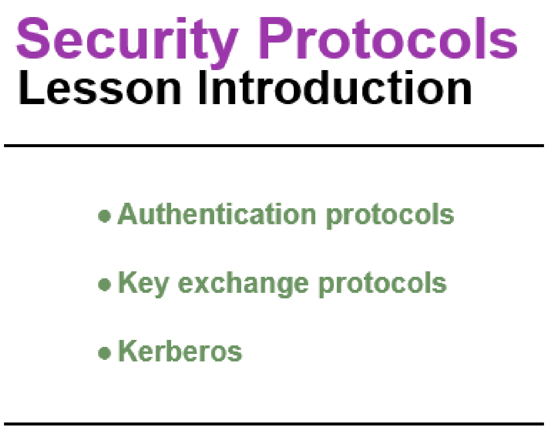
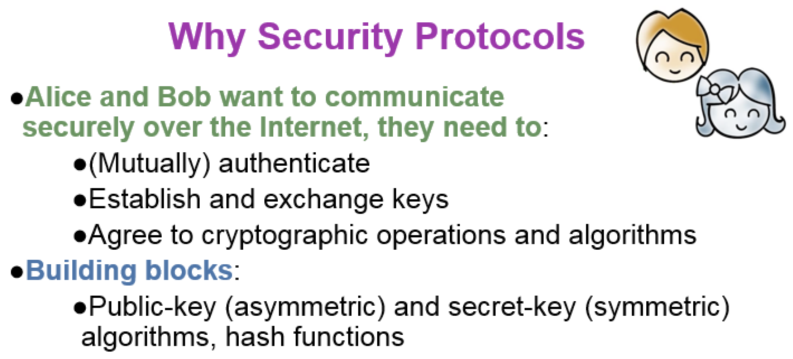
Reference: Computer Security by Stallings and Brown, Chapter 23



Security Protocols
Lesson Introduction

- Authentication protocols
- Key exchange protocols
- Kerberos

Security protocols are the foundation of secure network applications. In this lesson, we will discuss authentication protocols, key change protocols, and the Kerberos systems.



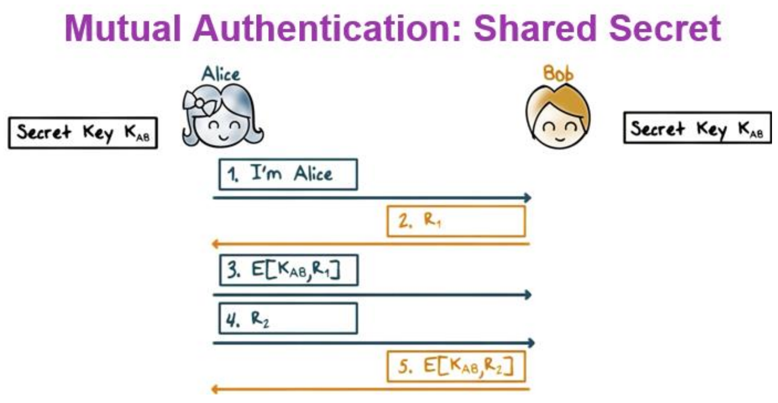
Why Security Protocols

- Alice and Bob want to communicate securely over the Internet, they need to:
  - (Mutually) authenticate
  - Establish and exchange keys
  - Agree to cryptographic operations and algorithms
- Building blocks:
  - Public-key (asymmetric) and secret-key (symmetric) algorithms, hash functions

A network protocol defines the rules and conventions for communications between two parties.

A security protocol defines the rules and conventions for secure work communications between two parties.

So what are these rules and conventions for secure communications? That is, suppose Alice and Bob want to communicate securely over the internet. What should they do? They need to authenticate each other so that Alice knows that she's communicating with Bob and Bob knows that he's communicating with Alice. Further, if they want to communicate securely, most likely they want to encrypt their messages. Therefore, they need to establish and exchange keys, and agree on what cryptographic operations and algorithms to use. The basic tools, or the building blocks of these security protocols, are the public-key and secret-key algorithms and hash functions that we have discussed.



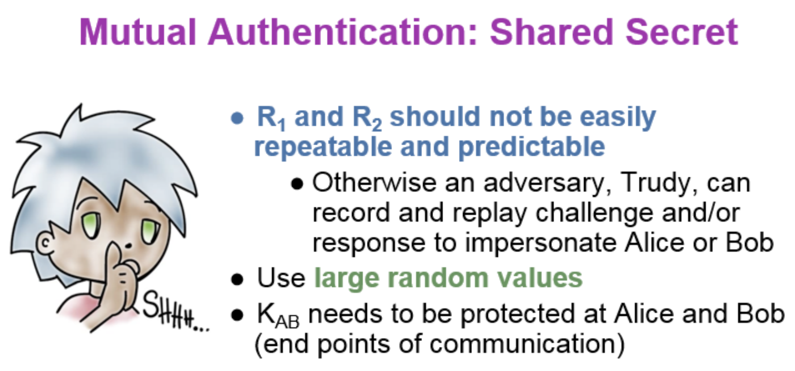
Let's take a look at authentication. That is, Alice needs to prove to Bob that she's really Alice, and vice versa. Meaning that Bob needs to prove to Alice that he is Bob.

Suppose that Alice and Bob share a secret key, KAB. That is, only Bob and Alice know this key, KAB. Here's an authentication protocol using symmetricartography:

1. First Alice says to Bob, hey I'm Alice.
2. Second, Bob said to Alice, really? Prove it. And he sends a random value R1, which we will call a challenge.
3. Third, Alice then encrypts R1 using the shared key KAB. And sends Bob the cybertext, which we'll call it the response to the challenge. When Bob receives the response, he decrypts it and see if it matches the plain text, R1 that he just sent to Alice. If it matches, then he knows that the party that he's communicating with must be Alice, because R then, himself, only Alice knows the shared key KAB. And without KAB, R1 cannot be encrypted properly. That is the server text that he receives won't be decrypted properly to R1.
4. Fourth, now it is Alice's turn to authenticate Bob. So, similarly, she sends Bob a random challenge, R2.
5. Fifth, Bob encrypts R2 with the shared key KAB and sends a cyber text to Alice.

Upon receiving the response from Bob, Alice decrypts the cyber text and if the result matches the plain text R2 that she just sent to Bob, then she knows that the party that she's communicating with must be Bob.

Note that if only one-way authentication is required, for example, Alice is a client and needs to authenticate to Bob, which is a server, but Bob does not need to authenticate to Alice, then only the first three steps are necessary. And if Alice is a human user, then typically the key KAB can be derived from a password hash, which is known to Bob.

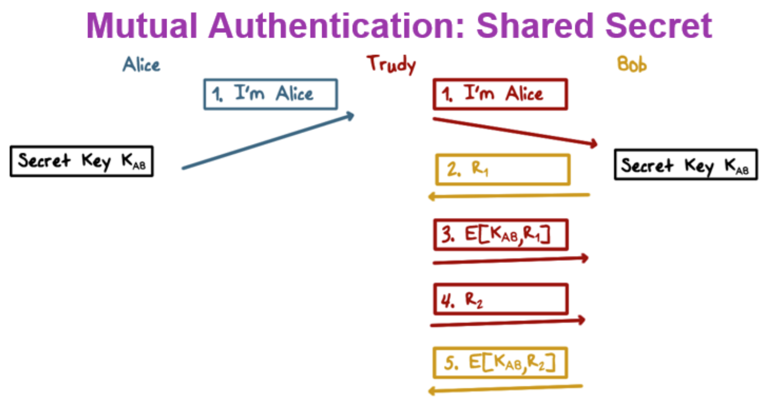


Mutual Authentication: Shared Secret

- R_1 and R_2 should not be easily repeatable and predictable
  - Otherwise an adversary, Trudy, can record and replay challenge and/or response to impersonate Alice or Bob
- Use large random values
- K_{AB} needs to be protected at Alice and Bob (end points of communication)

We just discussed how challenge and response are used in authentication.

It is very important for the challenges, say R1 and R2, to be not easily repeatable or predictable. Otherwise an intruder such as Trudy can record the challenge and response between Alice and Bob and replay them to impersonate Alice or Bob.

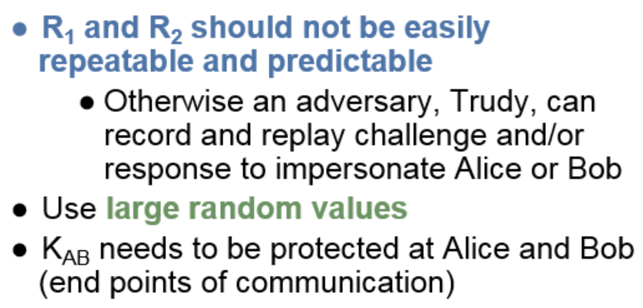


Mutual Authentication: Shared Secret

For example, an intruder called Trudy wants to impersonate either Alice or Bob. Suppose Trudy has spent time recording the authentication messages between Alice and Bob across multiple sessions. Now, if when Trudy tries to impersonate Alice and Bob happens to send R1, a challenge that has been used previously and recorded by Trudy, then Trudy can just simply send the response that she has recorded from Alice. That is, Trudy is able to send the cyber text of R1 to Bob and Bob will be tricked to believe that she is Alice.

As another example, suppose the challenges always increase in values. Then Trudy can first impersonate Bob and send a large challenge, say R1 equal to 1000, and record the response from Alice. Meanwhile, the real Bob is using a smaller R1, say R1 equal to 950, then Trudy can impersonate Alice some time in the future when the real Bob finally sends R1 equal to 1,000.

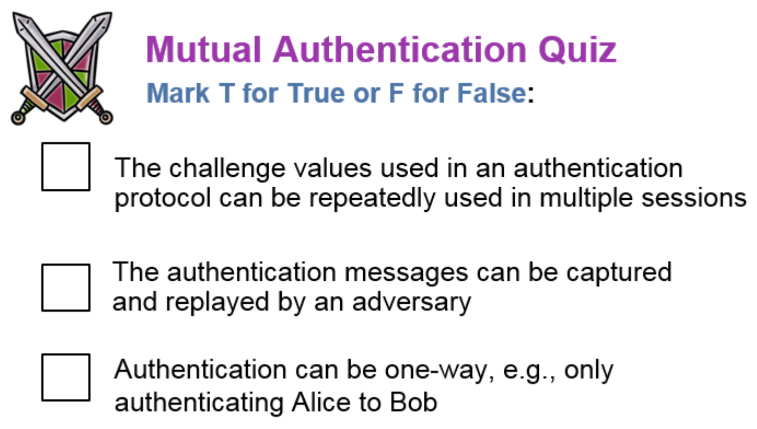
To recap, first of all, the messages that were sent over the Internet can be captured by intruder Trudy. And our job is to prevent these messages being replayed so that Trudy can't impersonate Alice and Bob.



- **R_1 and R_2 should not be easily repeatable and predictable**
  - Otherwise an adversary, Trudy, can record and replay challenge and/or response to impersonate Alice or Bob
- Use **large random values**
- K_{AB} needs to be protected at Alice and Bob (end points of communication)

And in order to do that, the random challengers R1 and R2 should really be not easily repeatable or predictable. Again, the most important precaution of using this authentication protocol is that R1 and R2 should not be easily repeatable or predictable. Otherwise an intruder such as Trudy can impersonate Alice and Bob by simply doing record and replay. Therefore, R1 and R2 should be large, random values.

Another security precaution is that the shared secret key, KAB, needs to be protected. Because if Trudy can steal a copy of the key, let's say from one of the complication end points either on Alice or Bob, then she can impersonate either Alice or Bob. In other words, the security of the end points is as important as the security of the link between the two end points.



Mutual Authentication Quiz

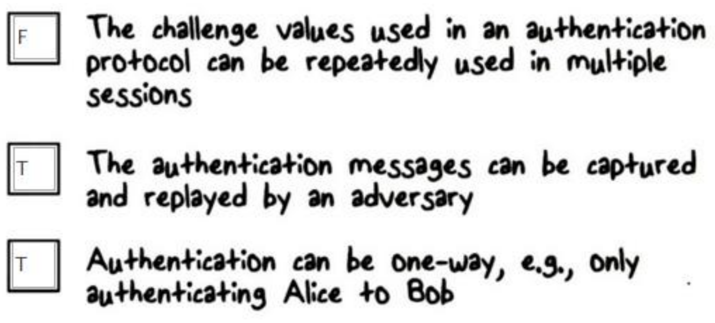
Mark T for True or F for False:

☐ The challenge values used in an authentication protocol can be repeatedly used in multiple sessions

☐ The authentication messages can be captured and replayed by an adversary

☐ Authentication can be one-way, e.g., only authenticating Alice to Bob

Now let's do a quiz. Mark each statement as true or false. Use T for true and F for false. First, the challenge values used in an authentication protocol can be repeatedly used in multiple sessions. Second, the authentication messages can be captured and replayed by an adversary. Third, authentication can be one-way, for example, only authenticating Alice to Bob.



[F] The challenge values used in an authentication protocol can be repeatedly used in multiple sessions

[T] The authentication messages can be captured and replayed by an adversary

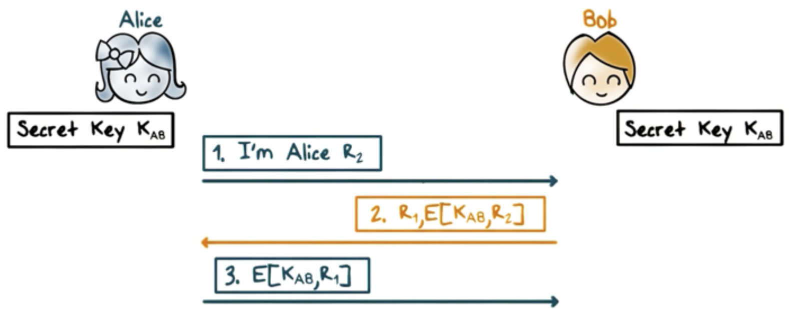
[T] Authentication can be one-way, e.g., only authenticating Alice to Bob

First, the challenge values used in an authentication protocol can be repeatedly used in multiple sessions. This is false. If the challenge values are repeated then there's a chance that the attacker can capture and replay the same challenge and response and impersonate Alice or Bob.

Second, the authentication messages can be captured and replayed by an adversary. This is true. We should always assume that the messages that are sent over the Internet can be captured by the adversary. An adversary can attempt to replay them in order to impersonate Alice or Bob.

Third, authentication can be one way, for example only authenticating Alice to Bob. This is true, as we have discussed the first three steps of the protocol authenticate Alice to Bob.

Mutual Authentication: Simplified

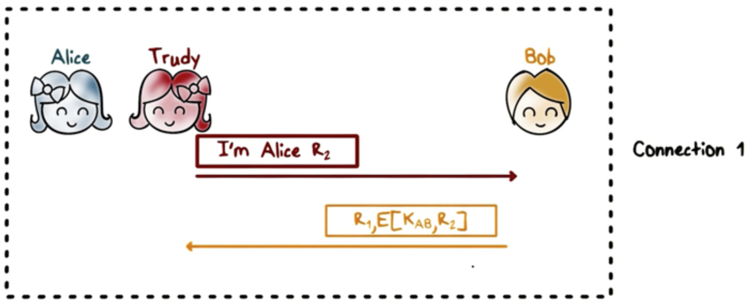


Mutual authentication protocol takes five steps. Can we simplify it? That is, can we use fewer steps? It seems that the following would work:

1. First, Alice says hi to Bob and sends along a challenge R2.
2. Second, in response, Bob sends Alice the ciphertext of R2, and his own challenge R1. Upon receiving this response, Alice decrypts the ciphertext to see if it matches the plain text, R2, that she has sent to Bob. If it matches, then she knows that she's communicating with Bob.
3. Third, Alice then sends Bob the ciphertext of R1. And Bob decrypts it and matches it with the plain text, R1, to authenticate Alice.

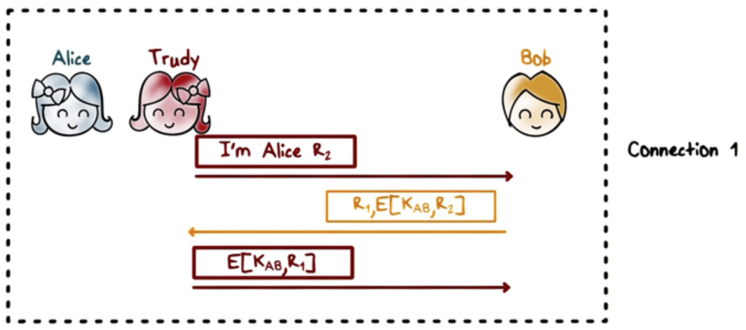
Is there anything wrong with this simplified protocol? It turns out that there is a reflection attack, a kind of man in the middle attacks. Here's how it works:

Mutual Authentication: Reflection Attack



First, Trudy impersonates Alice. By following the protocol, Trudy will be stuck at step three because she can not encrypt the challenge, R1, sent from Bob. She does not have the key KAB.

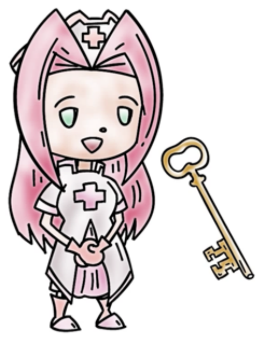
Mutual Authentication: Reflection Attack



Then Trudy simply opens another connection with Bob and, again, impersonating Alice. This time Trudy sends Bob the challenge, R1, that Bob has sent her in the first connection, the one that she stuck in. According to the protocol, Bob responds with a ciphertext of R1 and another challenge, R3. Notice that Bob has send the ciphertext of R1 to Alice. So what Trudy can do is to take this ciphertext of R1. Then Trudy can now go back to the step three of the first connection and send it back to Bob to complete the first connection. That is, at this point, the first connection successfully concludes, and Trudy has successfully impersonated Alice.

This is called a reflection attack because Trudy simply sends back to Bob what Bob had just sent her from another connection. So how can we prevent reflection attacks?

Mutual Authentication: Reflection Attack

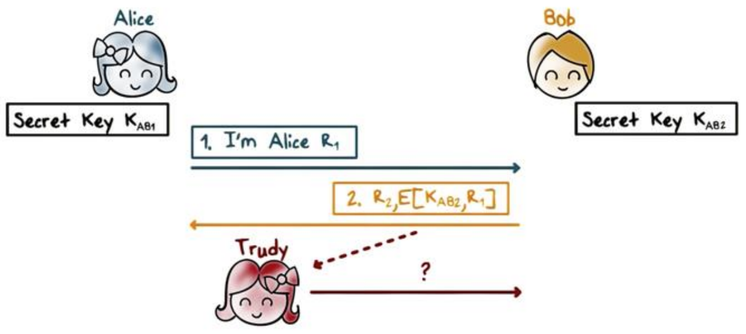


- **Fixes:**
  - Different keys for **initiator** and **responder**
    - Trudy can't get Bob to encrypt using Alice's key

One way is to use two different share keys. One is for the initiator of the connection, say Alice, and the other is for the responder, say Bob. The result is that the ciphertext that Trudy gets from Bob can not be presented as a ciphertext from Alice because these two ciphertexts should be using two different keys.

That is, there are two share keys being used. Say Alice used KAB1 and Bob used KAB2. Therefore the ciphertext from Bob is encrypted using KAB2, and Bob is going to expect a ciphertext using KAB1. Therefore, even though Trudy can accept the ciphertext, but this Cipher text is produced using KAB2. She cannot simply send it back to Bob because the ciphertext sent to Bob must be encrypted using KAB1.

Mutual Authentication: Reflection Attack

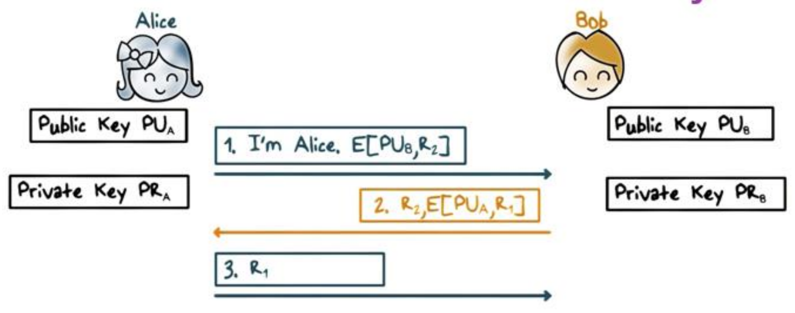


Mutual Authentication: Reflection Attack

- **Different type of challenges for initiator and responder**
  - **e.g.,** even number for initiator and odd number for responder

Another way to prevent the reflection attack is to use different challenges for the initiator and the responder. For example, we can use an even number challenge for Alice and odd number for Bob. Therefore, when Trudy receives a challenge, R1, from Bob the challenge is an odd number, and Trudy cannot use. This is a new challenge for Bob because Bob is expecting an even number.

Mutual Authentication Public Keys



- Variant:
  - Sign instead of encrypt

We can also use public key cryptography for mutual authentication:

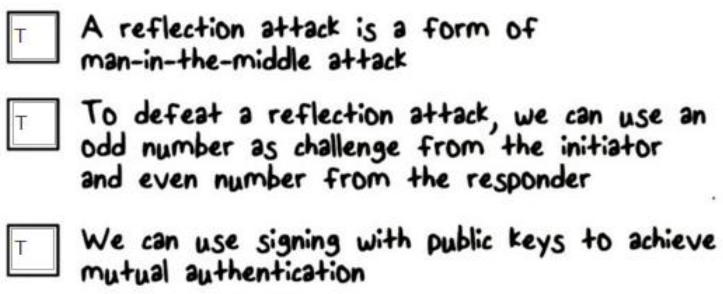
1. Suppose Alice and Bob have each other's public key. In this protocol, first, Alice sends Bob a challenge R2 and R2 is encrypted using Bob's public key.
2. Second, upon receiving this challenge Bob decrypts the server text using his own public key, and sends back the plain text challenge R2, along with his own challenge R1. And R1 is encrypted using Alice's public key.
3. Third, when Alice gets the response from Bob, the plain text R2 in Bob's response, tells Alice that she's communicating with Bob because only Bob has the private key that is paired with the public key that encrypted R2. Alice also decrypts the server text R1 using her own private key and sends a plain text back to Bob in step three. Bob will then know that he is communicating with Alice because only Alice has the private key that is paired with the public key used to encrypt R1. As an exercise, you can modify this protocol to use signing with private keys instead of encryption with public keys.

Attack Quiz

Mark T for True or F for False:

- ☐ A reflection attack is a form of man-in-the-middle attack
- ☐ To defeat a reflection attack, we can use an odd number as challenge from the initiator and even number from the responder
- ☐ We can use signing with public keys to achieve mutual authentication

Now let's do a quiz. For each statement, decide whether it is true or false, using T for true and F for false. First, a reflection attack is a form of man-in-the-middle attack. Second, to defeat a reflection attack, we can use an odd number as challenge from the initiator and even number from the responder. Third, we can use signing with public keys to achieve mutual authentication.



T A reflection attack is a form of man-in-the-middle attack

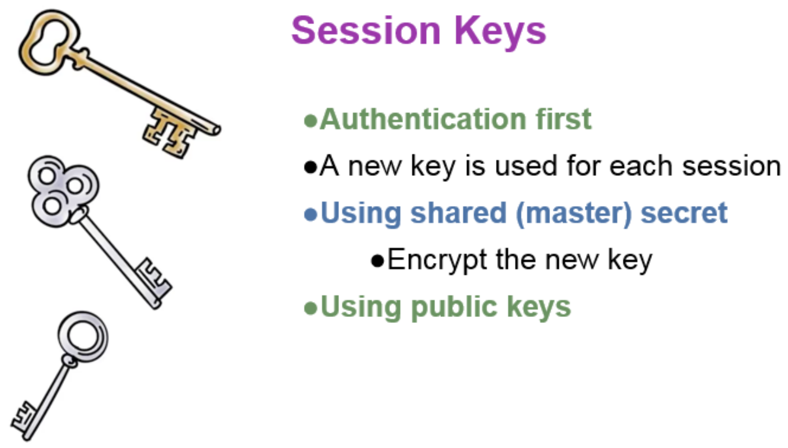
T To defeat a reflection attack, we can use an odd number as challenge from the initiator and even number from the responder

T We can use signing with public keys to achieve mutual authentication

First, a reflection attack is a form of man-in-the-middle attack. This is true, because in reflection attack, Trudy is the man in the middle. In particular, in order to impersonate Alice, Trudy intercepts the message Bob sends to Alice, and repay it back to Bob.

Second, to defeat a reflection attack, we can use an odd number as a challenge from the initiator and even number from the responder. This is true. By doing so, Trudy cannot repay the challenge that she received from Bob, which is an even number, back to Bob, because Bob is expecting an odd number.

Third, we can use signing with public keys to achieve mutual authentication. This is true. We have discussed a mutual authentication protocol using encryption with public keys, but you can easily modify the protocol using signing with public keys.



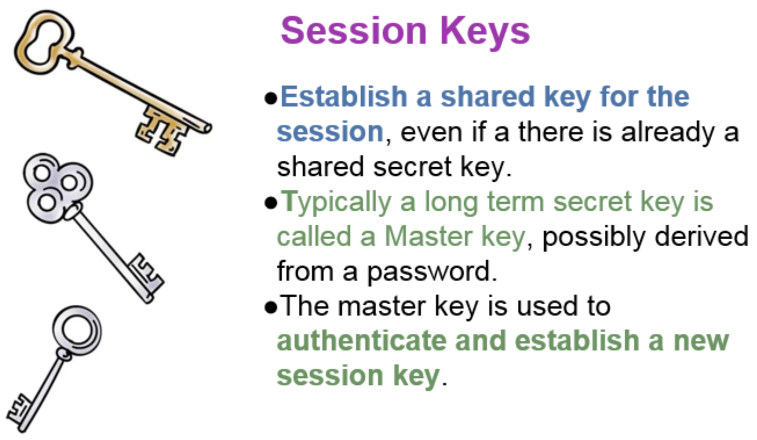
Session Keys

- Authentication first
- A new key is used for each session
- Using shared (master) secret
  - Encrypt the new key
- Using public keys

After authentication in order to protect message security, Alice and Bob will need to establish a shared secret key for each session.

This can be accomplished in several ways. Supposed Alice and Bob share a master secret key. Then Alice can use this master secret to encrypt a new key and send the encrypted new key to Bob. Or Alice and Bob can also use public keys to encrypt a new key.

We say that Alice and Bob should establish a shared key for each new session. This is true even if Alice and Bob already has a shared secret key.

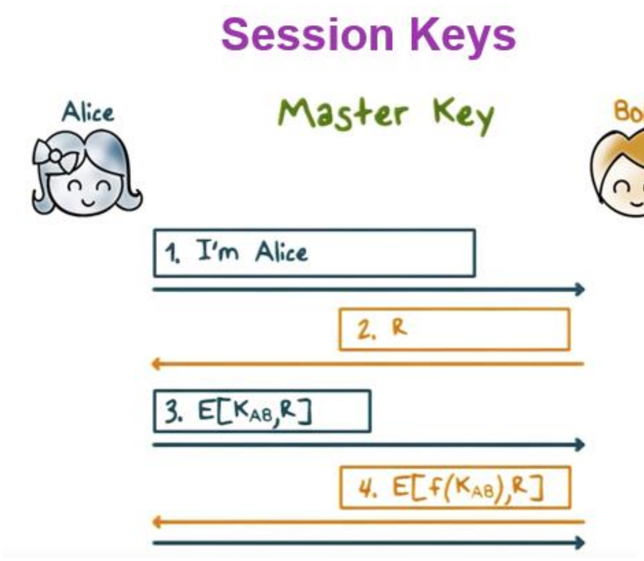


Session Keys

- Establish a shared key for the session, even if a there is already a shared secret key.
- Typically a long term secret key is called a Master key, possibly derived from a password.
- The master key is used to authenticate and establish a new session key.

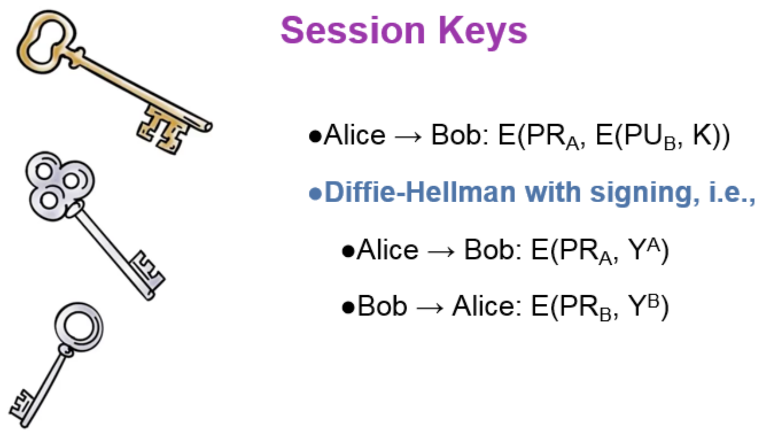
Typically, Alice and Bob share a long term secret key. And we call it the master key. For example, the master key can be derived from a password.

For each session, Alice and Bob would use the master secret key to authenticate and establish a new key for the session. Then all messages in the session are protected using the session key. The main benefit of using session key is that if the key is leaked or broken, the impact is limited only to the current session. Intuitively, the more a secret is used, the higher a chance that the secret can be leaked. Therefore, we should limit the use of the long term master secret. And only use it at the beginning of a session for authentication and establishing the session key.



Here's an example. Suppose Alice and Bob already share a master key, Kab.

The first three steps are just for Alice to authenticate Bob, say Bob is a server. Then, both Bob and Alice computer save session key that is based on the sheer master key and something about the current session. So that the session key is both a secret and unique to the session. So for example, they can add 100 to the master key and use the result as the key to encrypt all which is a challenge used in this session to authenticate Alice to Bob, and the result of encrypting R using a modified master key is the shared session key.



Session Keys

- Alice → Bob: $E(PR_A, E(PU_B, K))$
- **Diffie-Hellman with signing, i.e.,**
  - Alice → Bob: $E(PR_A, Y^A)$
  - Bob → Alice: $E(PR_B, Y^B)$

Alice and Bob can also use their public keys to exchange a shared session key.

For example, Alice can send to Bob a key, encrypt it using Bob's public key so that only Bob can decrypt and get the key. And then signs the result using Alice's private key. So Bob knows that the key is sent by Alice.

Or Alice and Bob can use their private keys to sign the public messages that they exchange in the Diffie-Hellman key exchange protocol.

And this can prevent the man in the middle attack, that is when Alice sends Bob the public message, she signs it. Likewise when Bob sends Alice the public message, he signs it with his private key.

Key Distribution Center (KDC)

- **Shared Master Keys do not scale easily**
- Each communication pair needs to share a master key

A major short coming of using pairwise key exchange based on a shared secret, is that it cannot scale. That is, suppose we use a shared master keys as a way to establish and exchange a new session key.

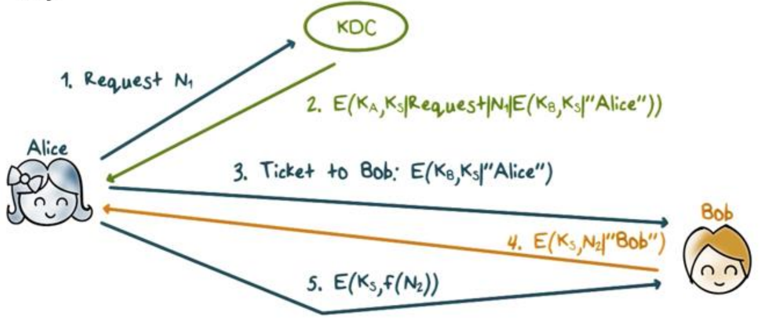
This scheme does not scale easily. That is Alice needs to share a master key with Bob, and then another master key with Carol, and so on and so forth.

Using a Key Distribution Center, or KDC, can solve this scalability problem.

Each party has his or her own master key shared with the KDC. That is, the KDC has many master keys, one for each party. But each party only keeps one master key that is shared with KDC. So, for example, Alice has KA that is shared with KDC. And Bob has KB that is shared with KDC.

Key Distribution Center (KDC)

K_A, K_B are master keys shared with KDC, K_s is a session key


KDC
1. Request N_1
2. $E(K_A, K_S|Request|N_1|E(K_B, K_S|"Alice"))$
Alice
3. Ticket to Bob: $E(K_B, K_S|"Alice")$
Bob
4. $E(K_S, N_2|"Bob")$
5. $E(K_S, f(N_2))$

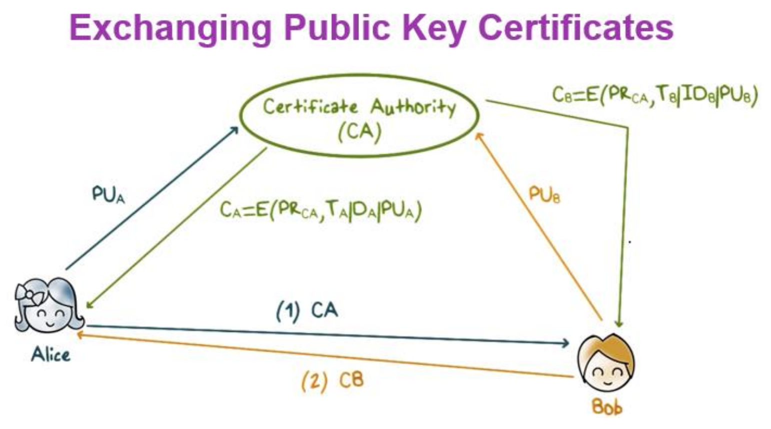
Now, suppose Alice and Bob wants to have a secure session, therefore, they need a session key KS. First, Alice sends a request to KDC saying that, I need a key to talk to Bob along with a nonce and 1. A nonce is a random value.

Then, the KDC sends a message back to Alice that's encrypted using the master key KA that is shared between Alice and KDC. This message contains the session key KS that the KDC just created for Alice and Bob to share. The message also contains the same request that Alice sent to KDC along with the same nonce value, N1 and a message record ticket. The ticket is encrypted using the master key KP that is shared between Bob and the KDC. And it contains session key KS and the ID of Alice.

When Alice gets back the message from the KDC, she can decrypt it because she has the master key, KA. And so, she can extract the session key KS. And she knows that the message is from the KDC, and it's fresh, that is, it is not a replay, because only the KDC can use KA to equip properly a message that contains the original request and the nonce that she just set.

Alice then sends the ticket to Bob. Note that only Bob can decrypt the ticket, because it is encrypted using KB, the master key shared between Bob and the KDC. In fact, when Bob decrypts the ticket, he knows that the ticket is created by the KDC because only the KDC can encrypt the ID of Alice properly. And he knows that the session key, KS was created by KDC and is for communication with Alice.

Then Bob sends a message that contains a nonce N2, which is a random value, and it's ID encrypted, using the session key KS to Alice. When Alice receives this message, she knows that she is communicating with Bob, because only he can decrypt the ticket and get a session key, Ks, and encrypt the ID properly. Alice then performs an agreed upon transformation on N2. Say, add 100 to N2 and encrypt the result using KS, and sends it back to Bob. This proves to Bob that he is communicating with Alice because only she has the session key KS.

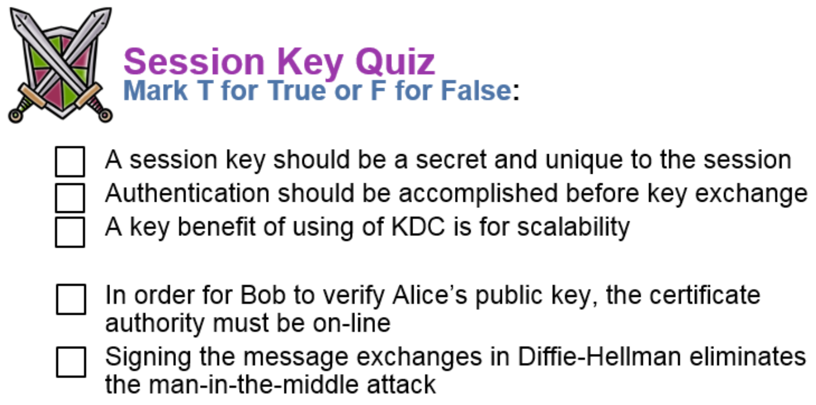

Exchanging Public Key Certificates
Certificate Authority (CA)
$C_B=E(PR_{CA},T_B|ID_B|PU_B)$
PU_A
$C_A=E(PR_{CA},T_A|ID_A|PU_A)$
PU_B
(1) CA
(2) CB
Alice
Bob

The distribution of public keys is often done through a certificate authority or CA.

Alice sends a public key to the CA. The CA verifies Alice's identification, and then creates a certificate of Alice's public key and sends it to Alice. The certificate contains time of creation, validity period, and the ID of Alice, and the public key. And, it is signed using a CA's public key.

Alice can then send this certificate to any user, say Bob. Or, Alice can publish it so that any user can get it. It is assumed that all users have the CA's public key, and therefore a user, say Bob, can use the CA's public key to verify the certificate and obtain Alice's public key.

Likewise, Bob can get his certificate and send it to Alice so that Alice has Bob's public key.



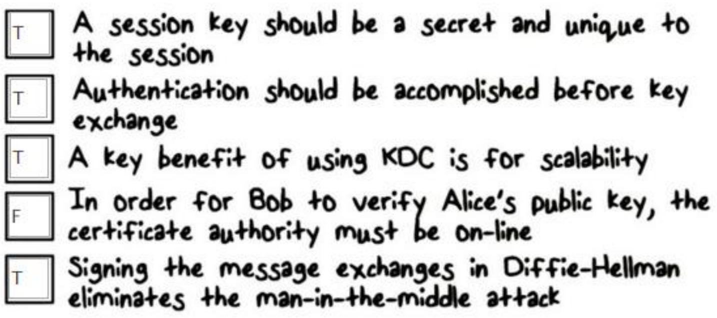
Session Key Quiz

Mark T for True or F for False:

☐ A session key should be a secret and unique to the session
☐ Authentication should be accomplished before key exchange
☐ A key benefit of using of KDC is for scalability

☐ In order for Bob to verify Alice's public key, the certificate authority must be on-line
☐ Signing the message exchanges in Diffie-Hellman eliminates the man-in-the-middle attack

Now let's have a quiz on session keys. Decide whether each statement is true or false. Use T for true, and F for false. First, a session key should be a secret and unique to the session. Second, authentication should be accomplished before key exchange. Third, a key benefit of using KDC is for scalability. Fourth, in order for Bob to verify Alice's public key, the certificate authority must be online. Fifth, signing the message exchanges in Diffie-Hellman eliminates the man-in-the-middle attack.



T	A session key should be a secret and unique to the session
T	Authentication should be accomplished before key exchange
T	A key benefit of using KDC is for scalability
F	In order for Bob to verify Alice's public key, the certificate authority must be on-line
T	Signing the message exchanges in Diffie-Hellman eliminates the man-in-the-middle attack

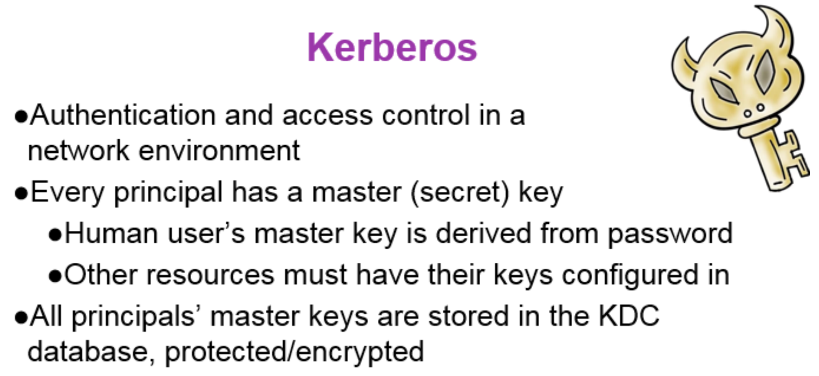
First, a session key should be a secret, and unique to the session. This is true.

Second, authentication should be accomplished before key exchange. This is true.

Third, a key benefit of using KDC is for scalability. This is true.

Fourth, in order for Bob to verify Alice's public key, the certificate authority must be online. This is false, because as long as the users have the CA's public key, they can verify the certificate.

Fifth, signing the message exchanges in Diffie-Hellman eliminates the man-in-the-middle attack. This is true.



Kerberos

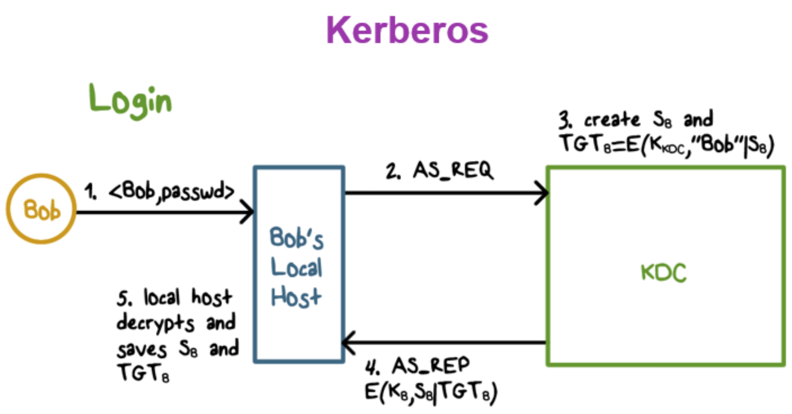
- Authentication and access control in a network environment
- Every principal has a master (secret) key
  - Human user's master key is derived from password
  - Other resources must have their keys configured in
- All principals' master keys are stored in the KDC database, protected/encrypted

Now let's look at how the principles of authentication and key exchange that we have discussed are used in actual systems.

Kerberos is a standard protocol used to provide authentication and access control in a network environment, typically an enterprise network.

Every entity in a network, that is, all users and network resources, such as workstations and printers, have a master key that it shares with the Kerberos servers. And the Kerberos servers perform both authentication and key distribution. So for convenience, we use KDC to refer to a Kerberos server. For a human user, the master key is derived from his or her password. For a network device, the key is configured in.

All the keys are stored securely at the KDC.


Kerberos
Login
Bob
1. <Bob,passwd>
Bob's Local Host
2. AS_REQ
3. create S_B and TGT_B=E(K_{KDC},"Bob"|S_B)
KDC
4. AS_REP E(K_B,S_B|TGT_B)
5. local host decrypts and saves S_B and TGT_B

Now let's go over a few typical Kerberos scenarios.

When user Bob logs in, his workstation contacts the KDC with an authentication service request message. The KDC generates a per day session key, SB. And the so-called ticket-granting ticket that contains SB and the ID of Bob. And the ticket is encrypted using the KDC's own key.

The KDC then sends back a message to Bob's workstation, and this message is the authentication service response. The message contains the per day session key and the ticket-granting ticket. And the message is encrypted using the shared master key between Bob and KDC. And because KB is the master key shared between Bob and KDC, only Bob's local workstation can decrypt this message. And then it can store the per day session key and the ticket granting ticket.

Bob's local workstation will then use SB for subsequent messages with the KDC, and it would include the ticket granting ticket to remind and convince the KDC to use SB. That is, any new request to the KDC will include the TGT in the request message. And the new ticket from the KDC will be encrypted using SB.

Kerberos

Kerberos Benefits:

- Localhost does not need to store passwords
- The master key that the user shares with the KDC is only used once every day

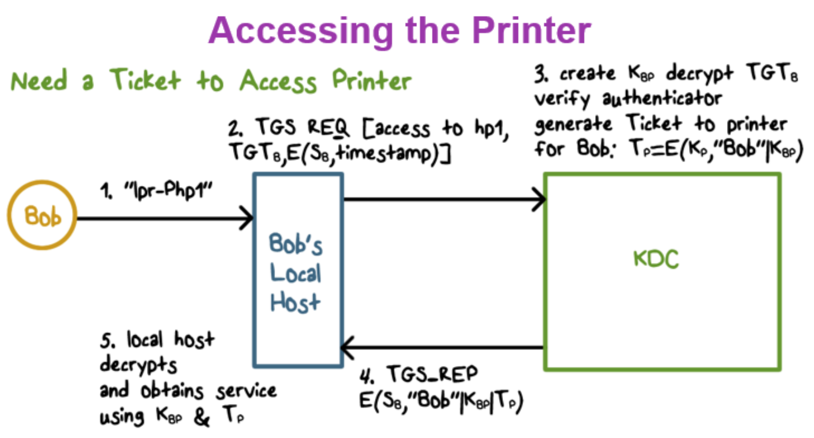
This limits exposure of the master key

There are several benefits with this scheme.

First, the localhost does not need to store Bob's password or password hash once Bob has logged in and once he has obtained SB from the KDC.

Second, the master key KB, that Bob shares with the KDC, is only used once everyday when Bob logs in initially.

That is, the exposure of the master key, which is derived from Bob's password and is subject to password guessing attacks is very limited.


Accessing the Printer
Need a Ticket to Access Printer
1. "lpr-Php1"
2. TGS REQ [access to hp1, TGT_B, E(S_B, timestamp)]
3. create K_BP decrypt TGT_B verify authenticator generate Ticket to printer for Bob: T_P=E(K_P, "Bob"|K_BP)
4. TGS_REP E(S_B, "Bob"|K_BP|T_P)
5. local host decrypts and obtains service using K_BP & T_P
Bob
Bob's Local Host
KDC

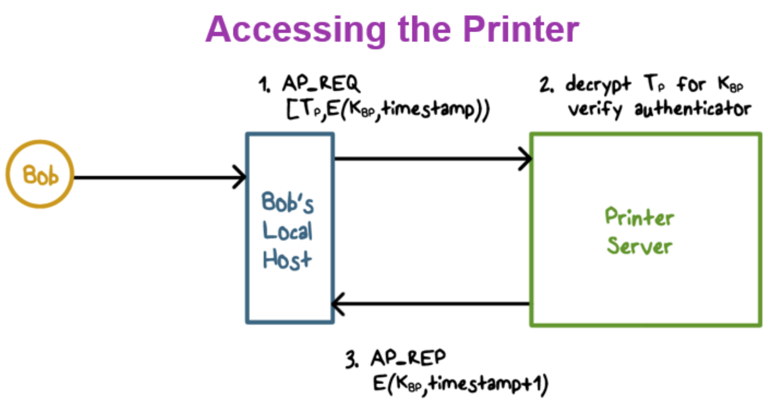
Now suppose Bob wants to access the printer HP1. That is, Bob wants to send a print job to the printer HP1.

His local host sends a ticket granting service request to the KDC. The request contains the ticket granting ticket and a authenticator. Which is the current time stamp encrypted using Bob's per day session key, SB.

When the KDC gets the request, it can decrypt the TGT. And it knows that it is a valid TGT. Because only the KDC has the key to properly encrypt the ID of Bob contained in the TGT. It then uses the key, SB, contained in the TGT to verify the authenticator by decrypting it. And checking the time stamp is current. This proves that the sender is Bob, because only Bob has the key SB. That can encrypt the current time stamp properly.

The KDC then generates a ticket for Bob to communicate with the printer. This ticket contains a session key KBP and Bob's ID and its encrypted using the printer's master key. Note that a network device such as a printer has a long and random master key configured in. And it's typically hard to guess or crack. Therefore, the tickets for these devices are encrypted using their master keys.

The KDC sends a ticket granting service response to Bob's local host. It contains a session key KBP. The ID of Bob and the ticket to the printer and it is encrypted using Bob's private key Sb. Therefore, only Bob's local station can decrypt this, and verify that it is from the KDC. Because only the KDC can decrypt the ID of Bob correctly with the key SP. Because the key sp is embedded in TGT and only the KDC can decrypt the TGT.


Accessing the Printer
1. AP_REQ [T_P, E(K_BP, timestamp))
2. decrypt T_P for K_BP verify authenticator
3. AP_REP E(K_BP, timestamp+1)
Bob
Bob's Local Host
Printer Server

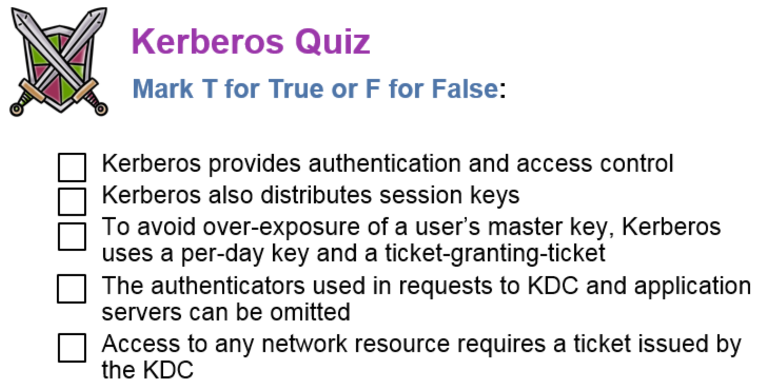
Then Bob's local host can authenticate itself to the printer.

It can send a message to the printer containing the ticket. The authentication request contains the ticket and an authenticator. The authenticator is the cipher text of the current timestamp encrypted. Using a share key between Bob's local host and the printer.

When the printer gets the ticket, it can decrypt the ticket using his master key shared with the KDC. Because the ticket was created by the KDC. And it was encrypted using the share key between the printer and the KDC. That is, the printer can verify that the KBP was created by the KDC, to communicate with Bob's Local Host.

The print server can then use KBP to verify the authenticator by decrypting the server text and verifying that the result matches with the current time stamp.

It then sends a response to authenticate itself by say, adding one to the current timestamp and encrypt it. Using the share key between Bob's Local Host and the printer. And after this, authentication steps, then Bob's Local Host can send a print job to the printer.

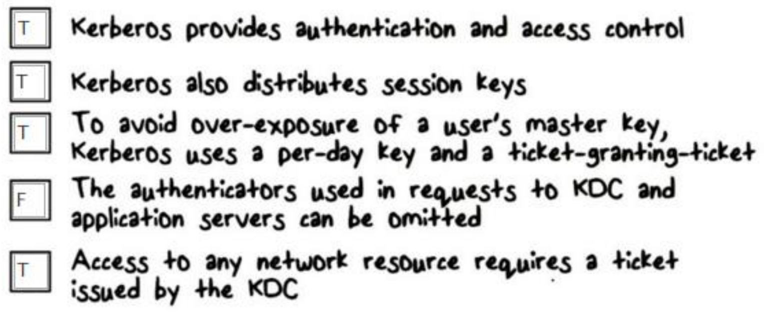


Kerberos Quiz

Mark T for True or F for False:

- ☐ Kerberos provides authentication and access control
- ☐ Kerberos also distributes session keys
- ☐ To avoid over-exposure of a user's master key, Kerberos uses a per-day key and a ticket-granting-ticket
- ☐ The authenticators used in requests to KDC and application servers can be omitted
- ☐ Access to any network resource requires a ticket issued by the KDC

Now let's do a quiz on Kerberos. Decide if each statement is true or false, using T for True and F for False. First, Kerberos provides authentication and access control. Second, Kerberos also distributes session keys. Third to avoid over exposure of a user's master key, Kerberos uses a per-day key and a ticket granting ticket. Four, the authenticators use the new quest to KDC and application servers can be omitted. Fifth, access to any network resource requires a ticket issued by the KDC.



- [T] Kerberos provides authentication and access control
- [T] Kerberos also distributes session keys
- [T] To avoid over-exposure of a user's master key, Kerberos uses a per-day key and a ticket-granting-ticket
- [F] The authenticators used in requests to KDC and application servers can be omitted
- [T] Access to any network resource requires a ticket issued by the KDC

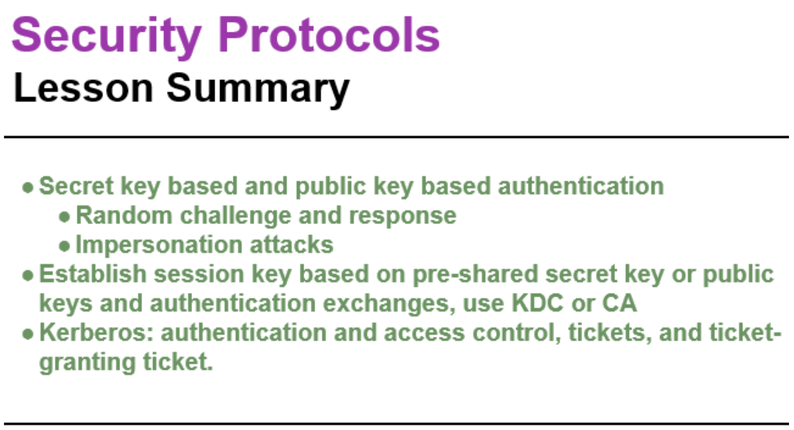
First, Kerberos provides authentication and access control. This is true.

Second, Kerberos also distributes session keys. This is true.

Third, to avoid over-exposure of a user's master key, Kerberos uses a per-day key and a ticket-granting-ticket. This is true.

Fourth, the authenticators used in requests to KDC and application servers can be omitted. This is false. These authenticators are used to authenticate the senders. And authentication is an important goal of Kerberos. Therefore, these authenticators must not be omitted.

Fifth, access to any network resource requires a ticket issued by the KDC. This is true.



Security Protocols
Lesson Summary

- Secret key based and public key based authentication
  - Random challenge and response
  - Impersonation attacks
- Establish session key based on pre-shared secret key or public keys and authentication exchanges, use KDC or CA
- Kerberos: authentication and access control, tickets, and ticket-granting ticket.

Authentication can be accomplished using random challenge and response. And a pre-shared secret key or public keys.

Session keys can be established using a pre-shared secret key or public keys and can involve KDC and CA.

The Kerberos system provides authentication and access control in a enterprise network environment.