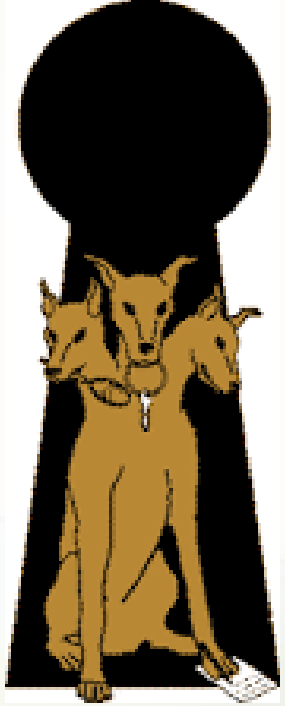


KERBEROS

Mukesh Chinta
Assistant Professor, Dept of CSE
V R Siddhartha Engineering College

What is Kerberos???

- Kerberos is a computer network authentication protocol that works on the basis of 'tickets' to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner.
- The protocol was named after the character Kerberos (or Cerberus) from Greek mythology, the ferocious three-headed guard dog of Hades (hellhound).
- Its designers aimed it primarily at a client–server model and it provides mutual authentication—both the user and the server verify each other's identity.
- Kerberos protocol messages are protected against eavesdropping and replay attacks.
- Massachusetts Institute of Technology (MIT) developed Kerberos to protect network services provided by Project Athena.

Need for Kerberos!!!

☺In an open distributed environment, users at workstations wish to access services on servers distributed throughout the network.

☺In such environments, servers should be able to restrict access to authorized users and be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services.

☺Three kinds of threats exist:

- *User pretends to be another user.*
- *User alters the network address of a workstation.*
- *User eavesdrop on exchanges and uses a replay attack.*

Motivation of Kerberos

- Today, more common is a distributed architecture consisting of dedicated user Work Stations (clients) and distributed or centralized servers.
- Three approaches envisioned for security
  1. Each WS assures the identity of its user and each server enforces a security policy based on user ID.
  2. Client systems authenticate themselves to servers, but servers trust Client systems concerning the identity of its user.
  3. The Client proves user's identity for each service invoked and the servers prove its identity to the clients.
- Kerberos supports this third approach.

Requirements of Kerberos

- The requirements of Kerberos based on the first published report [STEI88]:

☼ **Secure** : A network eavesdropper can't obtain the necessary information to impersonate a user

☼ **Reliable** : A distributed server architecture should be employed with one system to back up another

☼ **Transparent** : Users should not know the authentication process beyond entering a password.

☼ **Scalable** : The system should support large number of clients and severs (i.e. a modular, distributed architecture)

- The overall scheme of Kerberos is a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78].

Terminology of Kerberos

- Terms:
  - C = Client
  - AS = authentication server
  - V = server
  - ID_C = identifier of user on C
  - ID_V = identifier of V
  - P_c = password of user on C
  - AD_c = network address of C
  - K_v = secret encryption key shared by AS and V
  - TS = timestamp
  - || = concatenation

- Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service.
- We look through several hypothetical dialogues with increasing complexity to counter different security vulnerabilities.

A Simple Authentication Dialogue

- In an unprotected network environment, an opponent can pretend to be another client and obtain unauthorized privileges on server machines (impersonation). To counter this threat, servers must be able to confirm the identities of clients who request service, but it places a heavy burden on each server.
- An alternative is to use an **authentication server (AS)** that knows the passwords of all users and stores these in a centralized database. Also, the AS shares a unique secret key with each server

(1) C → AS: $ID_c \| P_c \| ID_v$

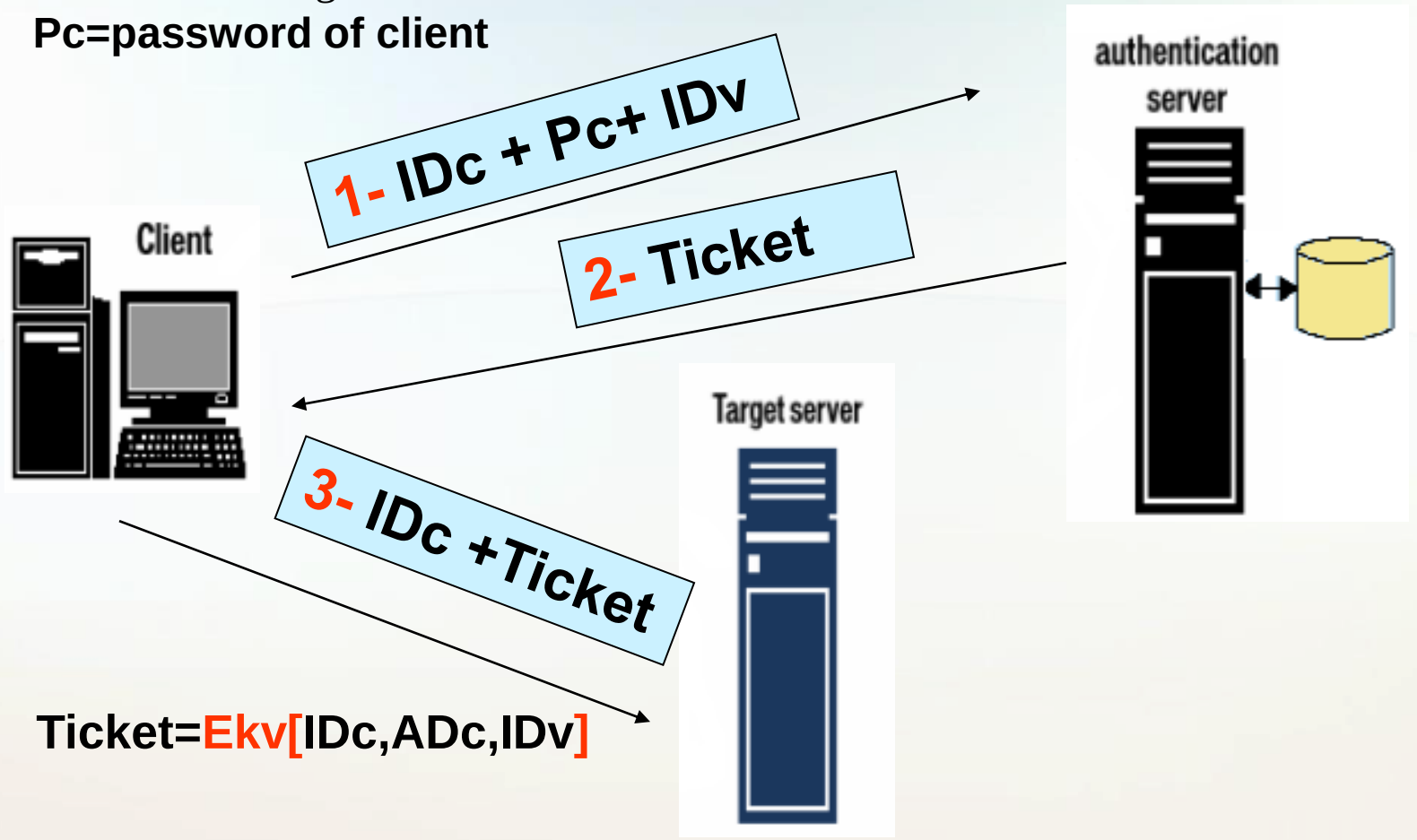
(2) AS → C: Ticket

(3) C → V : $ID_c \| Ticket$

$Ticket = \mathbf{E_{Kv}}[ID_c \| \mathbf{AD_c} \| ID_v]$

- The user logs on to a WS and requests access to server V
- The client module C requests user's password
- Then C sends message(1) to AS
- AS sendS a ticket to convince V of the user's authenticity

- The user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password.



- The AS now verifies the supplied password and access permissions of V for this user. Then, AS creates a ticket that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server (V) and sent back to C.
- With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket.

- Frequent requests to enter user's password
  - Suppose each ticket can be used only once
    - ►A user enters a password to get a ticket each time the user wants access to V
  - Suppose the tickets are reusable to improve the matters
    - ►A user needs a new ticket for every different service and hence be required to enter a password
- A plaintext transmission of password in message (1)
  - An opponent could capture the password and use any service accessible to the victim

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the **ticket-granting server (TGS).**

A More Secure Dialogue

Once per user logon session:

(1) C → AS: $ID_C \| ID_{tgs}$

(2) AS → C : $E(K_C, Ticket_{tgs})$

Once per a type of service:

(3) C → TGS: $ID_C \| ID_V \| Ticket_{tgs}$

(4) TGS → C: $Ticket_v$

Once per a service session:

(5) C → V: $ID_C \| Ticket_v$

$Ticket_{tgs} = E(K_{tgs}, [IDc \| ADc \| ID_{tgs} \| TS_1 \| Lifetime_1])$

$Ticket_v = E(K_v, [IDc \| ADc \| IDv \| TS_2 \| Lifetime_2])$

TGS issues tickets for services to users who have been authenticated to AS

- Thus, the user first requests a ticket-granting ticket (TGT) from AS.
- TGT is saved in the client module of WS and used to authenticate the user itself to TGS for each access to a new service.
- The service-granting ticket (SGT) issued by TGS is saved and used to authenticate its user to a server for a particular service.

Once per user logon session:

(1) C → AS: $ID_C \| ID_{tgs}$

(2) AS → C : $E(K_C, Ticket_{tgs})$

$$\mathbf{Ticket_{tgs} = E(K_{tgs}, [IDc \| Adc \| Id_{tgs} \| TS_1 \| Lifetime_1])}$$

- The client requests a Ticket-granting Ticket ($Ticket_{tgs}$) by sending msg(1) to AS containing users ID and TGS ID.
- The AS responds with a ticket encrypted with a key(K_c) derived from user's password.
- When this response arrives at the client, it prompts the user to enter his/her password and generates the key and decrypts the message. If the correct password is supplied, the ticket is successfully recovered.
  - ***As only the correct user knows the password, the user would be the one to recover the ticket. The problem of sending the password in the open has been solved.***
  - ***Now, the client can use this ticket to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable. Also as its encrypted by a secret key shared between AS and TGS, the ticket cannot be altered.***
  - ***To counter the scenario where an opponent capturing the login message, then capturing and reusing the ticket to spoof the TGS, a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid are added.***

Once per type of service:

(3) C → TGS: $ID_C \| ID_V \| Ticket_{tgs}$

(4) TGS → C: $Ticket_v$

$Ticket_v = E(K_v, [IDc \| ADc \| IDv \| TS2 \| Lifetime2])$

- The client requests a service-granting ticket (SGT) for the user with a message(3) containing the user's ID, the ID of the desired service, and the ticket-granting ticket.
- The TGS decrypts the incoming ticket using a key shared only by the AS and the TGS (K_{tgs}) and verifies the success of the decryption by the presence of its ID. It also checks the validity by the lifetime and authenticates the user by checking the user ID and the network address with the incoming information. If the user is permitted to access V, the TGS issues a ticket to grant access to the requested service.
- The SGT has the same structure as the TGT containing the timestamp and lifetime. It is encrypted using a secret key known only to the TGS and server preventing alteration.

Once per type of service:

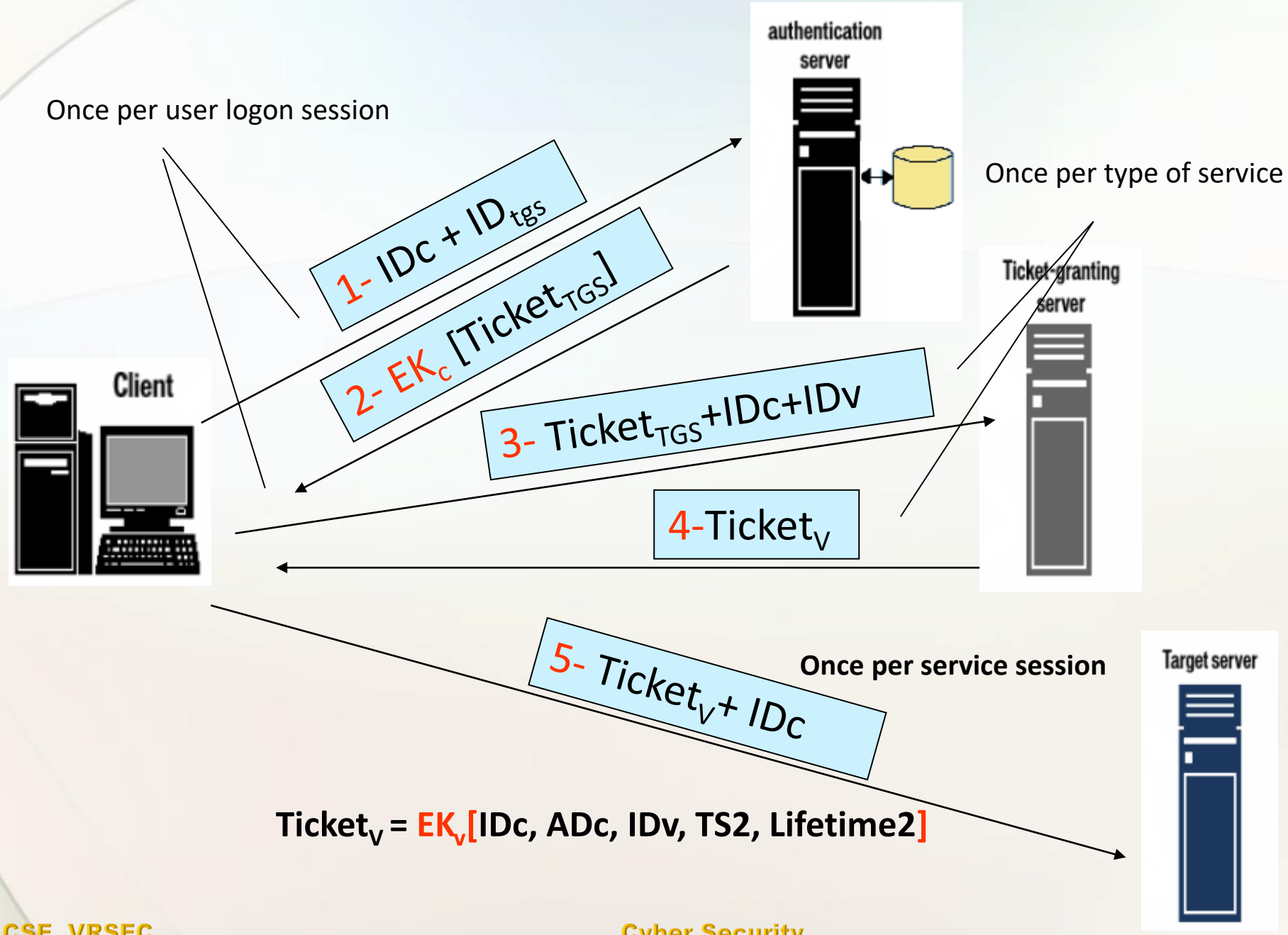
(5) C → V: $ID_C \| Ticket_v$

$Ticket_v = E(K_v, [Id_c \| Ad_c \| Id_v \| TS2 \| Lifetime2])$

With a particular service-granting ticket (SGT), the client can gain access to the corresponding service with step 5.

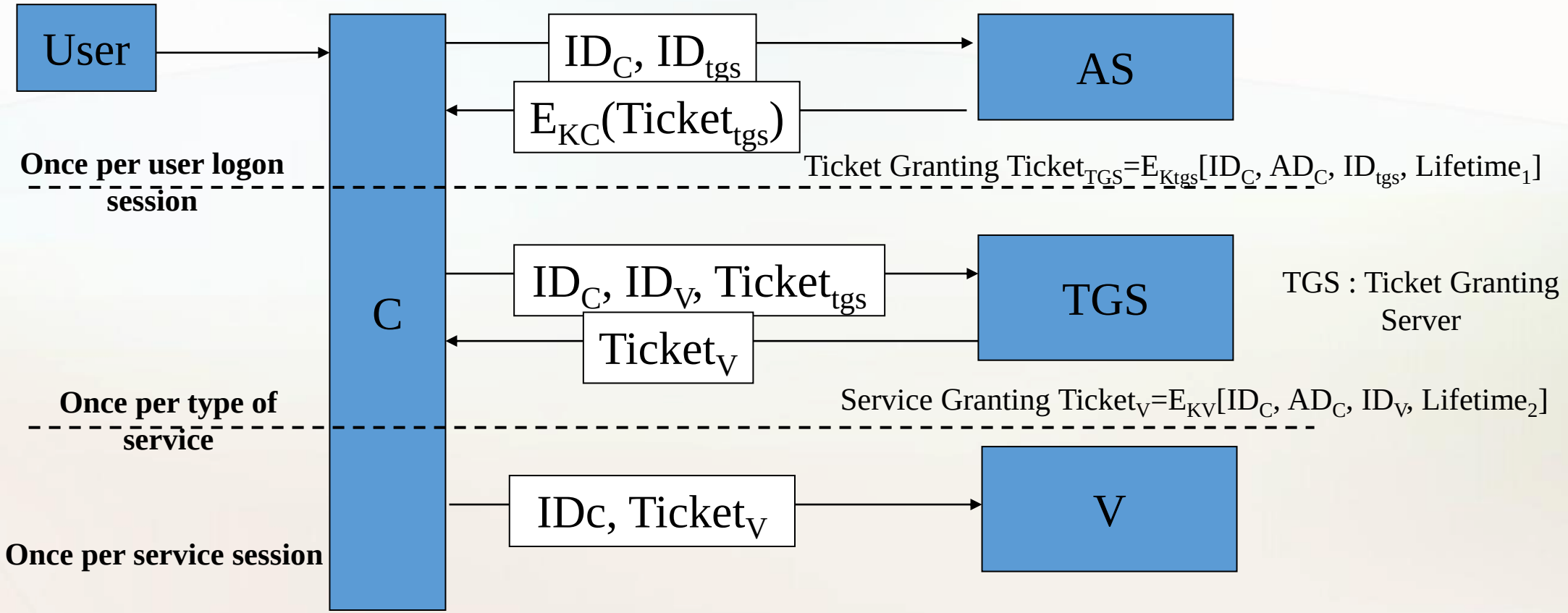
- The client requests access to a server for the user with message(5)
- The server authenticates by using the contents of the SGT
- The scenario satisfies the two requirements:
  - Only one password query
  - No transmission of the user password in plaintext

$\underline{Ticket_{TGS}} = EK_{tgs}[IDc, ADc, ID_{tgs}, TS1, LifeTime1]$



$Ticket_V = EK_v[IDc, ADc, IDv, TS2, Lifetime2]$

Authentication Dialogue -Summary



Version 4 Authentication Dialogue

Two additional problems in above scenario:

1. ***The lifetime associated with the TGT***
   - **Too short → frequent prompts for entering the password**
   - **Too long → replay attack after capturing the ticket**
   - An opponent could eavesdrop and capture a copy of the ticket-granting ticket, wait for the user to log out, forge the users network address and send message 3 to TGS. This gives the opponent unlimited access to all the users resources. Same scenario can occur with the SGT also.
   - TGS or AS must prove that the person using the ticket is the same person to whom that ticket was issued.
2. ***The requirement for servers to authenticate themselves to users.***
   - Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The impersonated server could deny the true service to the user

(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

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

(4) TGS → C $E(K_{c,tgs} [K_{c,v} \| ID_v \| TS4 \| Ticket_v])$

$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_c \| AD_c \| ID_{tgs} \| TS_2 \| Lifetime2])$

$Ticket_v = E(K_v, [K_{c,v} \| ID_c \| AD_c \| ID_v \| TS_4 \| Lifetime4])$

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

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

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

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

$Ticket_v = E(K_v, [K_{c,v} \| ID_c \| AD_c \| ID_v \| TS_4 \| Lifetime4])$

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

(c) Client/Server Authentication Exchange to obtain service

(1) C → AS $IDc \| ID_{tgs} \| TS_1$

(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

- The client sends the message(1) to AS requesting access to the TGS.
- The AS responds with a message, encrypted with a key derived from the user's password (K_c), that contains the ticket. The encrypted message also contains a copy of the session key, ($K_{c,tgs}$), for C and TGS.
- As this session key is inside the message encrypted with K_c, only the users client can read it and the same key is included in the ticket, which can be read only by TGS therefore securely delivering the secret key.
- Message(1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

(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 SGT

- C sends the TGS a message that includes the ticket plus the ID of the requested service. In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp.
- Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime.
- The TGS can decrypt the ticket with the key that it shares with the AS and obtain the session key. The TGS uses the session key to decrypt the authenticator and verifies the contents.
- If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner.
- Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.
- The reply from the TGS in message (4) follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself contains the same session key.

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

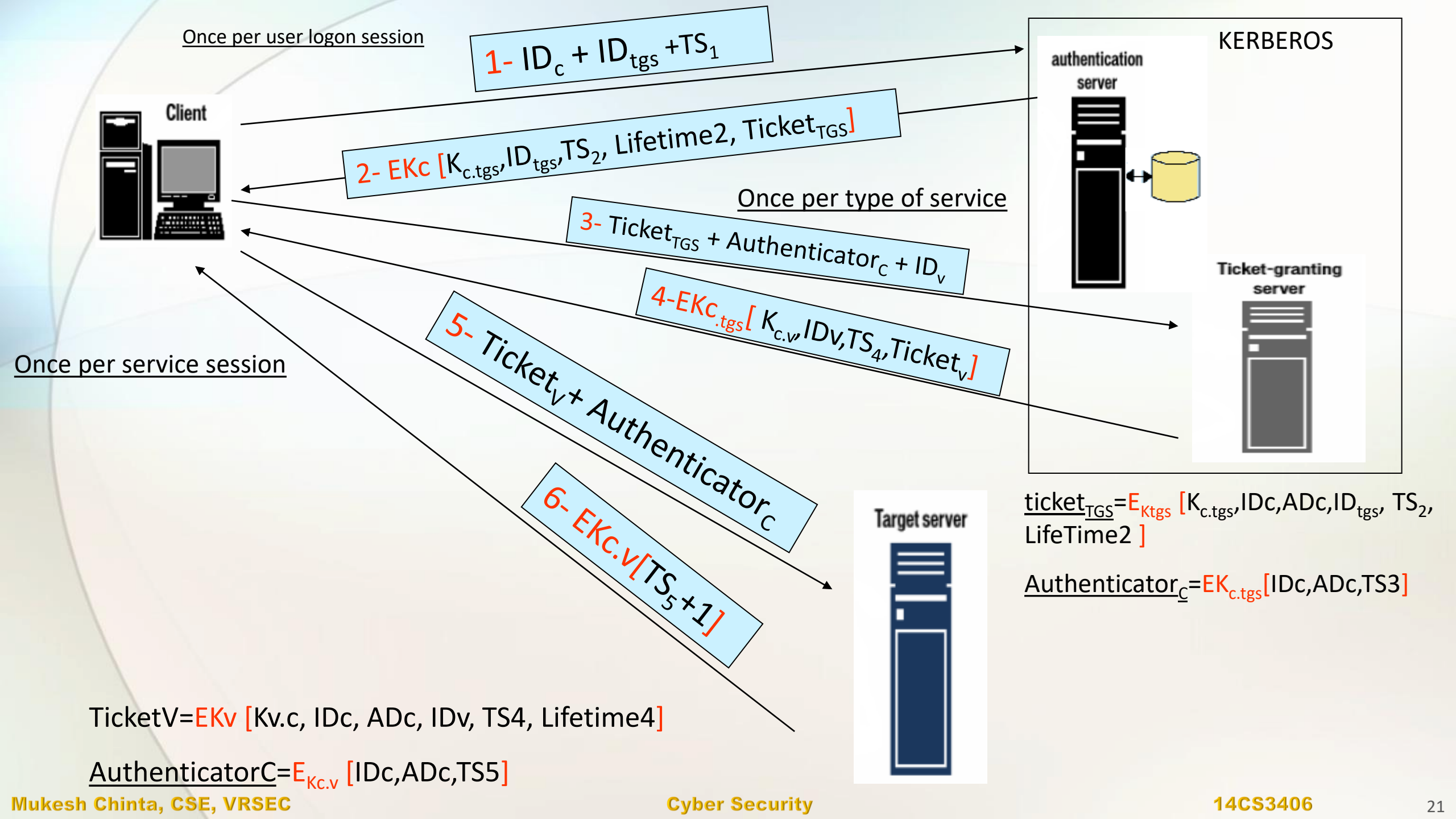
(6) V → C $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

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

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

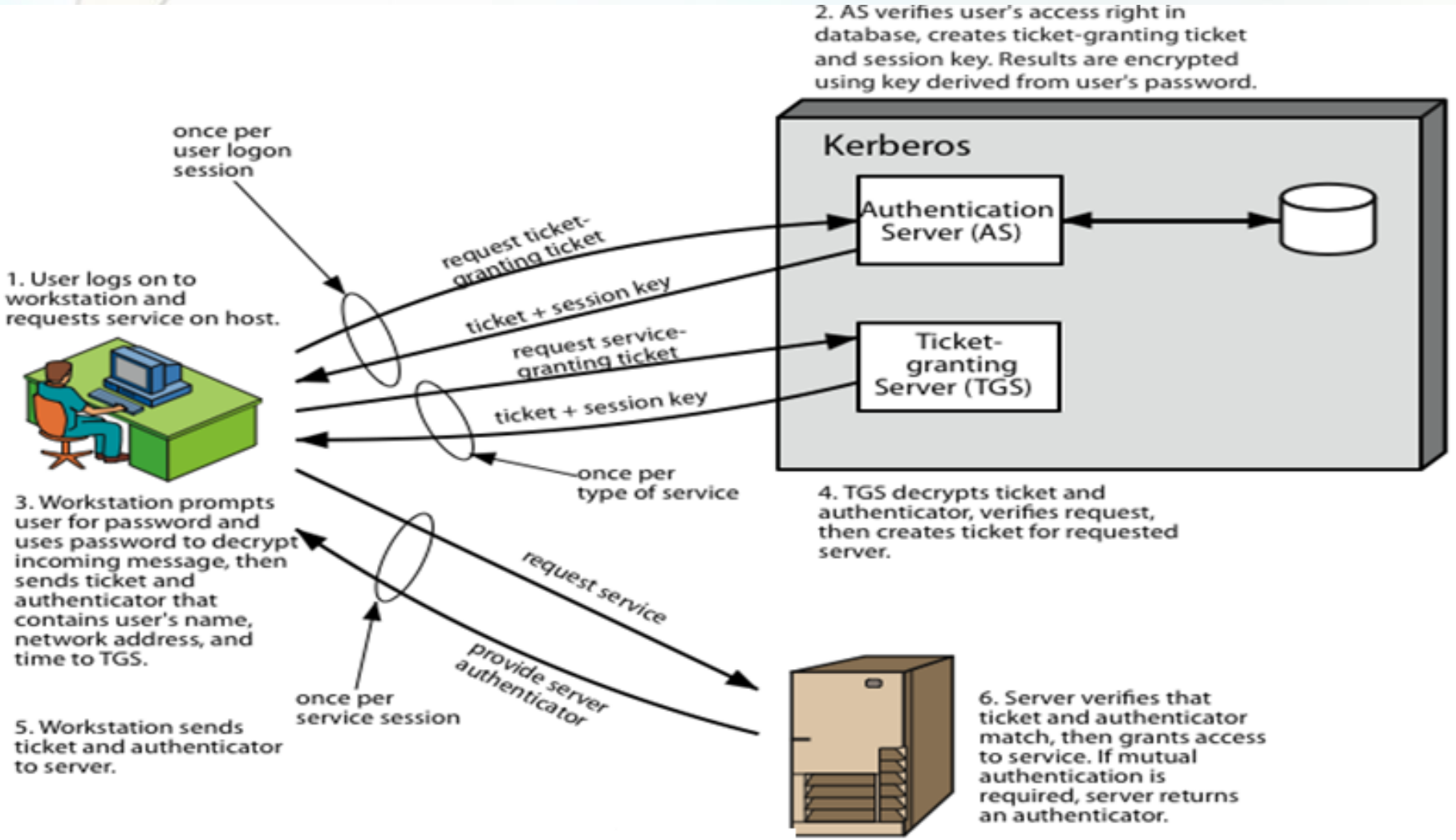
(c) Client/Server Authentication Exchange to obtain service

- C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.
- If mutual authentication is required, the server can reply as shown in message (6).
- The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.
- At the end, both client and server share a secret key, which can be used for future.


Once per user logon session
1- $ID_c + ID_{tgs} + TS_1$
KERBEROS
authentication server
Client
2- $EKc [K_{c.tgs}, ID_{tgs}, TS_2, Lifetime2, Ticket_{TGS}]$
Once per type of service
3- $Ticket_{TGS} + Authenticator_C + ID_v$
Ticket-granting server
4-$EKc_{.tgs}[K_{c.v}, IDv, TS_4, Ticket_v]$
Once per service session
5- $Ticket_V$+ $Authenticator_C$
6- $EKc.v[TS_5+1]$
Target server
$ticket_{TGS}=E_{Ktgs} [K_{c.tgs}, IDc, ADc, ID_{tgs}, TS_2,$ LifeTime2]
$Authenticator_C=EK_{c.tgs}$[IDc,ADc,TS3]
TicketV=EKv [Kv.c, IDc, ADc, IDv, TS4, Lifetime4]
AuthenticatorC=$E_{Kc.v}$ [IDc,ADc,TS5]

KERBEROS Environment

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:



Overview of Kerberos

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server. Such an environment is referred to as a Kerberos realm. A Kerberos realm is a set of managed nodes that share the same Kerberos database.

KERBEROS REALMS AND MULTIPLE KERBERI

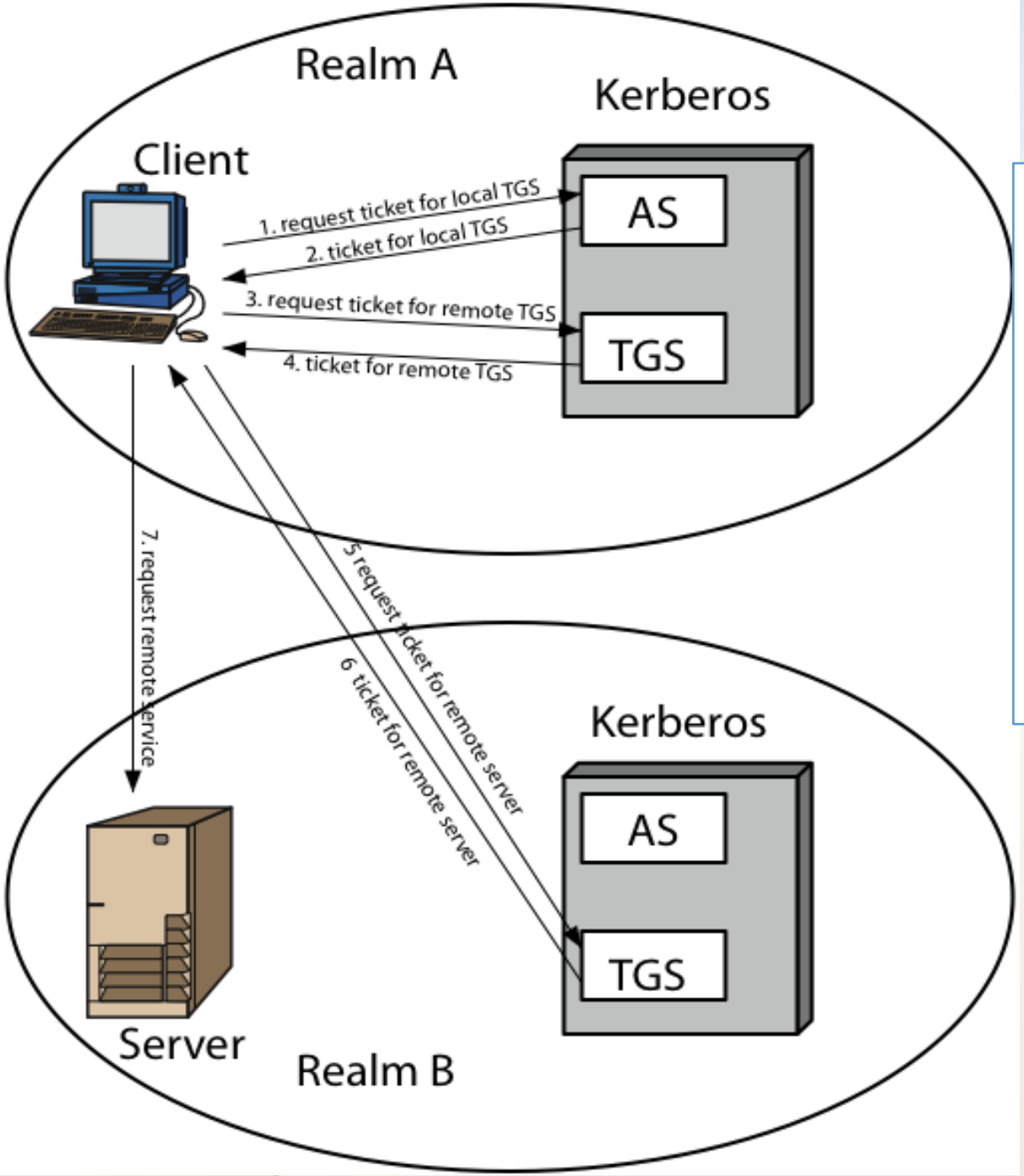
What will happen when users in one realm need access to service from other realms?:

- Kerberos provide inter-realm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

- In addition, Kerberos server in one realm should trust the one in other realm to authenticate its users. The second also trusts the Kerberos server in the first realm.

- A user, wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm).
- The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.
- The ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.



1. C → AS : $ID_c \| ID_{tgs} \| TS_1$
2. AS → C : $E(K_c,[K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$
3. C → TGS : $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_c$
4. TGS → C : $E(K_c,_{tgs} [K_{c,tgsrem} \| ID_{tgsrem} \| TS_4 \| Ticket_{tgsrem}])$
5. C → TGS_{rem} : $ID_{vrem} \| Ticket_{tgsrem} \| Authenticator_c$
6. TGS_{rem} → C : $E(K_{c,tgsrem}, [K_{c,vrem} \| ID_{vrem} \| TS_6 \| Ticket_{vrem}])$
7. C → V_{rem} : $Ticket_{vrem} \| Authenticator_c$

One problem presented by the foregoing approach is that it does not scale well to many realms.

Kerberos Version 5

- Kerberos version 5 is developed during mid 1990's and efforts to make Kerberos general-purpose.
- Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4 [KOHL94].
- Version 5 is intended to address the limitations of Version 4 in two areas :
  - Environment shortcomings, due to development for use within the Project Athena environment, not for general purpose
  - Technical deficiencies in the version 4 protocol itself

Environment Shortcomings of Version 4

1. Encryption system dependence : Version 4 requires the use of DES.
   - In Version 5, - Any encryption technique may be used
   - Encryption type identifier is tagged with ciphertext
   - Encryption Keys are tagged with type and a length to be used in different algorithms
2. Internet protocol dependence : the use of IP address only
   - In Version 5, Network addresses are tagged with type and length (e.g. ISO)
3. Message byte ordering : least or most significant byte ordering chosen by the sender of a message
   - Version 5 defines use of Abstract Syntax Notation One (ASN.1) and Basic Encoding rules (BER) for unambiguous ordering.
4. Ticket lifetime : an 8-bit quantity in units of 5 min (max= $2^8 \times 5 = 1280$ min)
   - use of explicit start and end time for arbitrary lifetime
5. Authentication forwarding : no forwarding
   - V5 allows credentials issued to one client to be forwarded to some other host & used by some other client
6. Interrealm authentication : N^2 Kerberos-to-Kerberos relationships
   - Version 5 supports a method with fewer relationships

Technical Deficiencies of Version 4

1. Double encryption : the tickets encrypted twice in messages 2 & 4
   - No double encryption on tickets in Version 5
2. PCBC encryption : use of nonstandard PCBC (Propagating Cipher Block Chaining) mode of DES
   - Its vulnerability has been demonstrated involving interchange of ciphertext blocks
   - PCBC was intended to provide an integrity check as part of encryption operation
   - Version 5 provide explicit integrity mechanisms, allowing the standard CBC mode for encryption
3. Session keys : possibility of replay attack by repeated uses of the same ticket
   - A subsession key for C and V is allowed to be used only for that connection
4. Password attack : Both versions are weak to this attack

The key is generated based on user's password

- The password is limited to characters in a 7-bit ASCII
- An opponent attempts to decrypt a message by trying various passwords
- Version 5 provides a mechanism "preauthentication" to make the attack more difficult (but not preventing it)

Version 5 Authentication Dialogue

(1) C → AS Options ll IDc || $Realm_c$ || ID_{tgs} ll Times ll $Nonce_1$

(2) AS → C $Realm_c$ ll IDc ll $Ticket_{tgs}$ || $E(K_c, [K_{c,tgs}$ ll Times ll $Nonce_1$ ll $Realm_{tgs}$ ll $ID_{tgs}])$

$Ticket_{tgs} = E(K_{tgs},[Flags$ ll $K_{c,tgs}$ || $Realm_c$ ll ID_c ll AD_cll Times])

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

(3) C → TGS Options ll ID_v ll times ll $Nonce_2$ ll $Ticket_{tgs}$ ll $Authenticator_c$

(4) TGS→ C $Realm_c$ ll ID_c ll $Ticket_v$ ll $E(K_{c,tgs}, [K_{c,v}$ ll Times ll $Nonce_2$ ll $Realm_v$ ll $ID_v])$

$Ticket_{tgs} = E(K_{tgs},[Flags$ ll $K_{c,tgs}$ ll $Realm_c$ ll ID_c ll AD_c ll Times])

$Ticket_v = E(K_v,[Flags$ ll $K_{c,v}$ ll $Realm_c$ ll ID_c ll AD_c ll Times])

$Authenticator_c = E(K_{c,tgs}$ [ID_c ll $Realm_c$ ll TS_1])

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

(5) C → V Options ll $Ticket_v$ ll $Authenticator_c$

(6) V → C $E_{Kc,v}$, [TS_2 ll Subkey ll Seq#]

$Ticket_v = E(K_v,[Flags$ ll $K_{c,v}$ ll $Realm_c$ ll ID_c ll AD_c ll Times])

$Authenticator_c = E(K_{c,v}$, [$ID_c$ ll $Realm_c$ ll TS_2 ll Subkey ll Seq#])

(c) Client/Server Authentication Exchange to obtain service

(1) C → AS Options || IDc || $Realm_c$ || ID_{tgs} || Times || $Nonce_1$

(2) AS → C $Realm_c$ || IDc || $Ticket_{tgs}$ $E(K_c,[K_{c,tgs}$ || Times || $Nonce_1$ || $Realm_{tgs}$ || $ID_{tgs}])

$Ticket_{tgs} = E(K_{tgs},[Flags || K_{c,tgs} || Realm_c || ID_c || AD_c || Times])$

- Message (1) is a client request for a TGT
  - Realm: Indicates the realm of the user
  - Options : used to request for certain flags to be set in returned ticket
  - Times: Used by client to request the following time settings in the ticket
    - From: The desired start time for the requested ticket
    - Till: the requested expiration time for the requested ticket
    - Rtime: requested renew-till time
  - Nonce : is a random value to be repeated in message(2) to counter replay attack
- Message (2) returns a TGT, identifying information for the client and a block encrypted using the encryption key based on the users password.
  - Flags : reflect the status of this ticket and requested options with new functionality added to Version 5

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

(3) C → TGS $\quad Options \| ID_v \| times \| Nonce_2 \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $\quad Realm_c \| ID_c \| Ticket_v \| E(K_{c,tgs},[K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_v])$

$$Ticket_{tgs} = E(K_{tgs},[Flags \| K_{c,tgs} \| Realm_c \| ID_c \| AD_c \| Times])$$

$$Ticket_v = E(K_v,[Flags \| K_{c,v} \| Realm_c \| ID_c \| AD_c \| Times])$$

$$Authenticator_c = E(K_{c,tgs} [ID_c \| Realm_c \| TS_1])$$

- Message (3) is similar to both versions. It includes an authenticator, a ticket and the name of the requested service.
  - Version 5 includes requested time, options for ticket and a nonce
- Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.

(c) Client/Server Authentication Exchange to obtain service

(5) C → V Options || $Ticket_v$ || $Authenticator_c$

(6) V → C $E_{Kc,v}$, [TS_2 || Subkey || Seq#]

$Ticket_v = E(K_v,[Flags \| K_{c,v} \| Realm_c \| ID_c \| AD_c \| Times])$

$Authenticator_c = E(K_{c,v}, [ID_c \| Realm_c \| TS_2 \| Subkey \| Seq\#])$

- Message(5) may include a request as an option for mutual authentication
  - Subkey: The clients choice for an encryption key to be used to protect this specific application session, if omitted, the session key is used
  - Sequence number : An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. It may be included in message to detect replay
- Message (6) is sent by server when mutual authentication is required. It includes the timestamp from the authenticator.

The flags field was added in Kerberos Version 5 for additional functionality.

- INITIAL: This flag indicates that a ticket was issued using the AS protocol and not issued based on a ticket-granting ticket
- INVALID: This flag indicates that a ticket is invalid, which means that application servers must reject tickets which have this flag set.
- RENEWABLE: This flag is normally only interpreted by the ticket-granting service, not by application servers, and can be used to obtain a replacement ticket that expires at a later date.
- POSTDATED: The POSTDATED flag indicates that a ticket has been postdated.
- PROXIABLE: normally interpreted by the ticket-granting service and ignored by application servers.
  - When set, this flag tells the ticket-granting server that it is OK to issue a new (proxy) 'client' ticket with a different network address based on this ticket.
- PROXY: This flag is set in a ticket by the TGS when it issues a proxy ticket.
- FORWARDABLE: This flag has an interpretation similar to that of the PROXIABLE flag, except ticket-granting tickets may also be issued with different network addresses (to be used with remote TGS)

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

Limitations of Kerberos Scheme

- Every network service must be individually modified for use with Kerberos
- Doesn’t work well in time sharing environment
- Requires a secure Kerberos Server
- Requires a continuously available Kerberos Server
- Stores all passwords encrypted with a single key
- Assumes workstations are secure
- Inter-realm authentication is allowed by forwarding, but no way to derive the complete "chain of trust", nor any way to do "authentication routing" within the hierarchy of authentication servers