CS5321 Network Security
Week3: Authentication and Secure Communication

Daisuke MASHIMA

http://www.mashima.us/daisuke/index.html

2022/23 Sem 2

Authentication and Secure Communication

- **Key Establishment / Distribution**
- Authentication using Symmetric Key
- Authentication using PKI
- Authentication using KDC
- TLS / IPSec

Session Keys vs Permanent Keys

- If Alice and Bob share a **symmetric secret key k**, they can use that key to run secure communications (confidentiality and authenticity).
- Typically, for a session of communication involving many rounds of messages, a randomly chosen **session key** is first established using the **permanent key** during the **"hand-shaking"** phase. Subsequent rounds of communications are then protected by this session key.
- Why not using the permanent key directly?
  - A randomly selected session keys help prevent replay attack.
  - Even if the session key is compromised, the permanent key is still secure.

Session Key Establishment

- In scenarios where Alice and Bob do not share a pre-determined (symmetric) permanent key, we need a secure mechanism for them to establish a session key.

- We have already seen how **PKI** can securely distributes public keys. In this lecture we study how to establish the session key using the public-private keys.

- Without PKI, we study how a **KDC (key distribution center),** who is trusted by Alice and Bob, helps mediate the session key establishment, via the **Needham-Schroeder protocol**.

Secure Communication

Establish session keys securely, based on shared symmetric key, PKI, or KDC.

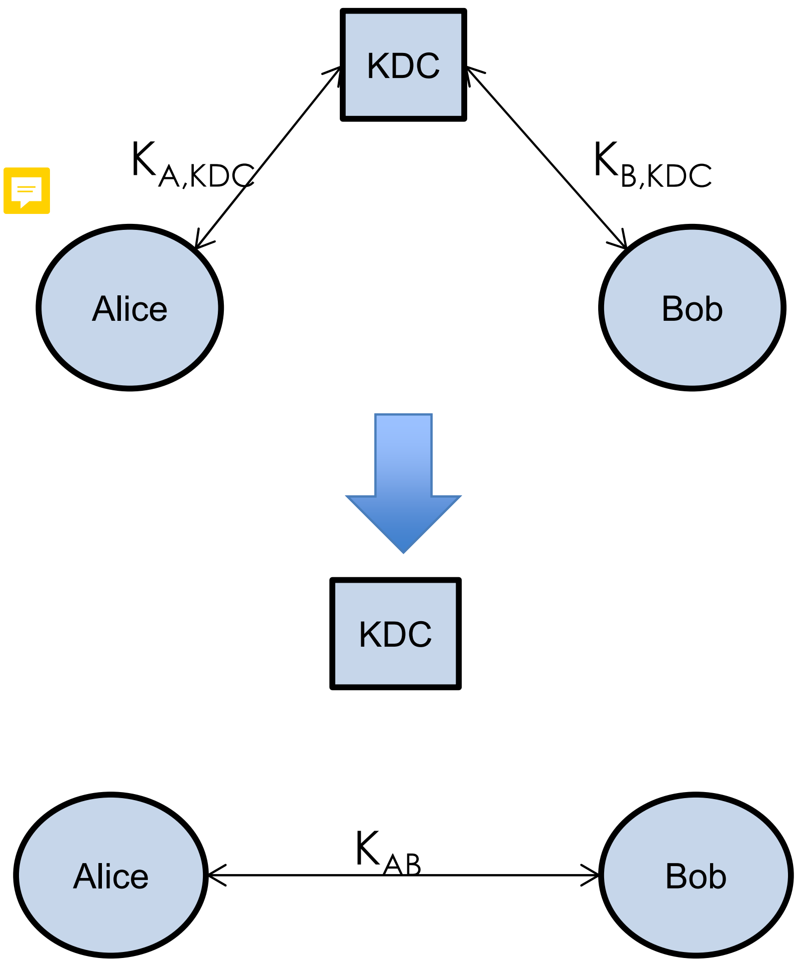
Handshaking.
(also called authentication protocol)

communication protected by session keys

Every message is encrypted for confidentiality. Integrity and authenticity are protected by MAC.

Key Distribution Center (KDC)

- A KDC facilitates ***mediated authentication***.
  - The KDC stores the secret keys it previously established securely with each user.
  - If there are n users, then there are n keys.
- When Alice (Bob) wants to establish a secure channel to Bob (Alice), (s)he first establishes a secure connection with KDC
- KDC next sends **a randomly generated session key K_{AB}** to Alice (Bob).
- Alice and Bob next use K_{AB} to secure communication.
- From Alice's point of view, since she trusts KDC, the user who has K_{AB} must be Bob, and vice-versa.



KDC vs PKI

- KDC plays similar roles as the CA in PKI in managing keys.
- A KDC **distributes** keys to two users who have not met before, **revokes** users, **verifies** the users during registration, and etc.

- KDC: symmetric keys. PKI: public keys.

- A KDC knows all the secret keys it shares with the users.
- In PKI, a CA does not know the private key of any user, even if it has signed the user's certificate.

- KDC must be online at least during key distribution phase.
- CA can be offline after certificates are issued.

Authentication and Secure Communication

- Key Establishment / Distribution
- **Authentication using Symmetric Key**
- Authentication using PKI
- Authentication using KDC
- TLS / IPSec

Unilateral Authentication (timestamp)

Both A and B share a secret key *k.* A initiates the communication and wants to prove to B that she is authentic. Let *m* be a short message A wants to pass to B, for e.g *m* can be a **session key** randomly selected by A.

(1) A → B : "I'm Alice" ‖ E (*k*, *timestamp* ‖ *m*)

The *timestamp* is the value of current time. B accepts if the current time match or within a window.

Main idea: B received a message which is correctly decrypted with the current time. So the entity who sends this message must have the secret key and thus must be Alice.

Weakness: Both A,B's time has to be synchronized. Furthermore, the replay attack can still be successful if it is carried out within the window.

also can try to race

Unilateral Authentication (nonce)

([KPS] protocol 11-2 in chapter 11)

Both A and B share a secret key *k*. A initiates the communication and wants to prove to B that she is authentic. Let *m* be a short message A wants to pass to B, for e.g *m* can be a session key randomly selected by A.

(1) $A \rightarrow B$: "I'm Alice"||

(2) $A \leftarrow B$: r_B

(3) $A \rightarrow B$: $E(k, m \,||\, r_B)$

B accepts if r_B is correctly decrypted.

Main idea: B sends a random number to the other party. If the random number is encrypted correctly, then the other party must have the secret key and thus must be Alice.

Mutual Authentication (nonce)

([KPS] protocol 11-7)

Both A and B share a secret key *k*. A initiates the communication. Let *m* be a short message A wants to pass to B, for e.g *m* can be a session key randomly selected by A. A and B want to authenticate each other.

(1) $A \rightarrow B$: "I'm Alice" $\| \; r_A$ // r_A is a nonce chosen by A

(2) $A \leftarrow B$: $E(k, \; r_B \; \| \; r_A)$ // r_B is a nonce chosen by B

(3) $A \rightarrow B$: $E(k, m \; \| \; r_A \; \| \; r_B)$

Authentication and Secure Communication

- Key Establishment / Distribution
- Authentication using Symmetric Key
- **Authentication using PKI**
- Authentication using KDC
- TLS / IPSec

- X.509 includes strong authentication protocols using **signature**. Main idea is the same as symmetric key authentication.

- In the next few slides on unilateral authentication, we assume that Alice initiates the communication, and Alice wants to prove that she is authentic.

- Alice also wants to send the message m over. Here, we are only concerned with **authenticity** of m. (i.e don't care about confidentiality, hence m can't be a session key). Let "ID_A", "ID_B" be the name of A, B respectively.

Unilateral authentication with timestamps

$$(1)\ A \rightarrow B:\ cert_A \,||\, t_A \,||\, \text{“}ID_B\text{”} \,||\, m \,||\, sig_A(\, t_A \,||\, \text{“}ID_B\text{”} \,||\, m\,)$$

From $cert_A$, B can extract & verify A's public key. t_A is timestamp.

Notation: sig_A ( ) is the signature of A.

Established:

1. The identity of A and that the message is generated by A.
2. The message is intended for B
3. Freshness
4. Non-repudiation.

Note that in the case for symmetric key, it is not necessary to include "ID_B". But in public key setting, it is necessary.

This is to prevent replays of messages previously sent to another party.

Authentication with nonce:

Unilateral:

(1) $A \rightarrow B$: "ID_A wants to connect"

(2) $A \leftarrow B$: r_B

(3) $A \rightarrow B$: $cert_A \,||\, r_A \,||\,$ "ID_B" $||\, m \,||\, sig_A(r_A \,||\, r_B \,||\, m \,||$ "ID_B" $)$

Mutual (Bi-directional):

(1) $A \rightarrow B$: "ID_A wants to connect" $||\, r_A$

(2) $A \leftarrow B$: $cert_B \,||\, r_B \,||\,$ "ID_A" $||\, sig_B(r_A \,||\, r_B \,||$ "ID_A" $)$

(3) $A \rightarrow B$: $cert_A \,||\,$ "ID_B" $||\, m \,||\, sig_A(r_A \,||\, r_B \,||\, m \,||$ "ID_B" $)$

Adding Confidentiality

- Note that the 3 previous authentication protocols (PKC) do not preserve confidentiality of the message m. Thus m cannot be served as session key.

- To securely establish the session key under mutual authentication, one can replace the "m" in step (3) by **the session key K encrypted with B's public key**. i.e.

(3) $A \rightarrow B$: $\quad cert_A \,||\,$ "ID_B" $||\, \mathbf{E_B(K)} \,||\, sig_A\,(r_A \,||\, r_B \,||\, K \,||\,$ "ID_B")

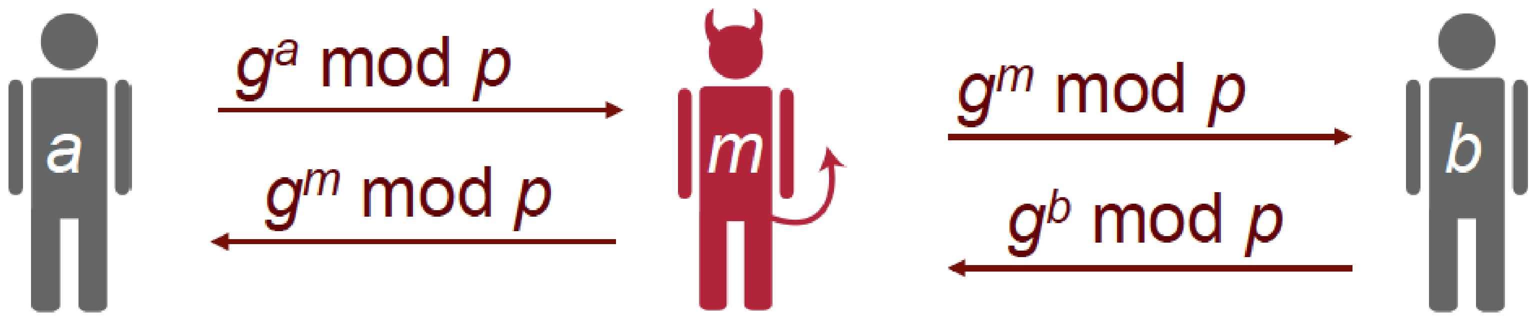
Hash values of these are signed,. Thus signature verification does not reveal K

DH Key Agreement

- **Diffie-Hellman (DH) <u>key agreement</u>**
  - Public values: large prime p, generator g
  - Alice has **secret value a**, Bob has **secret b**
  - A → B: **$g^a \pmod{p}$**
  - B → A: **$g^b \pmod{p}$**
  - Bob computes $(g^a)^b = g^{ab} \pmod{p}$
  - Alice computes $(g^b)^a = g^{ab} \pmod{p}$

Problem: Man-in-the-Middle Attack

- Public values: large prime p, generator g
- Problem: in Man-in-the-Middle attack, Mallory impersonates Alice to Bob and Bob to Alice



$$g^{am} \bmod p \quad \neq \quad g^{bm} \bmod p$$

Station-to-Station (STS) Protocol

a,
Alice's private key
Alice's certificate

A

$T_1 = g^a \bmod p, \quad T_2 = g^b \bmod p$

$E(k, \ sig_B \ (T_2, T_1)) \ || \ cert_B,$
$E(k, \ sig_A \ (T_1, T_2)) \ || \ cert_A$

B

b,
Bob's private key
Bob's certificate

Paper: https://link.springer.com/content/pdf/10.1007/BF00124891.pdf

Perfect Forward Secrecy

- **Perfect forward secrecy** refers to the requirement that, the loss of the permanent (long-term) key does not reveal the session key, and thus does not compromise confidentiality of the messages.

- Consider this scenario:
  - An eavesdropper sniffed and logged all messages exchanged between A and B, including the authentication/handshaking messages. The eavesdropper was unable to decrypt those messages.
  - Later, the eavesdropper somehow obtained both A's and B's private keys. Can the eavesdropper now decrypt the logged messages?

- STS protocol offers perfect forward secrecy?
  - When A and B use the station to station protocol, then the eavesdropper is still unable to recover k and thus the confidentiality of the messages is protected.

Authentication and Secure Communication

- Key Establishment / Distribution
- Authentication using Symmetric Key
- Authentication using PKI
- **Authentication using KDC**
- TLS / IPSec

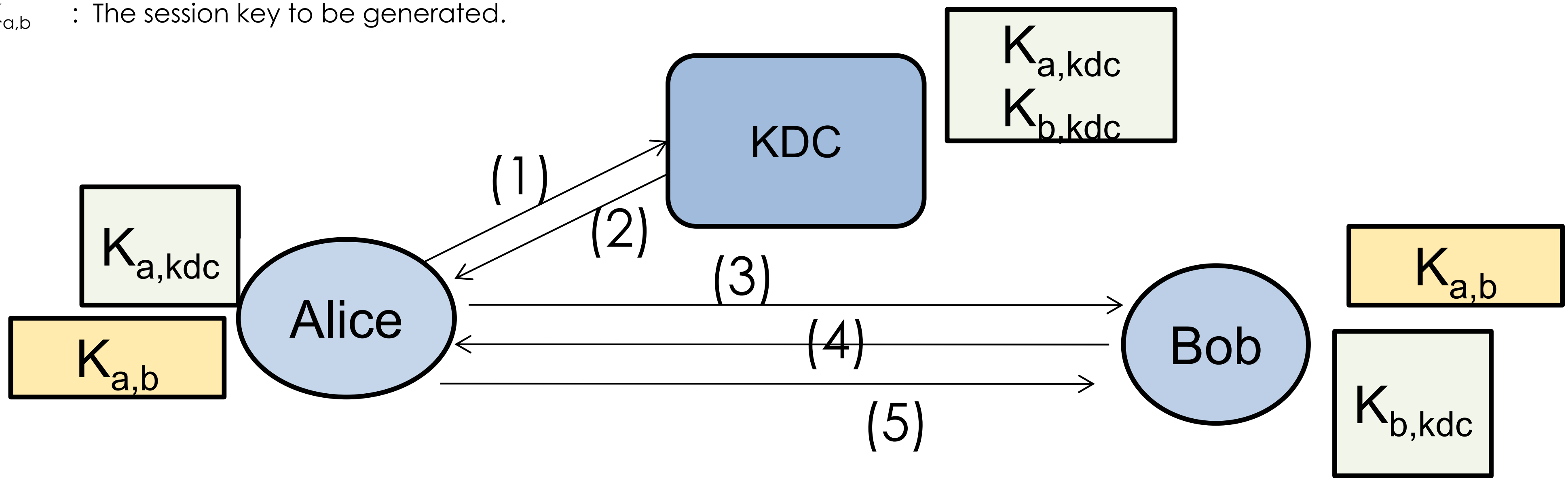
Mediated Authentication with KDC

- The **Needham-Schroeder Authentication** protocol is designed for Key Distribution Center (KDC) to "mediate" a session key between two users.
- Variants of Needham-Schroeder authentication protocol are adopted in various applications (e.g., **Kerberos**).
  - Used in Windows Service Active Directory

- Key Distribution Center (KDC) keeps a shared secret with each user.
- KDC is a trusted **"big-brother"**. It knows all the secrets and is always online.

Needham-Schroeder protocol ([KPS]chap 11.4.1)

$K_{a,kdc}$: A long-term key (master key) shared by Alice and KDC.
$K_{b,kdc}$: A long-term key (master key) shared by Bob and KDC.
$K_{a,b}$: The session key to be generated.



(1) Alice → KDC : "ID_A wants to talk to ID_B" || N_1

(2) KDC → Alice: E($K_{a,kdc}$, N_1 || "ID_B" || $K_{a,b}$ || ticket)

where ticket = E($K_{b,kdc}$, $K_{a,b}$ || "ID_A")

(3) Alice → Bob : ticket || E($K_{a,b}$, N_2)

(4) Bob → Alice: E($K_{a,b}$, N_2 - 1, N_3)

(5) Alice → Bob: E($K_{a,b}$, N_3 - 1)

Step (3),(4) & (5) are for **mutual authentication** of Alice and Bob.

Remarks

- The encryption scheme E( ) has to meet certain integrity requirements.
- If we use AES in ECB mode, and it happens that N2 and N3 are in two separate block, the following attack is possible!
  - Reflection attack

(3) Attacker →Bob : ticket || E($K_{a,b}$, N_2)
(4) Bob →Attacker: E($K_{a,b}$, N_2 - 1, N_3)

[Attacker opens a new session with Bob]
(3) Attacker →Bob : ticket || E($K_{a,b}$, N_3)
(4) Bob →Attacker: E($K_{a,b}$, N_3 - 1, N_4)

(5) Attacker →Bob: E($K_{a,b}$, N_3 - 1)

Kerberos

- **Kerberos authentication protocol** is based on Needham-Schroeder.
- History:
  - Kerberos is an authentication service developed as part of Project Athena at MIT.
  - Version 4 is designed at MIT in 1987, (Because version 4 uses DES, which is regulated by US's export law.)
  - Version 5, designed by John Kohl and Clifford Neuman, appeared as RFC 1510 in 1993 (obsoleted/superseded by RFC 4120 in 2005). Version 5 does not restrict the type of encryption used.
- Windows Server employed a variant of Kerberos as their default authentication method.
- Designed to be transparent to users and enable **Single Sign-on**

Authentication Process

Consider the scenario where the **user C** login to a workstation,
and later C wants to access the **network resource V**. There are 6 steps involved.

First, user C login to a workstation. The workstation then carries out these 2 steps:

(1) Using credential that is derived from user C's password, user C (the workstation performs this on behalf of the user) convinces AS that he is authentic.

(2) AS gives user C a **ticket**, *t1*.

The workstation should "forget" the password and keep only *t1* and a derived key.

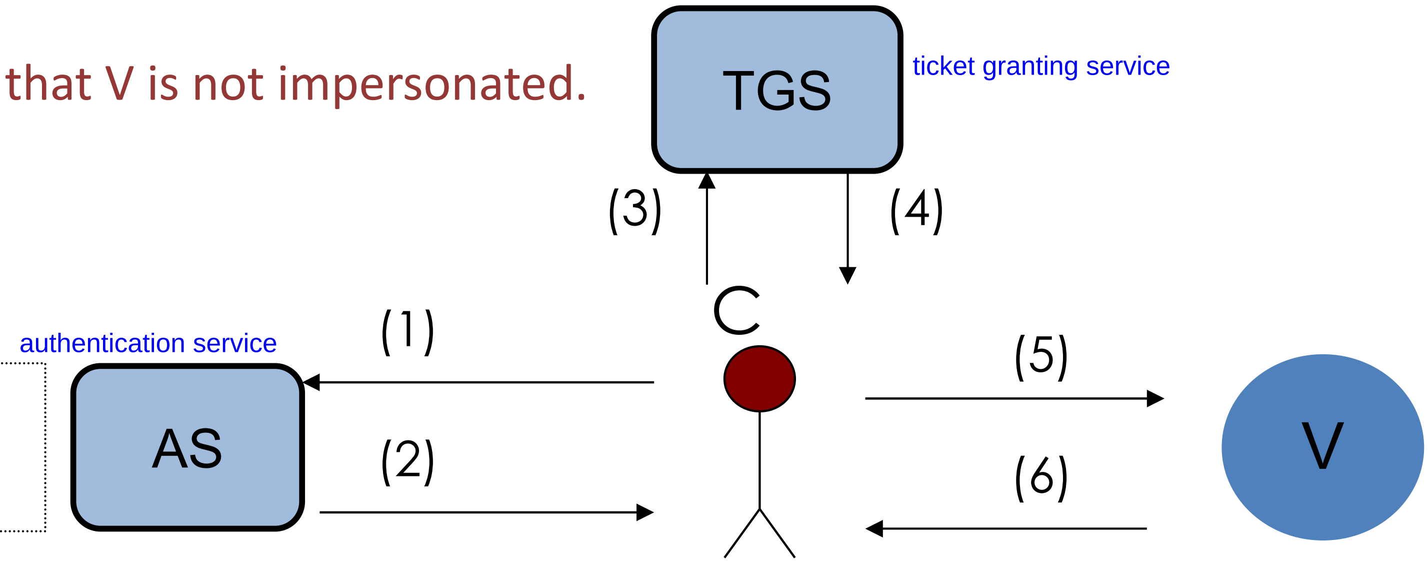
Now, user C wants to access resource V (e.g., printer).

(3) Using credential based on *t1*, user C convinces TGS that he had successfully login.

(4) TGS gives C another ticket *t2*.

(5) Using credential based on *t2*, user C convinces resource V that he is authorized to use the resources.

(6) One more step to ensure that V is not impersonated.



Note that there are no communication between TGS and AS. In many configurations,
TGS and AS residue in the same server.

Kerberos v4 Message Exchange

(1) **C → AS** $ID_c \,\|\, ID_{tgs} \,\|\, TS_1$

(2) **AS → C** $E(K_c, [K_{c,tgs} \,\|\, ID_{tgs} \,\|\, TS_2 \,\|\, Lifetime_2 \,\|\, Ticket_{tgs}])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \,\|\, ID_C \,\|\, AD_C \,\|\, ID_{tgs} \,\|\, TS_2 \,\|\, Lifetime_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

K_c: C's key shared with AS
K_{tgs}: TGS's key shared with AS
$K_{c,tgs}$: Session key for C and TGS
$Ticket_{tgs}$: Ticket Granting Ticket

(3) **C → TGS** $ID_v \,\|\, Ticket_{tgs} \,\|\, Authenticator_c$

(4) **TGS → C** $E(K_{c,tgs}, [K_{c,v} \,\|\, ID_v \,\|\, TS_4 \,\|\, Ticket_v])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \,\|\, ID_C \,\|\, AD_C \,\|\, ID_{tgs} \,\|\, TS_2 \,\|\, Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \,\|\, ID_C \,\|\, AD_C \,\|\, ID_v \,\|\, TS_4 \,\|\, Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \,\|\, AD_C \,\|\, TS_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

K_v: V's key shared with TGS
$K_{c,v}$: Session key for C and V
$Ticket_v$: Service Granting Ticket

(5) **C → V** $Ticket_v \,\|\, Authenticator_c$

(6) **V → C** $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_v = E(K_v, [K_{c,v} \,\|\, ID_C \,\|\, AD_C \,\|\, ID_v \,\|\, TS_4 \,\|\, Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \,\|\, AD_C \,\|\, TS_5])$$

(c) Client/Server Authentication Exchange to obtain service

Authentication and Secure Communication

- Key Establishment / Distribution
- Authentication using Symmetric Key
- Authentication using PKI
- Authentication using KDC
- **TLS / IPSec**

What is TLS?

- TLS stands for **Transport Layer Security** protocol
  - De facto standard for Internet security
  - The latest version is **TLS 1.3**, which was published as RFC 8446 in 2018
  - Provide **confidentiality and data integrity** between two communicating applications
  - Widely used to protect information transmitted between browsers and Web servers (**https**)
- Based on **Secure Sockets Layers** protocol, ver 3.0
  - Same protocol design, different crypto algorithms

TLS Basics

- TLS consists of two protocols
  - **Handshake** protocol
    - Agree on encryption algorithms to be used
    - Use public-key cryptography to establish a shared secret key between the two parties (e.g., client and server)
    - Described in [KPS] page 479
  - **Record** protocol
    - Use the secret key established in the handshake protocol to protect communication
- Typically TLS is used for **unilateral authentication**
  - Client authenticates server
  - Server must have digital certificate
  - Can support mutual authentication if client is equipped with digital certificate

Unilateral authentication (up to TLS 1.2)

- The handshaking steps can be grouped into 3 phases

negotiation

Client

Masterkey
= h(S,r_A, r_B)

(1) Client hello, $\mathbf{r_A}$

(2) Server hello, $\mathbf{r_B}$

negotiate the cipher suite to be used.

(3) Server's certificate

(4) $\text{Server}_{\text{public\_key}}\{\mathbf{S}\}$

Key exchange+ authentication

client says he is done.

(5) ChangeCipherSpec

(6) A finish message, containing a MAC over the previous handshake messages, computed using the master key.

server says he is done.

(7) ChangeCipherSpec

(8) A finish message, containing a MAC over the previous handshake messages, computed using the master key.

Server

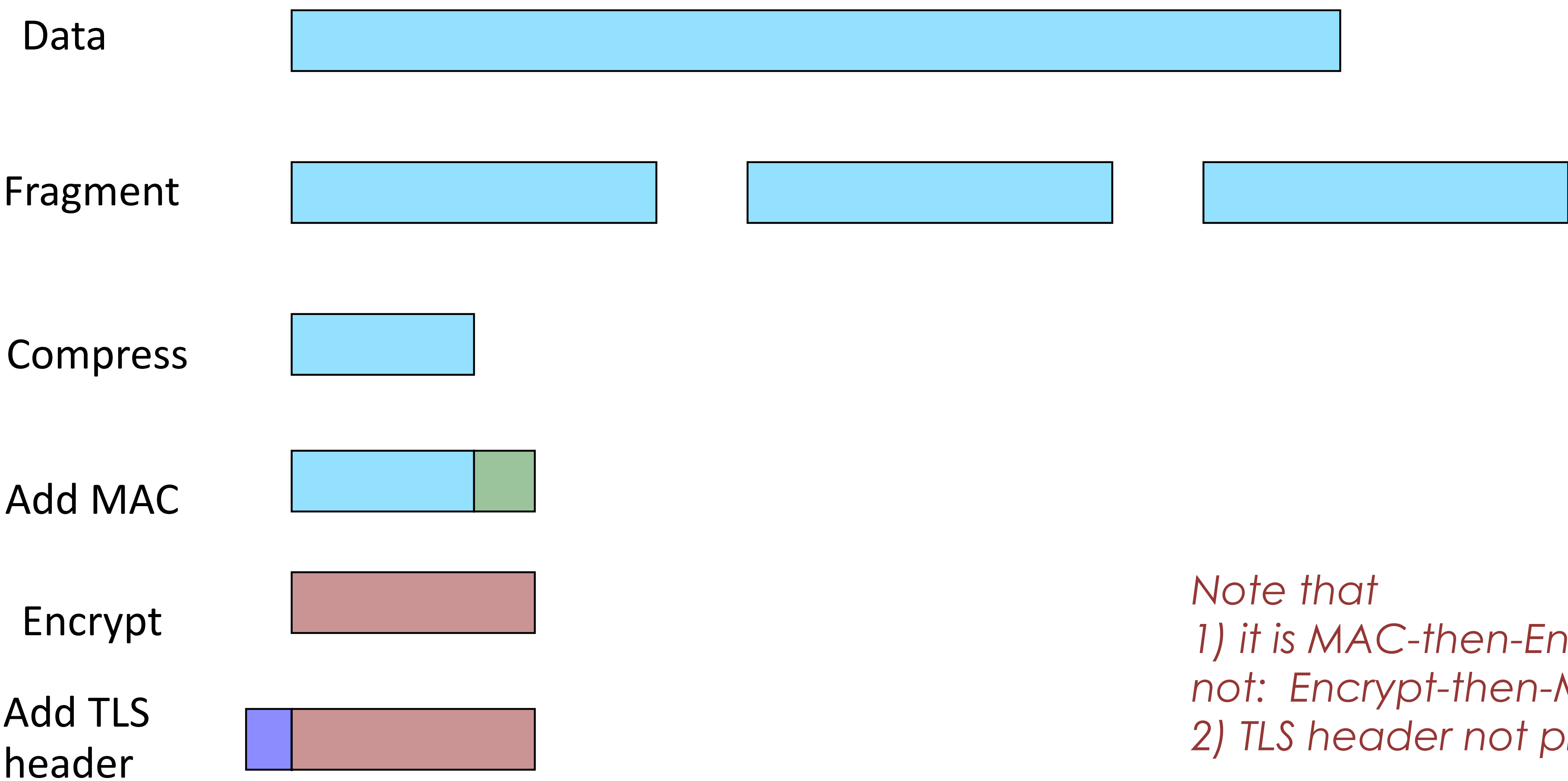
- Both r_A, r_B, and S are to be randomly generated. (TLS recommends that the first few bits of r_A, r_B to be derived from time, so that it is assured that they will be different for different sessions.)
- Client indicates in step(1) **a list of suggested cipher suites and compression methods** he can use.
- Server chooses **a cipher and compression from the list** (in step(1)) and indicates his choice in step (2).
- Step (4) consists of a random number S (known as the premaster key) encrypted with the server public key.
- The **master key** is derived by both parties from S, r_A, r_B using a hash function after step (4)

$$\mathbf{h(\ S,\ r_A,\ r_B\ )}$$

- Messages in step (6)&(8) include a MAC of all previously exchanged messages. The key of the MAC is derived from the master key.
  - This step helps to prevent the ***"downgrade"* attack**, which mislead entities to use weak cipher suite.
- Essentially, by sending "ChangeCipherSpec" in step (5), the client tell the server:
  - *"Everything I tell you from now on will be authenticated (and encrypted if required)"*. Similarly, in step (7).

Encrypted Record

- Records sent after a “ChangeCipherSpec” are cryptographically protected with the negotiated cipher suite. Note that the “finish” messages (step (6)&(8)) during handshake are protected.



Note that
1) it is MAC-then-Encrypt.
not: Encrypt-then-MAC
2) TLS header not protected

Renegotiation Attacks against SSL/TLS

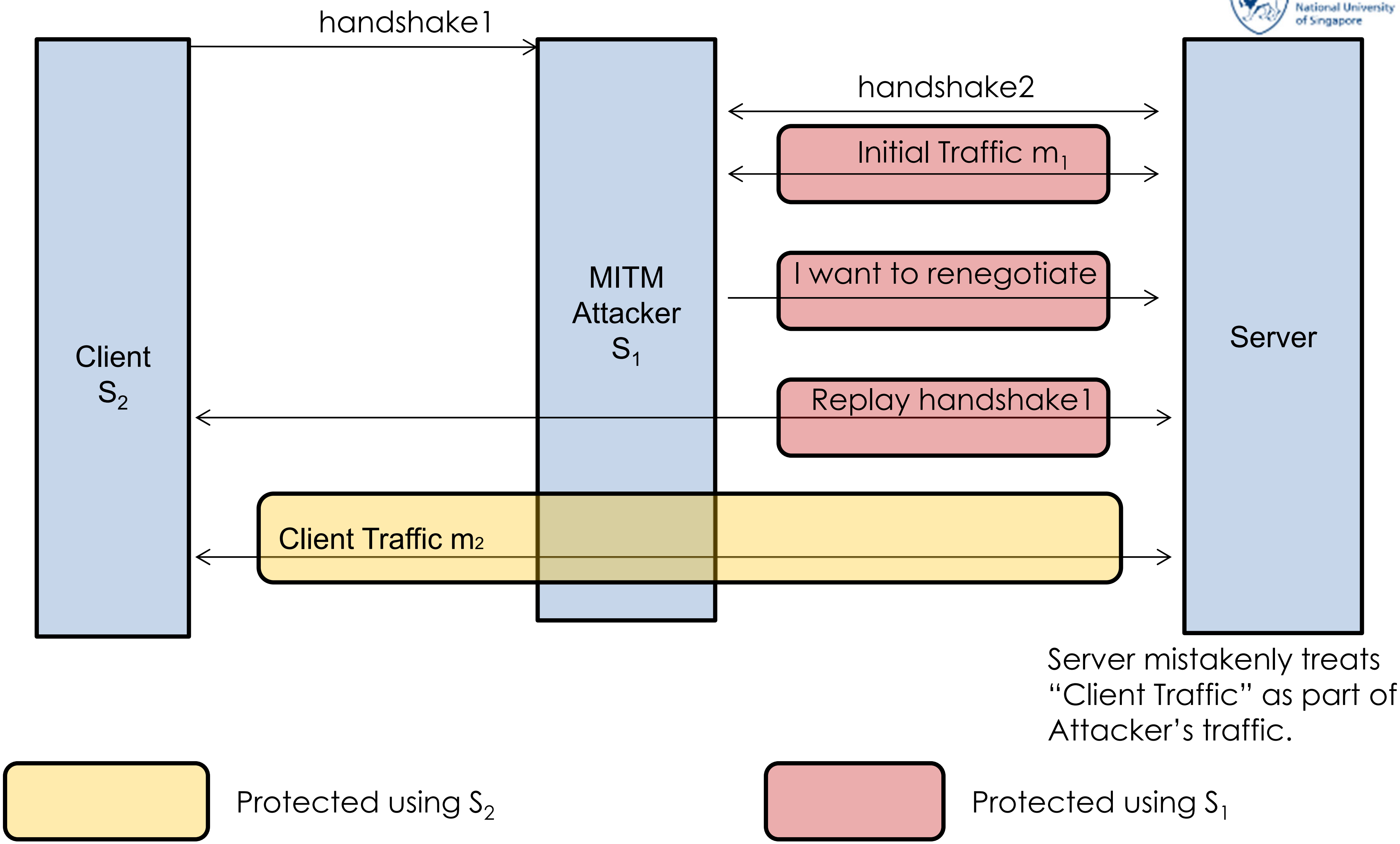
- https://owasp.org/www-pdf-archive/OWASP_-_TLS_Renegotiation_Vulnerability.pdf
- https://www.ietf.org/proceedings/76/slides/tls-7.pdf

- Attacker is a man-in-the-middle.

- This attack shows how a man-in-the-middle can compromise the message integrity. Note that confidentiality is still preserved.

- Patch (RFC5746)

Renegotiation Attacks against SSL/TLS



Client sends m_2, but Server mistakenly thought that the message is the concatenated $m_1 \| m_2$

Renegotiation attack in https

Many web applications first authenticate the client using userid/password. Next, server gives the client a **cookie** to be stored in the client-side.

Subsequently, the client just has to present the cookie to get authenticated and no userid/password is required. This frees the users from repeated logging-in.

Note that the cookie is associated with the **client's account** (e.g., payment method etc.) upon login.

Now, suppose to order pizza, one has to send the following via https.

```
GET /pizza?toppings=sausage;address=homeaddress HTTP/1.1
Cookie: MYcookie
```

Step1: The attacker sends:

```
GET /pizza?toppings=sausage;address=attackeraddress
X-Ignore-This:
```

Attacker initiates "renegotiation" under TLS protocol.

Step2: Victim handshakes (TLS) with server.

Victim sends

```
GET /pizza?toppings=sausage;address=CLIENTaddr HTTP/1.1
Cookie: CLIENTcookie
```

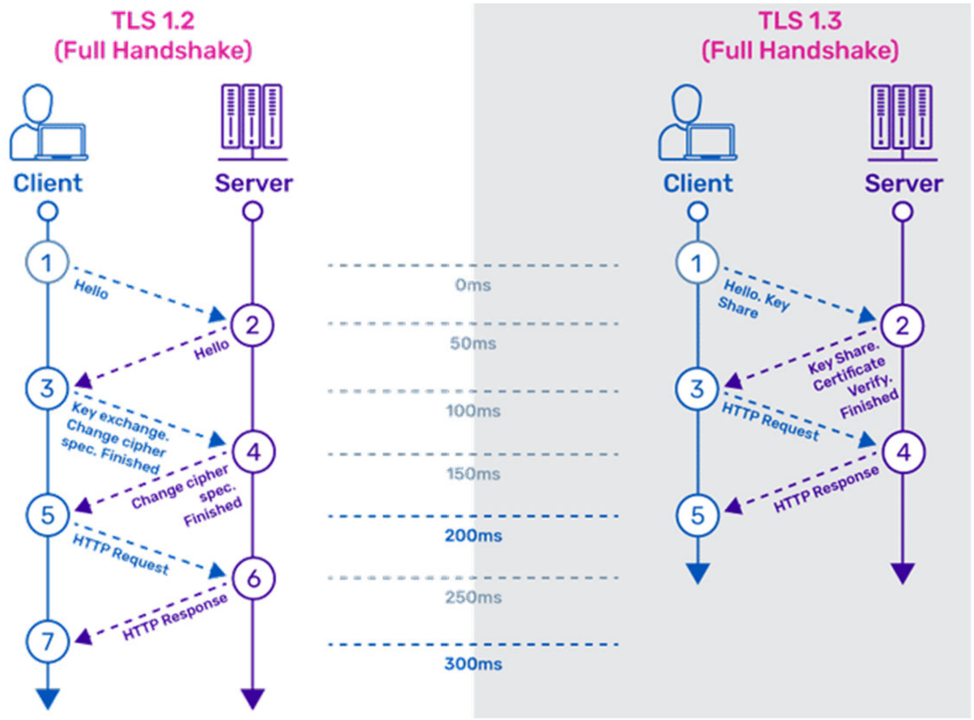
The server, who treats data sent during the two step as one single session, mistakenly takes the following as the message.

```
GET /pizza?toppings=sausage;address=attackeraddress
X-Ignore-This: GET /pizza?toppings=sausage;address=CLIENTaddr HTTP/1.1
Cookie: CLIENTcookie
```

The server then delivers the pizza to "attackeraddress" and charges it under the account associated with the CLIENTcookie!!

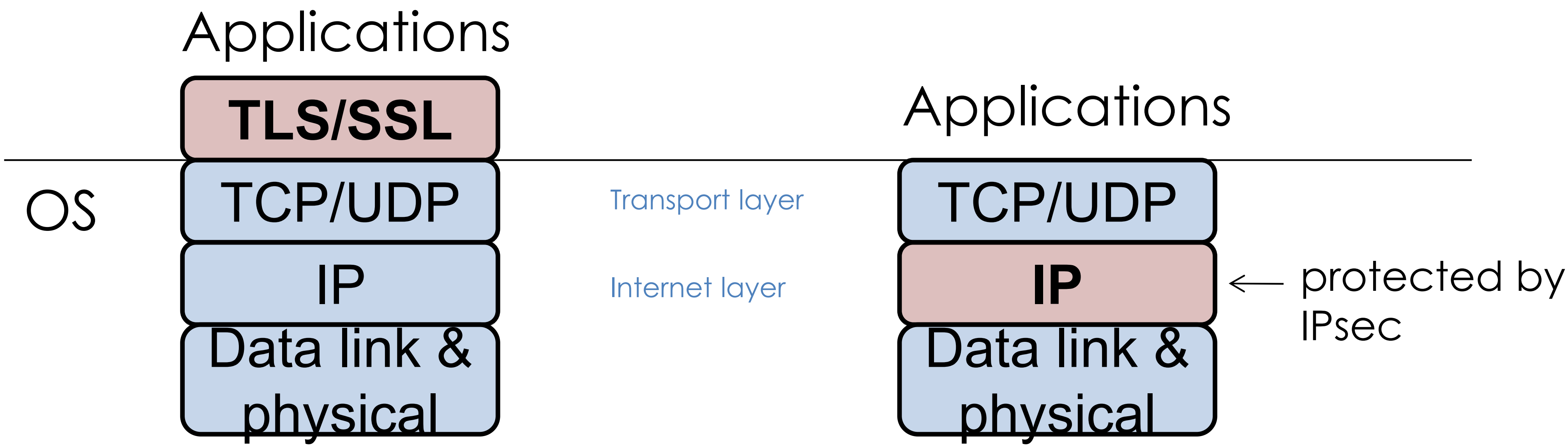
TLS 1.3

- Cipher suite that has known vulnerability or does not support perfect forward secrecy is dropped.
  - RC4 algorithm, CBC mode etc. are not on the list
- "Renegotiation" feature is dropped.
- Major change in handshake



https://www.a10networks.com/glossary/key-differences-between-tls-1-2-and-tls-1-3/

TLS/SSL vs IPSec



To introduce TLS/SSL,
• OS does not need to be modified,
but
• Network applications have to be modified (e.g., HTTP vs HTTPS)

To introduce IPSec,
• OS has to be modified,
but
• Network applications does not need to be modified.

Protocols in IPSec

- Consists of 3 protocols
  - Internet Key Exchange (IKE)
    - Authentication and key establishment using STS Diffie-Hellman
  - Authentication Header (AH)
    - Provide integrity (including part of IP header)
  - Encapsulating Security Payload (ESP)
    - Provide confidentiality and (optionally) integrity
- **Security Association (SA)** is information indicating the parameters used between the sender and receiver
- IKE is first executed to authenticate each other and establish SA.
- Communication is protected by either AH, ESP, or both (ESP followed by AH)

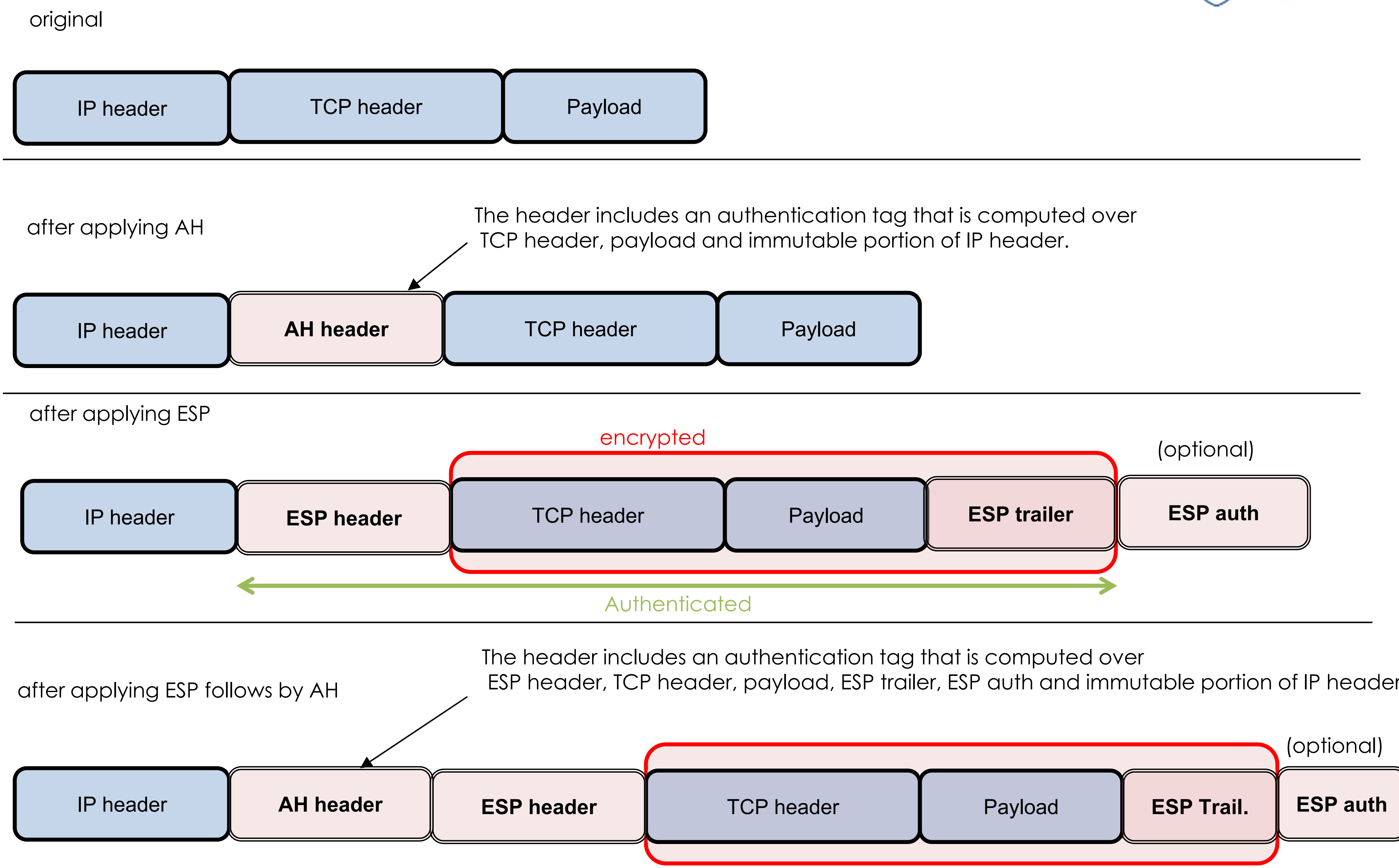
Security Association

- SA contains information like:
  - ESP information: symmetric key, encryption algorithm etc.
  - AH information: symmetric key, authentication algorithm etc.
  - Lifetime of SA
  - Sequence number (to counter replay attack)
- SA is uniquely identified by
  - Security parameter index
  - IP destination address
  - Protocol identifier (whether SA is for AH or ESP)

Tunnel and Transport Mode

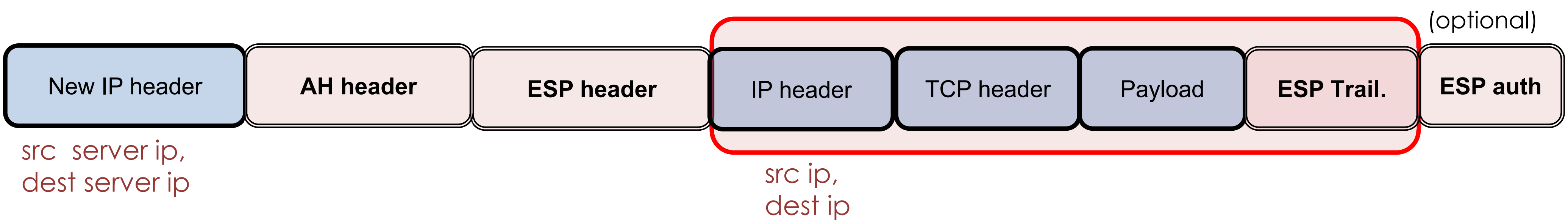
- Transport Mode
  - Provide protection in upper layers (i.e., transport layer and above)
- Tunnel Mode
  - Provides protection to the entire IP packet

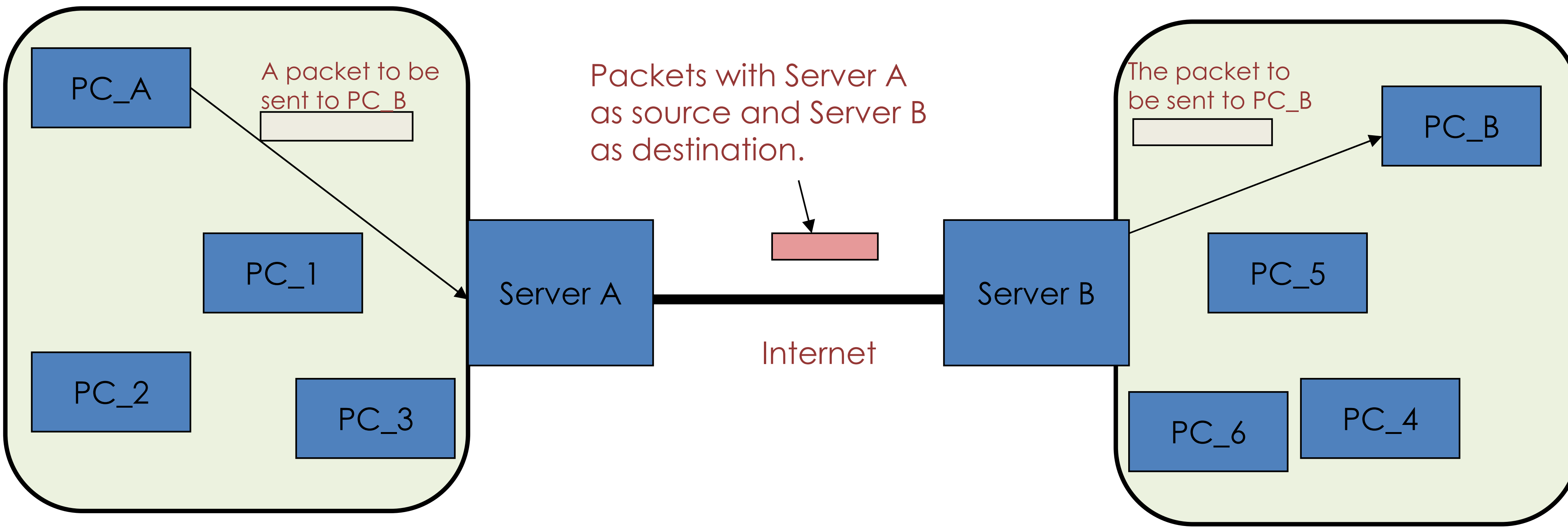
Transport Mode



Tunnel mode

- The original IP header is also protected





QUESTIONS?