Lecture 16: Key Establishment

COSC362 Data and Network Security

Book 1: Chapters 14 and 15 – Book 2: Chapters 21 and 23

Spring Semester, 2021

Motivation

- Distribution of cryptographic keys to protect subsequent communications sessions.
- Key establishment in TLS uses public keys to allow clients and servers to share a new communication key.
- Kerberos is a widely used system for secure communications which achieves key establishment without using public keys.

Outline

Outline

Key Management

- Critical aspect of any cryptographic system.
- Phases:
  - Key generation: keys should be generated s.t. they are equally likely to occur.
  - Key distribution: keys should be distributed in a secure fashion.
  - Key protection: keys should be accessible for use in relevant cryptographic algorithms, but not accessible to unauthorised parties.
  - Key destruction: once a key has performed its function, it should be destroyed s.t. it is of no value to an attacker.

Key Types

Keys are often organized in a hierarchy.
A simple 2-level hierarchy is common:

- Long-term keys:
  - Also called *static keys*.
  - Intended to be used for a long time.
  - Depending upon the application, from few hours to few years.
  - Used to protect distribution of session keys.
- Short-term keys:
  - Also called *session keys*.
  - Intended to be used over a short period.
  - Depending upon the application, from few seconds to few hours.
  - Used to protect communications in a session (e.g. with authenticated encryption).

Key Establishment

- In practice, session keys are symmetric keys used with ciphers (e.g. AES, MAC):
  - Due to their greater efficiency over public key algorithms.
- Long-term keys can be either symmetric or asymmetric keys, depending on how they are used.
- How to *establish* secret session keys among communicating parties using the long-term keys.
- Common approaches:
  - Key pre-distribution.
  - Using an online server with symmetric long-term keys.
  - Using asymmetric long-term keys.

Key Distribution Security Goals

2 properties:

- Authentication: if Alice completes the protocol and believes that the key is shared with Bob, then it should not be the case that the key is actually shared with another party Carol.
- Confidentiality: the adversary is unable to obtain the session key accepted by a particular party.

In formal models, the protocol is seen as *broken* if the adversary can distinguish the session key from a random string.

Mutual and Unilateral Authentication

- If both parties achieve the authentication goal, then the protocol provides *mutual authentication*.
- If only one party achieves it, then the protocol provides *unilateral authentication*.
- Many real-world key establishment protocols achieve only unilateral authentication:
  - Typically, clients can authenticate servers.
  - Client authentication often happens later, protected with the established key.

Adversary Capabilities

Let a strong adversary know the details of the cryptographic algorithms involved and be able to:

- *Eavesdrop* on all messages sent in a protocol.
- *Alter* all messages sent in a protocol using any information available to him/her.
- *Re-route* any messages (including new ones) to any other party.
- *Obtain* the value of the session key used in any previous run of the protocol.

Distribution of Pre-Shared Keys

- A trusted authority (TA) generates and distributes long-term keys to all users when they join the system.
- Simple schemes:
  - Assigning a secret key for each pair of users.
  - The number of keys thus grows quadratically.
- The TA only operates in the pre-distribution phase:
  - It does not need to be online afterwards.
- Poor scalability.
- Probabilistic schemes:
  - Reducing key material at each party.
  - But only guaranteeing a secure channel between any 2 users with some (high) probability.
  - Suitable for sensor networks.

Key Distribution using Symmetric Keys

- Key distribution with an online server.
- The TA shares a long-term shared key with each user.
- An online TA generates and distributes session keys to users when requested:
  - In a secure fashion using the long-term keys.
- 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 Distribution using Asymmetric Cryptography

- No online TA is required.
- Public keys used for authentication.
- Public keys managed by PKI (certificates and CAs).
- Users are trusted to generate good session keys:
  - A good pseudo-random number generator required at each party.
- Types:
  - *key transport*
  - *key agreement*

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:
  - This is the case with key transport!
  - This can be prevented with key agreement.

A protocol provides *(perfect) forward secrecy* if compromise of long-term secret keys does NOT reveal session keys previously agreed using those long-term keys.

Key Transport

- User chooses key material and sends it encrypted to another party:
  - Sometimes, the message is also signed by the sender.
- TLS includes options for key transport.
- Not providing *forward secrecy*.

Key Agreement

- 2 parties each provide input to the key material.
- Providing authentication with public keys:
  - By signing the exchanged messages.
- Example: Diffie-Hellman protocol (widely used).
- TLS includes options for key agreement.
- Providing *forward secrecy*.

Outline

Notation

- Alice and Bob want to share a secret key.
- Computations done in $\mathbb{Z}_p^*$ with a large prime p:
  - Generator g.
  - Random values a, b chosen by Alice and Bob, where $1 \leq a, b \leq p - 1$.
  - $Sign_A(m)$ is a signature on message m from Alice.
  - $Sign_B(m)$ is a signature on message m from Bob.
- Both Alice and Bob know each other's public verification key.
- Forward secrecy since long-term signing keys are only used for authentication.

Protocol

- Alice checks the signature in flow 2:
  - If invalid then Alice aborts.
  - Otherwise, Alice computes the session key as

$$K_{AB} = (g^b)^a = g^{ab}$$

- Bob checks the signature in flow 3:
  - If invalid then Bob aborts.
  - Otherwise, Bob computes the session key as

$$K_{AB} = (g^a)^b = g^{ab}$$

Outline

- Key Establishment
  - Key Pre-Distribution
  - Session Key Distribution using Symmetric Keys
  - Session Key Distribution using Asymmetric Keys
- Example of Session Key Distribution using Asymmetric Keys
  - Signed Diffie-Hellman
- Examples of Session Key Distribution using Symmetric Keys
  - Needham-Schroeder Protocol
  - Kerberos

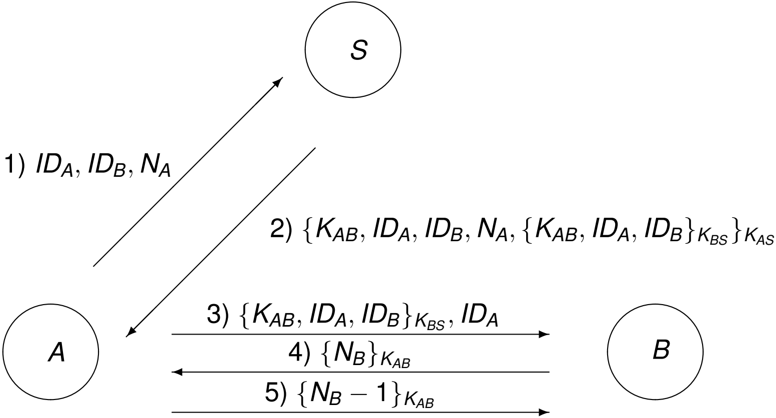
Description

- Published by Needham and Schroeder in 1978.
- A widely known key establishment protocol.
- Basis for many related protocols:
  - Example: Kerberos
- Vulnerable to *replay attacks* found by Denning and Sacco in 1981:
  - An attacker can replay old protocol messages s.t. an honest party will accept an old session key.

Notations

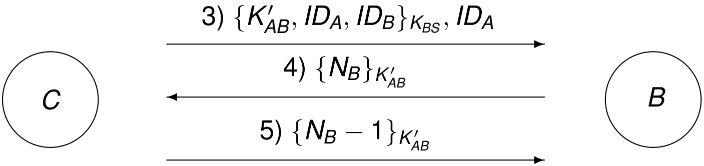
- Parties:
  - 2 parties A and B want to establish a shared secret key.
  - S is the TA.
- Shared secret keys:
  - A and S share the long-term key K_{AS}.
  - B and S share the long-term key K_{BS}.
  - New session key K_{AB} generated by S.
- Nonces:
  - N_A, N_B are randomly generated for one-time use.
- $S \rightarrow A : M$ means that S sends a message M to A.
- $\{M\}_K$ denotes the authenticated encryption of message M using the key K.

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

Replay Attacks

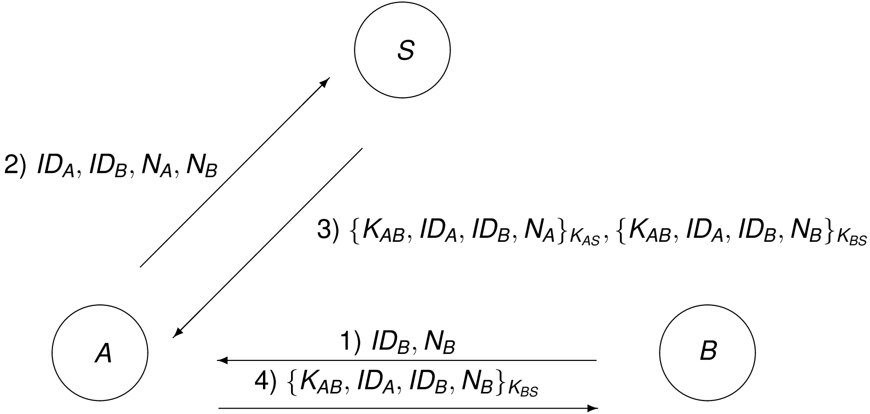
- Let an attacker C get a session key K'_{AB} previously established between A and B.
- C masquerades as A, and persuades B to use the old key K'_{AB}.



Freshness

- To defend against replay attacks, established key must be *fresh* (new) for each session.
- Mechanisms:
  - Random challenges (nonces).
  - Timestamps (string on the current time).
  - Counters (increased for each new message).
- Repaired protocol uses random challenges:
  - It can be adapted to use timestamps and counters.

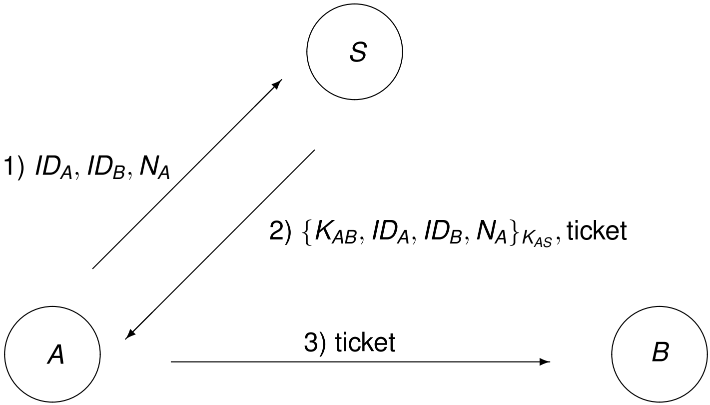
Repaired Protocol using Random Challenges



Tickets

- Another way to fix Needham-Schroeder protocol.
- The client A wishes to get access to the server B:
  - The authentication server S issues a *ticket* to allow A to obtain access.
- Ticket is $\{K_{AB}, ID_A, ID_B, T_B\}_{K_{BS}}$ where T_B is a timestamp (e.g. validity period).
- A gets ticket and uses it at any time while T_B is valid.

Repaired Protocol using Tickets



where ticket $= \{K_{AB}, ID_A, ID_B, T_B\}_{K_{BS}}$ for some validity period T_B

Description

- Developed at MIT as part of Project Athena.
- Several versions since its beginning in early 80s.
- Latest version (V5) released in 1995.
- Standard: RFC 4120 (2005).
- Default Windows domain authentication method from Windows 2000.

Goals

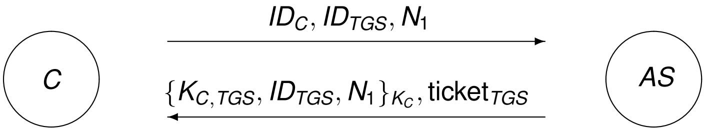
- Secure network authentication service in an insecure network environment.
- Single sign-on (SSO) solution:
  - Users only need to enter usernames and passwords once for a session.
- Providing access selectively for a number of different online services, using individual tickets.
- Establishing session keys to deliver confidentiality and integrity services for each service access.

3-Level Protocol

- Level 1: client C interacts with authentication server AS in order to obtain a ticket-granting ticket:
  - Happening once for a session (e.g. one day long).
  - C only authenticates once at the start of the session.
- Level 2: C interacts with ticket-granting server TGS in order to obtain a service-granting ticket:
  - Happening once for each server during the session.
- Level 3: C interacts with application server V in order to obtain a service:
  - Happening once for each time C requires service during the session.

Level 1

Interaction with the authentication server *AS*



where $\text{ticket}_{TGS} = \{K_{C,TGS}, ID_C, T_1\}_{K_{TGS}}$ for some validity period T_1

Result: C has a ticket-granting ticket that can be used to obtain different service-granting tickets.

Notes for Level 1

- K_C:
  - Symmetric key shared between AS and C.
  - Typically generated by the workstation of C from a password entered by C at logon time.
- $K_{C,TGS}$:
  - New symmetric key generated by AS and shared between TGS and C.
- N_1:
  - Nonce used by C to check that key $K_{C,TGS}$ is fresh.
- K_{TGS}:
  - Long-term key shared between AS and TGS.

Level 2

Interaction with the ticket-granting server *TGS*

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

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

where $\text{ticket}_V = \{K_{C,V}, ID_C, T_2\}_{K_V}$ for some validity period T_2 and $\text{authenticator}_{TGS} = \{ID_C, TS_1\}_{K_{C,TGS}}$ for some timestamp TS_1

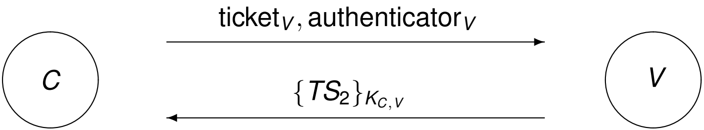
Result: *C* has a service-granting ticket that can be used to obtain access to a specific server.

Notes for Level 2

- ticket_{TGS} is the same than the one sent in Level 1.
- $K_{C,V}$:
  - Session key shared between V and C.
- N_2:
  - Nonce used by C to check that key $K_{C,V}$ is fresh.
- TGS first gets $K_{C,TGS}$ from ticket_{TGS}, and then checks the fields in $\text{authenticator}_{TGS}$ are valid:
  - Checking that TS_1 is recent.
  - Checking that C is authorized to access V.
- In practice, both AS and TGS are the same machine.

Level 3

Interaction with the application server V



where $\text{authenticator}_V = \{ID_C, TS_2\}_{K_{C,V}}$ for some timestamp TS_2

Result: C has secure access to a specific server V.

Notes for Level 3

- ticket_V is the same than the one sent in Level 2.
- $K_{C,V}$, contained in ticket_V, is the same than the one sent in Level 2.
- Reply from V intended to provide mutual authentication:
  - C can check that it is using the right application server V.

Miscellaneous

- Above descriptions are simplified.
- Timestamp:
  - includes start and end times.
  - can be suggested by C in the last version of Kerberos (v5).
- Realm: a domain over which an authentication server has the authority to authenticate a user.
- Flag: used in tickets to indicate when and how tickets should be used.
- Sequence number: optional, initiated during the client-server exchange.
- Subkey: derived from the key $K_{C,V}$.

Limitations

- Limited scalability:
  - Even though different realms are supported, one realm needs to share a key with each other realm.
  - Kerberos best suited for corporate environments with shared trust.
  - Public-key variants exist.
- Attack:
  - Offline password guessing.
  - When the key K_C derived from a human memorable password.
- Standard:
  - Does not specify how to use the session key once it is established.