Authentication Applications

Outline

- Kerberos: uses conventional encryption
  - V4
  - V5
  - Comparison

Common Configuration of Applications

- Dedicated PC with no network connections.
  - Resources/files can be protected by physical security
  - Threats come from users who have physical access to it.
- Centralized time-sharing system over a network.
  - The time sharing OS provides security of applications.
  - Access control policies can be enforced depending on user ID.
- Distributed/centralized clients and servers over a network.
  - Security is a mess!!
  - Need better system of authentication of user and application

Approaches to Security in a Distributed Environment

- Rely on the individual client workstation to assure the identity of its users and then have the servers enforce some security policy based on the user ID.
  - ✓This is good for small closed environments.
- Require the client system to authenticate themselves to the server and leave the user identity authentication to the client.
  - ➢ Here all users attached to the client will have access to the server after authentication. This is also good for small closed environments.
- Require the user to provide identity for each service offered by the server. The servers can also identify themselves.
  - ➢This is good for distributed system security. (supported by Kerberos)

Kerberos

- Developed as a part of Project Athena at MIT.
- Kerberos addresses the authentication for services offered by servers in a distributed environment that are accessed using workstations.
- The basis is that in such an environment the workstations cannot be trusted to identify and authenticate users for network services on the servers.
- Three threats exist in such an environment.
  - Gain access to the workstation and then pretend to be another user.
  - User alter the network address of a workstation. This can be used to impersonate a workstation.
  - User eavesdrop on exchanges and use a replay attack.
- Provides a centralized authentication server to authenticate users to servers vise versa.
- Relies on conventional encryption, making no use of public-key encryption

Kerberos

- Two versions: version 4 and 5
- Version 4 makes use of DES
- Requirements for Kerberos
  - **Secure:** an opponent should not be able to impersonate another user.
  - **Reliable:** The Kerberos should be based on a distributed server architecture to make it available always. When one server fails the other should be able to authenticate to maintain the service.
  - **Transparent:** The user should not be aware that authentication is taking place beyond entering his/her password.
  - **Scalable:** The system should be capable of supporting a large number of clients as well as servers.
- The Kerberos system, in short, acts like a third party authentication system.

Preview to Kerberos V4

Terminology

- 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
- ADc = network address of C
- K_v = secret encryption key shared by AS and V
- TS = timestamp
- || = concatenation

Justification for the use of Authentication Servers

- **Scenario:** In an unprotected network environment any client can apply for any service in any server.
- **Risk:**
  - ✓ Major risk of impersonation.
  - ✓ The opponent can pretend to be another client and gain access to a server.
- **Solution:** The servers have to undertake the client authentication.
- **Downside:** This puts an enormous burden on the server.
- **Better solution:**
  - ✓An AS can solve this problem.
  - ✓The AS will share a unique secret key with each server and also it knows all the passwords coming from each client.

Kerberos V4
A Simple Authentication Dialogue

(1) C → AS: $ID_c \,||\, P_c \,||\, ID_v$

(2) AS → C: Ticket

(3) C → V: $ID_c \,||$ Ticket

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

•ADc is important since now no one can capture the ticket and transmit with IDc from a different workstation. Now the ticket is only valid from this network address. If ADc is not in the ticket any workstation can be authenticated if impersonated.

Highlighted Issues

- **Problems:**
  - Lifetime associated with the ticket. E.g. For mail. For a single logon to the client you have to type the password to access the mail service many times. Solution is to have reusable tickets.
  - Then each service will require a ticket and need to send password for each service at initiation. i.e. after login you still have to send password for each new service.
  - If lifetime too short → repeatedly asked for password.
  - If lifetime too long → greater opportunity to replay.
- The threat is that an opponent will steal the ticket and use it before it expires.
- Another problem is the password is sent using plaintext which can be grabbed and used to gain access to any service.

Authentication Dialogue with more security

- We need to fix two problems of the previous implementation
  - Sending of plaintext password
  - New server logon for each service.
- Both problems are fixed using a ticket-granting server (TGS)
- The process requires three stages.
  - Login authentication, once per logon session
  - Service authentication, once per type of service
  - Session authentication, once per service session.
- The three stages have five steps to complete before the authentication is complete.

Stage 1: Logon authentication

- **Step 1:** C→ AS : ID_c | | ID_{tgs}
  - The client will request a ticket-granting ticket on behalf of the user by sending the user ID which is associated with the username at logon.
- **Step 2:** AS→C : $E_{Kc}[Ticket_{tgs}]$
  - The ticket is sent after encrypting it with a key that is based on the users password which is known to the AS.
  - Then client C will prompt for the password from the user.
  - Then the client will generate the key using the given password and decrypt the ticket.
  - The correct password is only known by the proper user.
  - The text password has not been transmitted at any time.
  - The ticket is encrypted using the key of the TGS so that the client cannot change it.

Stage 1: Logon authentication Problems

- Note that the ticket thus generated can be used by the client to request for multiple service granting tickets. Hence reusable.
- Consider a case when an opponent gets access to the ticket in Step 2.
- Waits till the actual user logs off and then either gains access to the workstation fraudulently or configures his/her workstation with the same network address and then uses the ticket to spoof the TGS.
- The time stamp and the lifetime in the ticket will circumvent this to a certain degree.
- Note also that the ticket can only be recovered by the intended user since the encryption key is based on the password. Either the opponent should get the ticket once it is decrypted or when it is sent to the TGS.

Stage 2: Service Authentication

- **Step 3:** C→ TGS : ID_c || ID_v || $Ticket_{tgs}$
  - $Ticket_{tgs}=E_{Ktgs}[ID_c||AD_c||ID_{tgs}||TS_1||Lifetime_1]$
  - The client now requests the service-granting ticket on the users behalf. IDv is the ID of the required service.
- **Step 4:** TGS→ C : $Ticket_v$
  - $Ticket_v=E_{Kv}[ID_c||AD_c||ID_v||TS_2||Lifetime_2]$
  - The TGS will verify the success of the decryption by the presence of its ID. Note: any key can be used to decrypt but the output is usable only if the correct key is used.
  - Checks to see if the lifetime has not expired.
  - Compares the user ID and the network ID with the incoming information to authenticate the user.
  - If the user is permitted to access V, the TGS will issue a ticket to grant access to the requesting service.

Stage 2: Service Authentication

- Note that the service-granting ticket has the same structure as the ticket granting ticket.
- This is because the same elements would be required by this server to authenticate too.
- If the user wants to access the same service within the same logon session, the client can simply use the previously acquired service-granting ticket and need not bother the user for the password again.
- Note that the ticket v is encrypted using K_v, the secret key known only to the TGS and V which will prevent C or any other opponent changing this.

Stage 3: Session Authentication

- **Step 5:** C→ V: $ID_c||Ticket_v$
  - The client will request access from the server V on behalf of the user.
  - The users ID is also transmitted with the ticket. So that the user can be authenticated once the ticket is decrypted.

Issues with Given Scenario

- Lifetime associated with tickets
  - If the lifetime of the ticket-granting ticket is short the user may have to enter the password repeatedly.
  - If the lifetime of the ticket-granting ticket is too long the greater the opportunity for replay (assume the identity of the server once the legitimate server shuts down and use the ticket to gain access to the TGS: all the legitimate resource available to the original user)
  - The same with the service-granting ticket. In this case all the services accessible to the original user is now available to the opponent.

Issues with Given Scenario

- Authentication of the server to the user
  - If the server does not authenticate to the user, the opponent can take over the identity of the client and configure the server to send the messages due to the legitimate user permanently to a different location.
  - The false server will act as the real server and capture any information from the user and then deny the true service to the user.

Version 4 Authentication Dialogue

Authentication Service Exhange: To obtain Ticket-Granting Ticket

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

(2) AS → C: $E_{Kc}[K_{c,tgs} || ID_{tgs} || TS_2 || Lifetime_2 || Ticket_{tgs}]$

$Ticket_{tgs} = E_{Ktgs}[K_{c,tgs} II\ ID_c || AD_c || ID_{tgs} || TS_2 || Lifetime_2]$

Ticket-Granting Service Echange: To obtain Service-Granting Ticket

(3) C → TGS: IDv $||Ticket_{tgs}||$ Authenticatorc

(4) TGS → C: $E_{Kc}[K_{c,v}||$ IDv $||$ TS_4 $||$ Ticketv]

Client/Server Authentication Exhange: To Obtain Service

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

(6) V → C: EKc,v[TS_5 +1]

Design Highlights (1-2)

- **Threat:** Capture of the ticket-granting ticket and use it before it expires.
- **To Solve:** Need to ensure that the ticket presenter is the same client to whom the ticket was issued.
- **Implementation:** The AS will provide both the client and the TGS with a secret piece of information securely in the form of a shared key $K_{c,tgs.}$ The client can then prove its identity to the TGS by revealing the secret information, again in a secure manner.
- This key is also called the Kerberos session key.
- The session key is deliver so that one can be read by C and the other can only be read by the TGS.
- Hence we see that the session key has been securely delivered to both C and TGS.

Additions to Stage 1

- The message in step (1) now includes a time stamp to indicate to the AS that the message is timely.
- The message in step (2) includes several elements in a form that they are accessible by C. These information will enable C to confirm that this ticket is for TGS and to learn its expiration time.

Design Highlights (3-4)

- The authenticator is not reusable and has a short lifetime and is meant to be used only once compared to the ticket which is reusable.
- When the TGS opens the ticket and finds the session key it will assume that anyone who uses $K_{c,tgs}$ must be C. Since C is the only other party to whom this key has been issued.
- Also the TGS should use the session key to decrypt the authenticator tag. All of the information within the authenticator and being able to decrypt using the session key will indicate that this has come from C.
- Note that the ticket $ticket_{tgs}$ does not prove anyone's identity. It’s the authenticator which does it.
- The authenticator has a short life. So we are assuming that an opponent cannot capture both the ticket and the authenticator and use it for an attack.

Design Highlights (4-6)

- The reply from the TGS in step 4 will send the session key that has to be used between C and V.
- The ticket sent by the TGS in 4 also contains the same session key but the ticket is encrypted using the key shared by TGS and V so C cannot see it.
- In step 5 the ticket is sent with an authenticator. Once the ticket is decrypted, the session key can be used to decrypt the authenticator and then use the information within tie authenticate.
- If mutual authentication is required, the time stamp in the authenticator is returned incremented by one.
- When it is received by C, as the message is encrypted using the session key it can authenticate V (only other party having the key) and also the contents will ensue that this is not a replay of an old reply.

Rationale for Message Elements

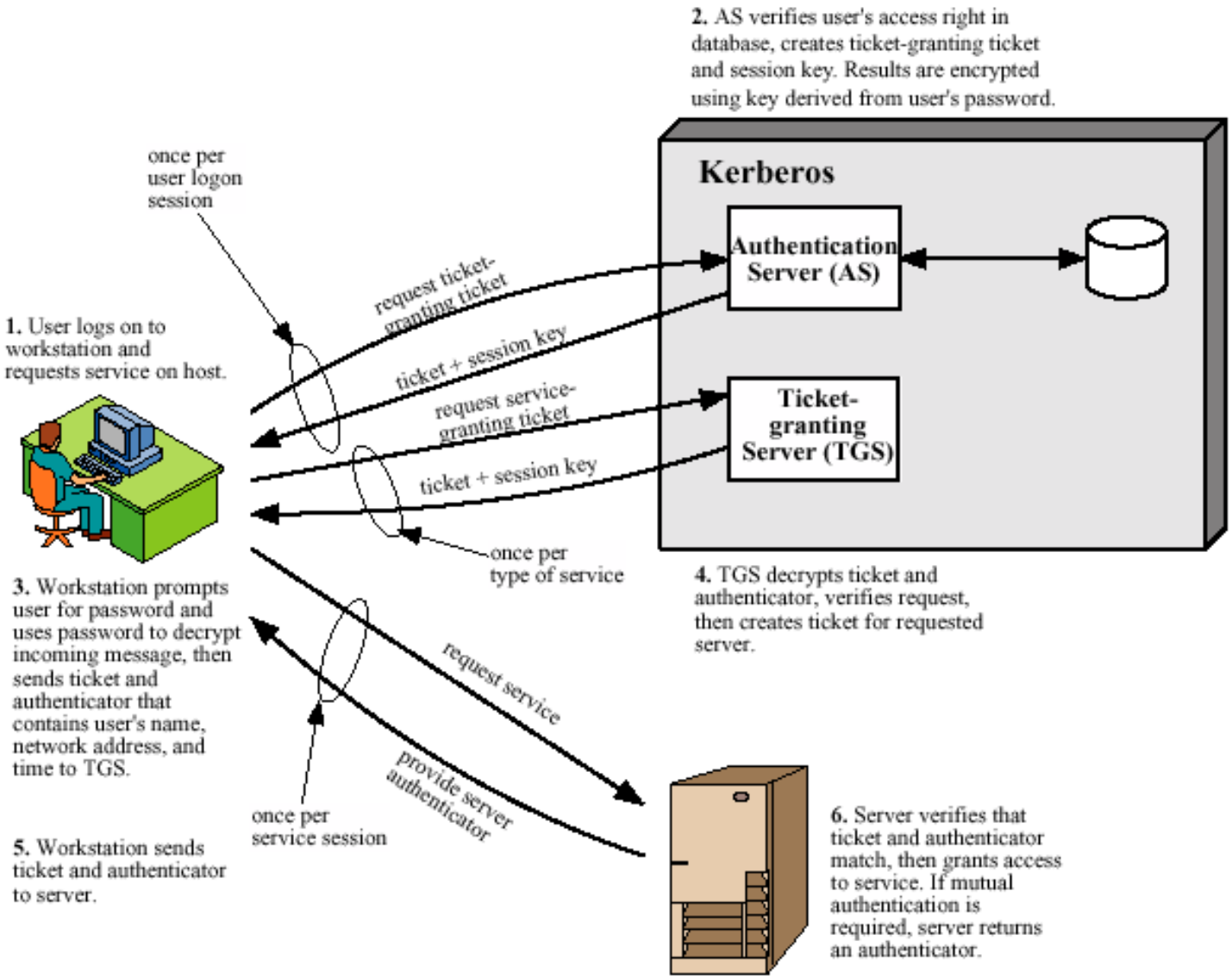
(a) Authentication Service Exchange	
Message (1)	Client requests ticket-granting ticket
ID_C:	Tells AS identity of user from this client
IDtgs:	Tells AS that user requests access to TGS
TS_1:	Allows AS to verify that client's clock is synchronized with that of AS
Message (2)	AS returns ticket-granting ticket
E_{K_c}:	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2)
$K_{c,tgs}$:	Copy of session key accessible to client; created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key
ID_{tgs}:	Confirms that this ticket is for the TGS
TS_2:	Informs client of time this ticket was issued
$Lifetime_2$:	Informs client of the lifetime of this ticket
$Ticket_{tgs}$:	Ticket to be used by client to access TGS

(b) Ticket-Granting Service Exchange	
Message (3)	Client requests service-granting ticket
ID_V:	Tells TGS that user requests access to server V
$Ticket_{tgs}$:	Assures TGS that this user has been authenticated by AS
$Authenticator_c$:	Generated by client to validate ticket
Message (4)	TGS returns service-granting ticket
$K_{c,tgs}$:	Key shared only by C and TGS; protects contents of message (4)
$K_{c,v}$:	Copy of session key accessible to client; created by TGS to permit secure exchange between client and server without requiring them to share a permanent key
ID_V:	Confirms that this ticket is for server V
TS_4:	Informs client of time this ticket was issued
$Ticket_V$:	Ticket to be used by client to access server V
$Ticket_{tgs}$:	Reusable so that user does not have to reenter password
$E_{K_{tgs}}$	Ticket is encrypted with key known only to AS and TGS, to prevent tampering
$K_{c,tgs}$:	Copy of session key accessible to TGS; used to decrypt authenticator, thereby authenticating ticket
ID_C:	Indicates the rightful owner of this ticket
AD_C:	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_{tgs}:	Assures server that it has decrypted ticket properly
TS_2:	Informs TGS of time this ticket was issued
$Lifetime_2$:	Prevents replay after ticket has expired

$Authenticator_c$:	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$E_{K_{c,tgs}}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering
ID_C:	Must match ID in ticket to authenticate ticket
AD_C:	Must match address in ticket to authenticate ticket
TS_2:	Informs TGS of time this authenticator was generated

(c) Client/Server Authentication Exchange	
Message (5)	Client requests service
$Ticket_V$:	Assures server that this user has been authenticated by AS
$Authenticator_C$:	Generated by client to validate ticket
Message (6)	Optional authentication of server to client
$E_{K_{c,v}}$:	Assures C that this message is from V
$TS_5 + 1$:	Assures C that this is not a replay of an old reply
$Ticket_v$:	Reusable so that client does not need to request a new ticket from TGS for each access to the same server
E_{K_v}:	Ticket is encrypted with key known only to TGS and server, to prevent tampering
$K_{c,v}$:	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket
ID_C:	Indicates the rightful owner of this ticket
AD_C:	Prevents use of ticket from workstation other than one that initially requested the ticket
ID_V:	Assures server that it has decrypted ticket properly
TS_4:	Informs server of time this ticket was issued
$Lifetime_4$:	Prevents replay after ticket has expired
$Authenticator_C$:	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay
$E_{K_{c,v}}$:	Authenticator is encrypted with key known only to client and server, to prevent tampering
ID_C:	Must match ID in ticket to authenticate ticket
AD_C:	Must match address in ticket to authenticate ticket
TS_5:	Informs server of time this authenticator was generated

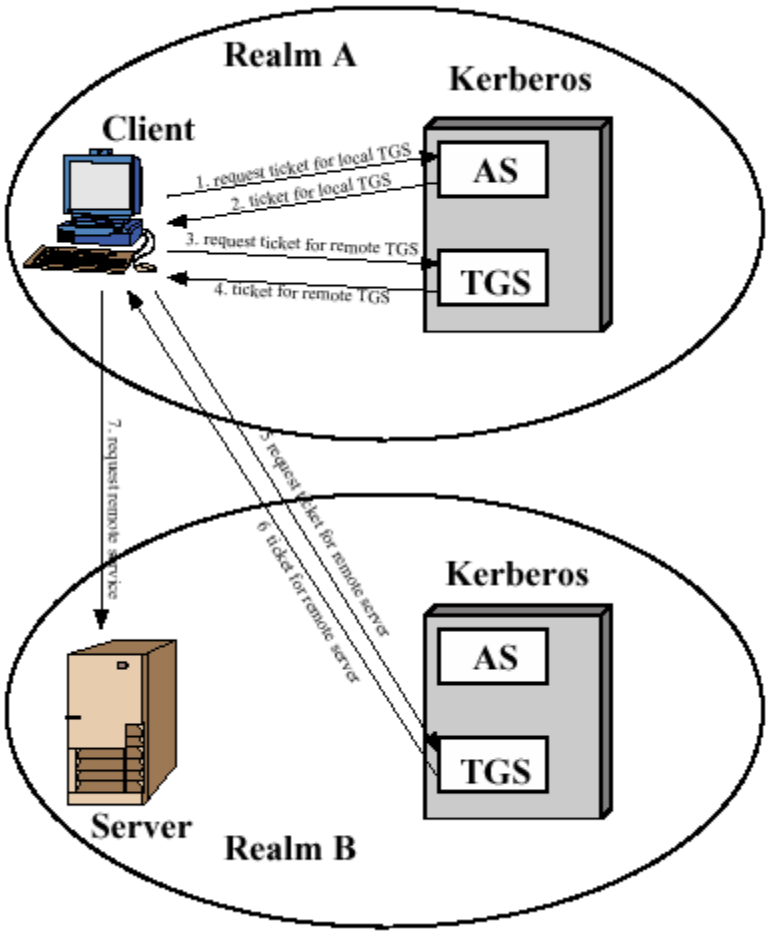
A Full Service Kerberos Environment



Kerberos Environment Requirements

- The Kerberos server should have all the UIDs and hashed passwords of all participating users in its database. Hence registering of all users is a requirement.
- The Kerberos server should share a secret key with each server hence all servers are required to register with the Kerberos server.
- Networks of clients and servers under different administrative organizations typically constitute a Kerberos realm. Inter-realm communication should be possible.
- The Kerberos server in each realm shares a secret key with the server in the other realm. That is the two Kerberos servers are registered with each other.

Requesting a Service in another Realm



- To use a service in a server located in another realm it needs a ticket.
- Users client will gain access to the TGS in the usual manner and request a ticket-granting ticket for a remote TGS.
- 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

Message Exchange for Remote Realm Access

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

The only problem with this exchange is that when there are many Realms.

There are too many key exchanges to different realms.

Differences Between V4 and V5

- Environmental Shortcomings
  - **Encryption system dependence:** V4 uses DES (credibility in doubt). In V5 the ciphertext is tagged with an encryption type identifier hence any type of encryption can be used. The key is also tagged with a type and length hence the key is reusable in different algorithms.
  - **Internet protocol dependence:** V4 uses IP addressing. V5 can use any address since the address is now tagged with type and length.
  - **Ticket lifetime:** In V4 the lifetime has to be specified in units of 5 mins (2^8*5=1280min=21hrs max). In V5 one can specify an explicit start and finish times allowing arbitrary lifetimes.

Differences Between V4 and V5 …

- **Inter-realm authentication:** V4 requires many kerberos-to-kerberos relationships. V5 supports a fewer relationships.

- Technical deficiencies
  - **Double encryption:** Message 2 and 4 require encryption. Maybe one can be eliminated. The ticket is already encrypted so no use of encrypting it again.
  - **PCBC encryption:** Propagating CBC is used in V4 and this is non-standard. In V5 CBC is used.
  - **Session key:** Each ticket includes a session key that can be used to encrypt the authenticator. Since the same key is used repeatedly to gain a service from particular server, there is a risk that an opponent can replay messages from an old session to the client or server. In V5 this is avoided by requiring a sub session key which is used only for one connection.

V5 Authentication Dialogue

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

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

(2) AS → C: $Realm_c \| ID_C \| Ticket_{tgs} \| E_{K_c}\left[K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}\right]$

$$Ticket_{tgs} = E_{K_{tgs}}\left[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times\right]$$

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

(3) C → TGS: Options || ID_v || $Times$ || || $Nonce_2$ || $Ticket_{tgs}$ || $Authenticator_c$

(4) TGS → C: $Realm_c \| ID_C \| Ticket_v \| E_{K_{c,tgs}}\left[K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_V\right]$

$$Ticket_{tgs} = E_{K_{tgs}}\left[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times\right]$$

$$Ticket_v = E_{K_v}\left[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times\right]$$

$$Authenticator_c = E_{K_{c,tgs}}\left[ID_C \| Realm_c \| TS_1\right]$$

(c) Client/Server Authentication Exchange: to obtain service

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

(6) V → C: $E_{K_{c,V}}$ [TS_2 || Subkey || Seq#]

$$Ticket_v = E_{K_v}\left[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times\right]$$

$$Authenticator_c = E_{K_{c,V}}\left[ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#\right]$$

V5 Improvements

- The service exchange has several new elements such as Realm, Options, Times (to indicate start stop and update times for the ticket) and Nonce (a random value based on the instance to indicate that the messages have not been replayed).
- In the ticket granting service exchange additions similar to above have been incorporated.
- Note that now the ticket that is sent in message 2 and 4 are not twice encrypted.
- For the client/server authentication several new fields such as sub key (this is the clients choice of the encryption key to be used for protecting the application session. If this field is omitted then $K_{c,v}$ is used) and sequence (sequence numbers are used to detect replays) number are added.

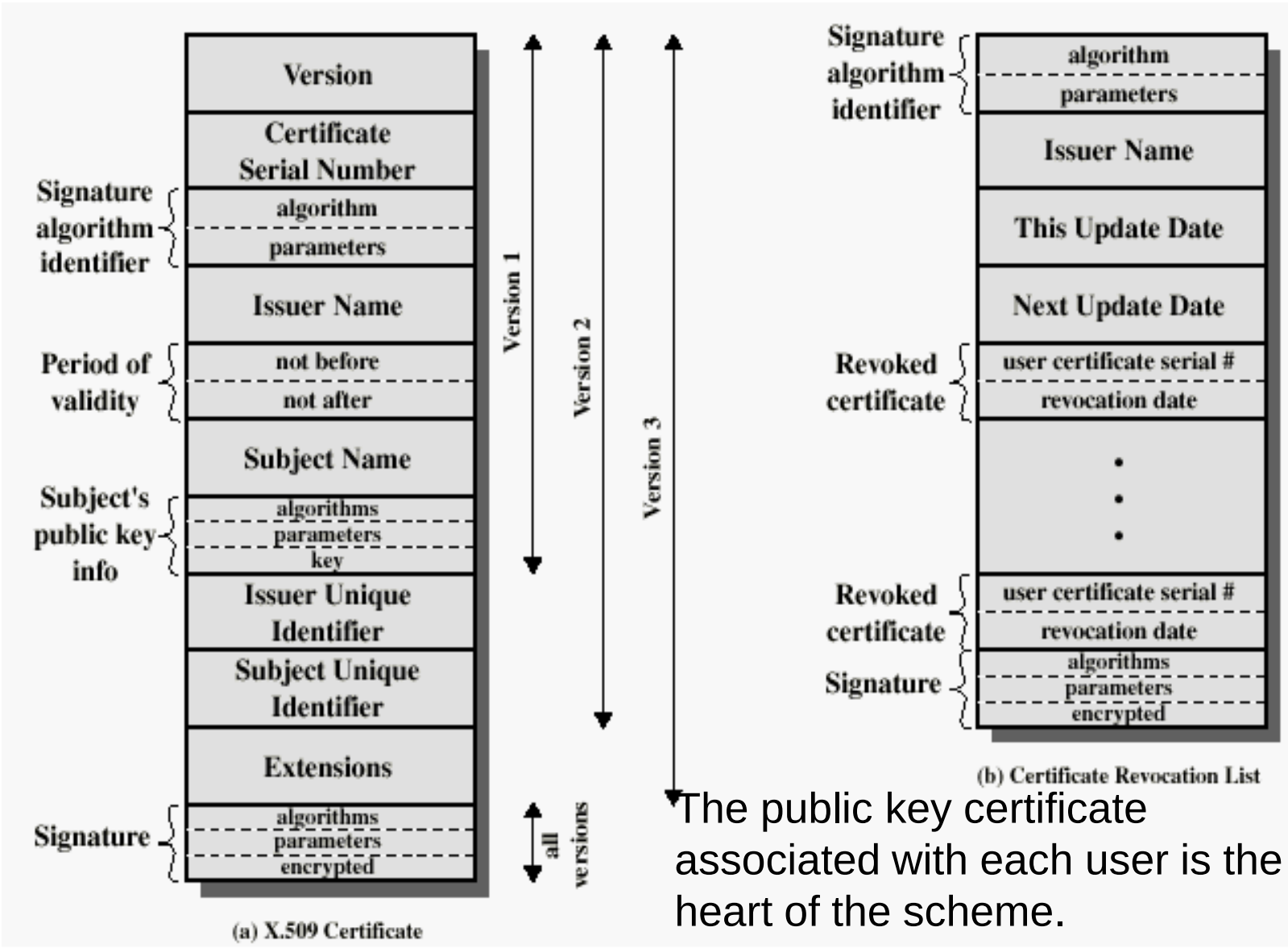
Kerberos - in practice

- Currently, we have two Kerberos versions:
  - 4 : Restricted to a single realm being phased out
  - 5 : Allows inter-realm authentication, rolled out.
- Kerberos v5 is an Internet standard.
- To use Kerberos:
  - Need to have a KDC on your network .
  - Need to have Kerberised applications running on all participating systems
- Major problem - US export restrictions
- Kerberos cannot be directly distributed outside the US in source format (& binary versions must obscure crypto routine entry points and have no encryption).
- Else crypto libraries must be re-implemented locally.

X.509 Authentication Service

- More broadly X.509 defines a framework to provide a secure directory services.
- Each directory is a server or a distributed set of servers that maintains a database about users.
- The directory may server as a repository of public key certificates. Each certificate contains the public key of a user and is signed with the private key of a CA (certification authority).
- The X.509 certification structure and authentication protocols are used in S/MIME, IP Security, SSL/TLS and SET.
- X.509 is based on the use of public-key cryptography and digital signatures. RSA algorithm is recommended to be used in this algorithm.

X.509 Formats



The public key certificate associated with each user is the heart of the scheme.

Obtaining a User's Certificate

- Characteristics of certificates generated by CA:
  - Any user with access to the public key of the CA can recover the user public key that was certified.
  - No part other than the CA can modify the certificate without this being detected. (it uses a hash encrypted with a private key)
- Because the certificates are unforgeable, they can be placed in a directory without the need for the directory to make special effort to protect them.
- If all users subscribe to the same CA, then there is a common trust.
- When the CA signs the certificate, each participating use must have a copy of the CAs public key to verify signatures.
- The public key has to be provided in a very secure manner so that the confidence in it will be maintained.

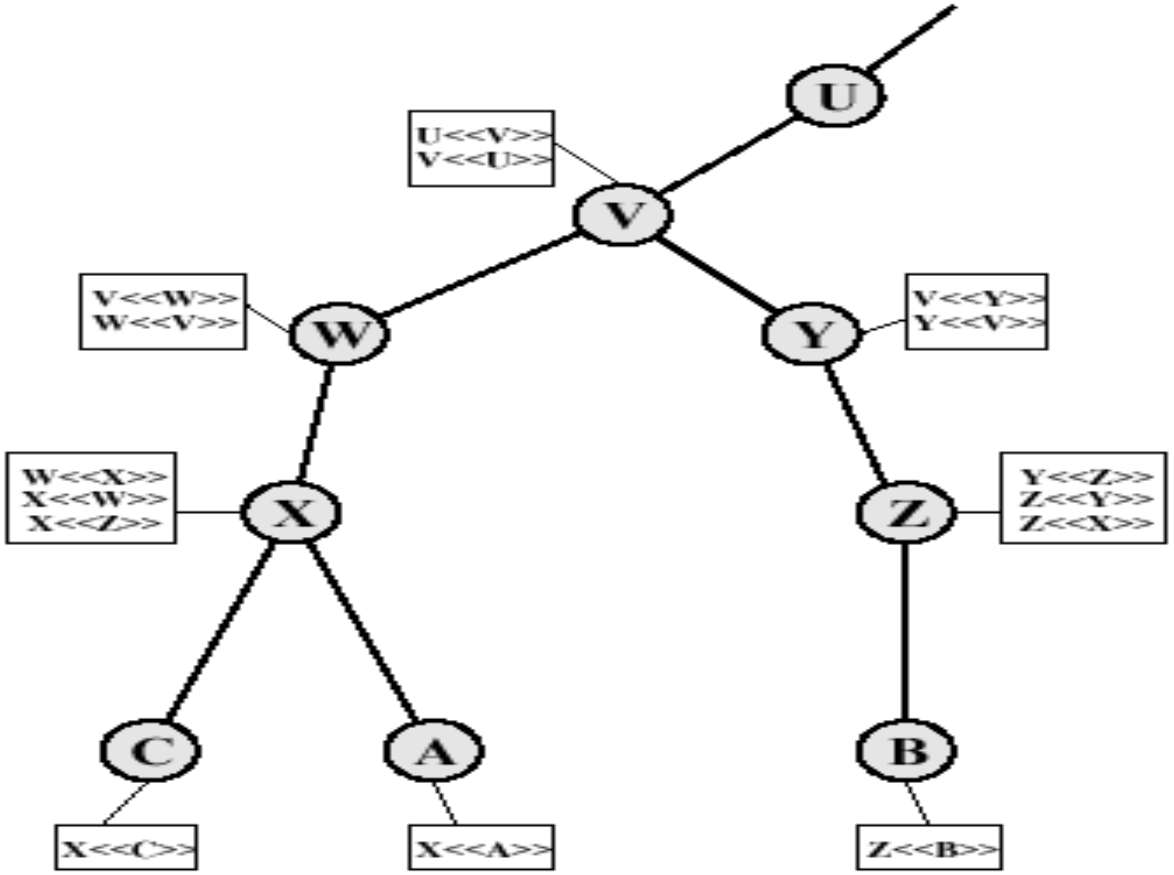
Example Scenario (1)

- A has obtained a certificate from CA X1 (X1<<A>>)
- For B is it X2<<B>>
- If A does not securely know K^+_B then Bs certificate issued by X2 is useless to A. i.e. A can read Bs certificate X2<<B>> but cannot verify the signature.
- If the two CAs have securely exchanged their public keys, the following procedure can be used to obtain K^+_B
  - A obtains from the directory, the certificate of X2 signed by X1. Because A securely knows K^+_{x1}, A can obtain K^+_{x2} from its certificate and verify by means of X1's signature on the certificate.
  - A then goes back to the directory and gets X2<<B>>. Because now A has a trusted copy of K^+_{x2}, A can verify the signature and securely obtain K^+_B

Example Scenario (2)

- The above chain of processes can be represented by X1<<X2>>X2<<B>>
- For B the process would be X2<<X1>>X1<<A>>
- All of the certificates required should be in the directory. The directory structure is hierarchically organized.
- The boxes in the figure given in the next page are the certificates maintained in the directory. These can be two types
  - Forward certificates: certificates of A generated by other CAs
  - Backward certificates: certificates generated by X that are the certificates of other CAs.

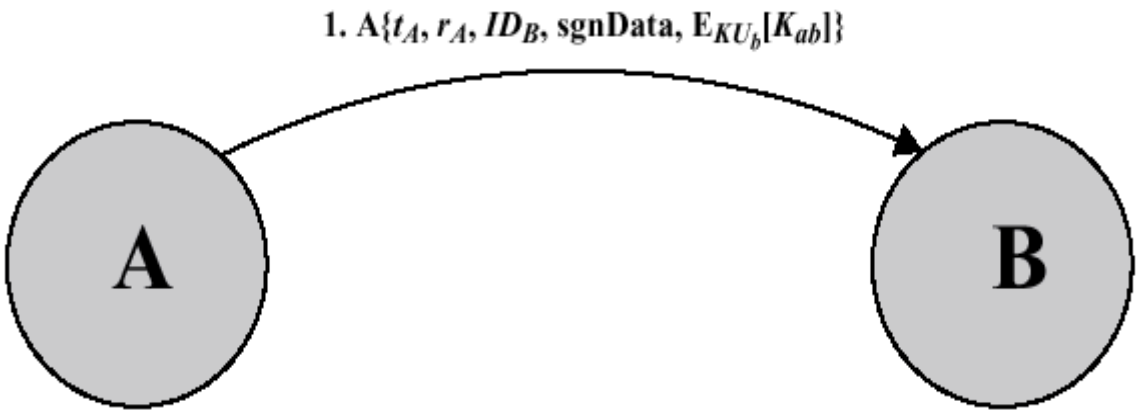
X.509 CA Hierarchy



Revocation of Certificates

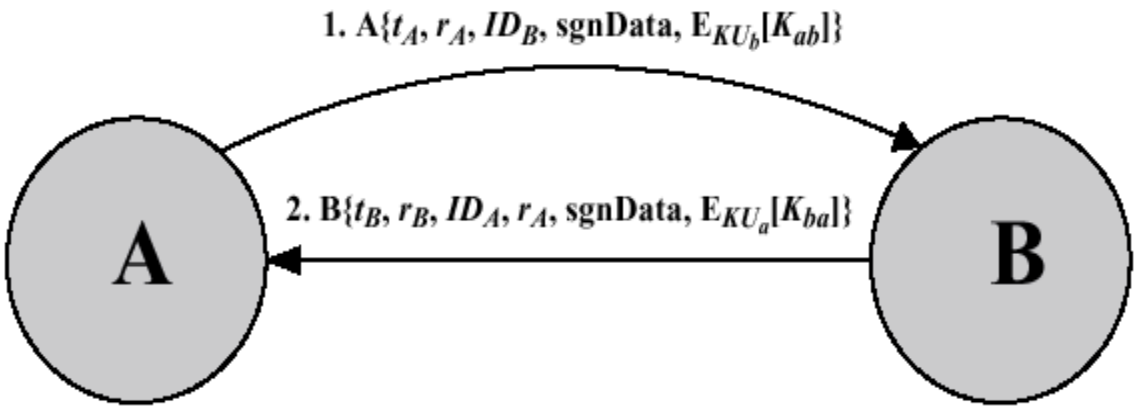
- The certificates has a period of validity that expires just before it is renewed. Revocation means canceling the certificate before it expires.
- Reasons for revocation:
  - The users secret key is assumed to be compromised.
  - The user is no longer certified by this CA.
  - The CA's certificate is assumed to be compromised.
- After revocation the certificate revocation list (CRL) in the directory is updated.
- The user should check the CRL before using any certificate.

Authentication Procedures
One way authentication



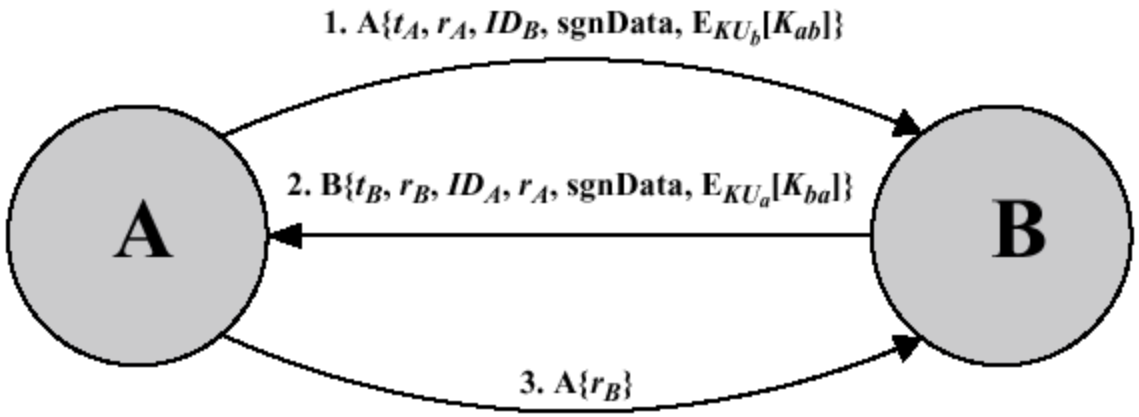
- The communication establishes the following
  - The identity of A and that the message was generated by A
  - That the message was intended for B
  - The integrity and the originality of the message
    - Delayed attacks stopped by the time stamp
    - Nonce for replay attacks

Authentication Procedures
Two-way authentication



- The communication establishes the following
  - All of the previous facts established
  - The identity of B and that the reply message was generated by B
  - That the message was intended for A
  - The integrity and originality of the reply.

Authentication Procedures
Three-way authentication



- A final message from A to B containing a signed copy of the nonce rB is sent
- The intent of this is that now the time stamps need not be checked.
- Since both nonce's are echoed back by the other side, each side can detect replay attacks
- This approach is suitable when synchronized clocks are not available.

What are the ADV/DISADV of Kerberos Over SSL?

- In brief, the question seems to be, "What does Kerberos give me that SSL doesn't?"
- Or: "What are the advantages and disadvantages of a private-key, trusted-third-party authentication system vs. a public-key, certificate-based authentication system?"
- SSL has two major advantages over Kerberos:
  - It doesn't require an accessible trusted third party;
  - it can be used to establish a secure connection even when one end of the connection doesn't have a "secret" (a.k.a. "key" or "password").
- These two advantages make it ideal for secured Web communication and for similar applications where there is a large user base which is not known in advance.

Some DISADV of SSL

- Complexity of key revocation
- Security of the key
- SSL service is not really free (eg. Verisign)
- Open standards observed by kerberos is not available in SSL since it is a commercial product.
- Flexibility of operation is easier in kerberos and harder in SSL.

Key Revocation

- If a Verisign certificate issued to a user is compromised the revocaton should be announced to all servers with which communication was sought.
- revocation certificates have to be circulated to all relevant servers and cached for a long time this is not possible.
- Or revocation servers that are accessible to third parties have to be used. Again not safe.
- Kerberos principals can be disabled on the KDC and will then become unusable as soon as any cached tickets expire without any action by servers. Very quick operation.

Key security

- If I'm issued a Verisign certificate, it has to live on my hard disk.
- Yes, it may be encrypted there such that I have to unlock it with a password before I can use it, but it's still on the hard disk.
- Long time residence is vulnerable to cracking attacks.
- On the other hand, any sort of certificate is not required to authenticate to Kerberos -- all I need is my password, which is in my brain, not on a hard disk.

Cost Aspect

- Kerberos doesn't infringe on any patents.
- That it can be used for free, while SSL users may have to pay.
- SSL is a commercial product and hence packaged for money
- Kerberos is maintained free by MIT.

Open Standards

- Kerberos has been free from the beginning.
- The standards documenting it are open and have been developed openly from the start.
- SSL was developed by a company with a commercial interest in ensuring that its standards become THE standard.

Flexibility of Operation

- Kerberos is somewhat more flexible than SSL.
- For example, if I want to add a new authentication technology to Kerberos (e.g., a new kind of SmartCard with its own algorithm), all that has to be done is to modify my KDC and my ticket-acquiring client to know how to do the new authentication.
- Then, it can be used to get Kerberos tickets which will look the same as any other Kerberos tickets and will be usable with any Kerberos-capable application.
- For a new authentication technology for SSL, one has to to get anew versions of all my SSL-capable applications.