to obtain access to the server *B*
- The authentication server *S* issues a *ticket* to allow *A* to obtain access
- A ticket has the format

  $$\{K_{AB}, ID_A, ID_B, T_B\}_{K_{BS}}$$

  where T_B is a timestamp which we can also interpret as a validity period
- *A* can obtain the ticket and use it to gain access to *B* at any time while T_B is still valid

Repaired Needham–Schroeder using tickets



Kerberos

- Originally developed in early 1980s, latest version Kerberos V5 released in 1993 described in RFC 4120 (2005)
- Since Windows 2000, Kerberos V5 is the default Windows domain authentication method
- A single sign-on (SSO) solution: users only need to enter usernames and passwords once for a session
- Provide access selectively for a number of different online services using individual tickets
- Establish session key to deliver confidentiality and integrity services for each service access

Kerberos three level protocol

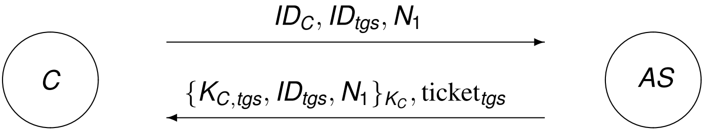
Level 1 Client *C* interacts with authentication server *AS* in order to obtain a ticket-granting ticket – happens once for a session (maybe a working day)

Level 2 Client *C* interacts with ticket-granting server *TGS* in order to obtain a service ticket – happens once for each server during the session

Level 3 Client *C* interacts with application server *V* in order to obtain a service – happens each time the client requires service during the session

- The protocol descriptions following have some simplifications

Level 1: Interaction with authentication server



C

AS

ID_C, ID_{tgs}, N_1

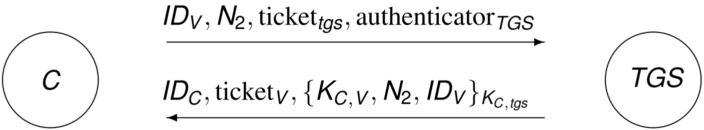
$\{K_{C,tgs}, ID_{tgs}, N_1\}_{K_C}, \mathrm{ticket}_{tgs}$

- $\mathrm{ticket}_{tgs} = \{K_{C,tgs}, ID_C, T_1\}_{K_{tgs}}$ for some validity period T_1
- Result: user has ticket-granting ticket which can be used to obtain different service-granting tickets

Level 1: notes

- K_C is the symmetric key shared with the authentication server *AS*, typically it is generated by the workstation of *C* from a password entered by *C* at login time
- $K_{C,tgs}$ is a new symmetric key generated by *AS* to share with the ticket granting server *TGS*
- N_1 is a nonce used by *C* to check that the key $K_{C,tgs}$ is fresh
- K_{tgs} is a long-term key shared between *AS* and *TGS*

Level 2: Interaction with ticket-granting server



$ID_V, N_2, \text{ticket}_{tgs}, \text{authenticator}_{TGS}$

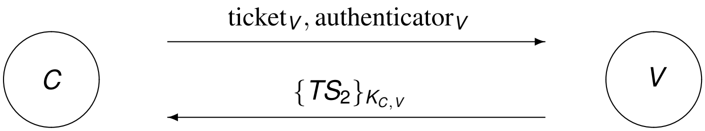
$ID_C, \text{ticket}_V, \{K_{C,V}, N_2, ID_V\}_{K_{C,tgs}}$

- $\text{ticket}_V = \{K_{C,V}, ID_C, T_2\}_{K_V}$ for some validity period T_2
- $\text{authenticator}_{TGS} = \{ID_C, TS_1\}_{K_{C,tgs}}$ for some timestamp TS_1
- Result: user has service-granting ticket which can be used to obtain access to a specific server

Level 2: notes

- The ticket_{tgs} is the same as sent in the level 1 interaction
- $K_{C,V}$ is the session key to be used with server V
- N_2 is a nonce used by C to check that the key $K_{C,V}$ is fresh.
- TGS first obtains $K_{C,tgs}$ from ticket_{tgs} and then checks the fields in the authenticator are valid – includes checking that TS_1 is recent and that C is authorized to access service V
- In practice both AS and TGS may be the same machine

Level 3: Interaction with application server



- $\text{authenticator}_V = \{ID_C, TS_2\}_{K_{C,V}}$ for some timestamp TS_2
- Result: user has secure access to a specific server V

Level 3: notes

- The ticket_V is the same as sent in the level 2 interaction
- $K_{C,V}$ is contained in ticket_V and was also sent to C in the level 2 interaction
- The reply from V is intended to provide mutual authentication so that C can check it is using the right service V

Kerberos limitations

- Limited scalability: even though different realms are supported, each realm needs to share a key with each other realm
- Suited for corporate environments with shared trust (although public key variants exist)
- Offline password guessing is a possible attack when the client key K_C is derived from a human memorable password
- Kerberos standard does not specify how to use the session key once it is established