Applied Cryptography and Network Security

William Garrison
bill@cs.pitt.edu
6311 Sennott Square

Mediated Authentication and Kerberos

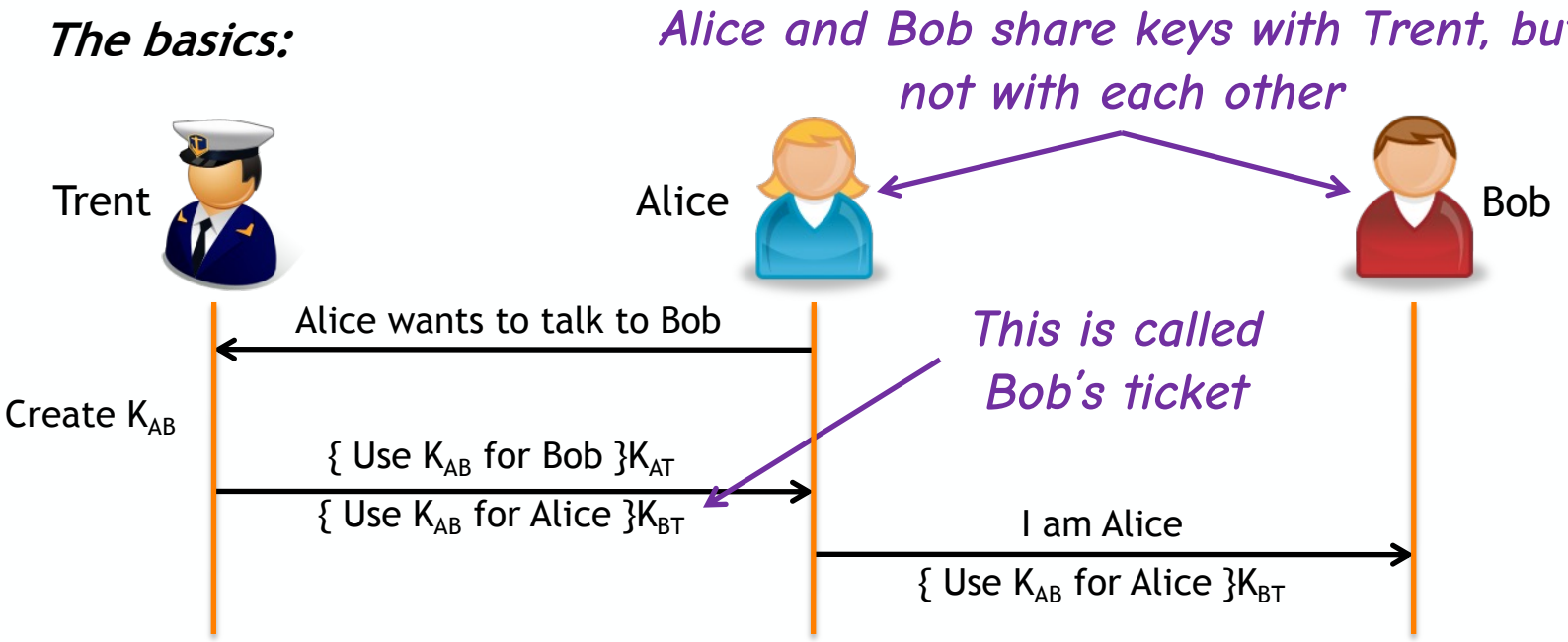
University of Pittsburgh

What if we like the speed of symmetric key cryptography, but not the key management headaches?

Specifically, how can we let two users securely establish a shared key?

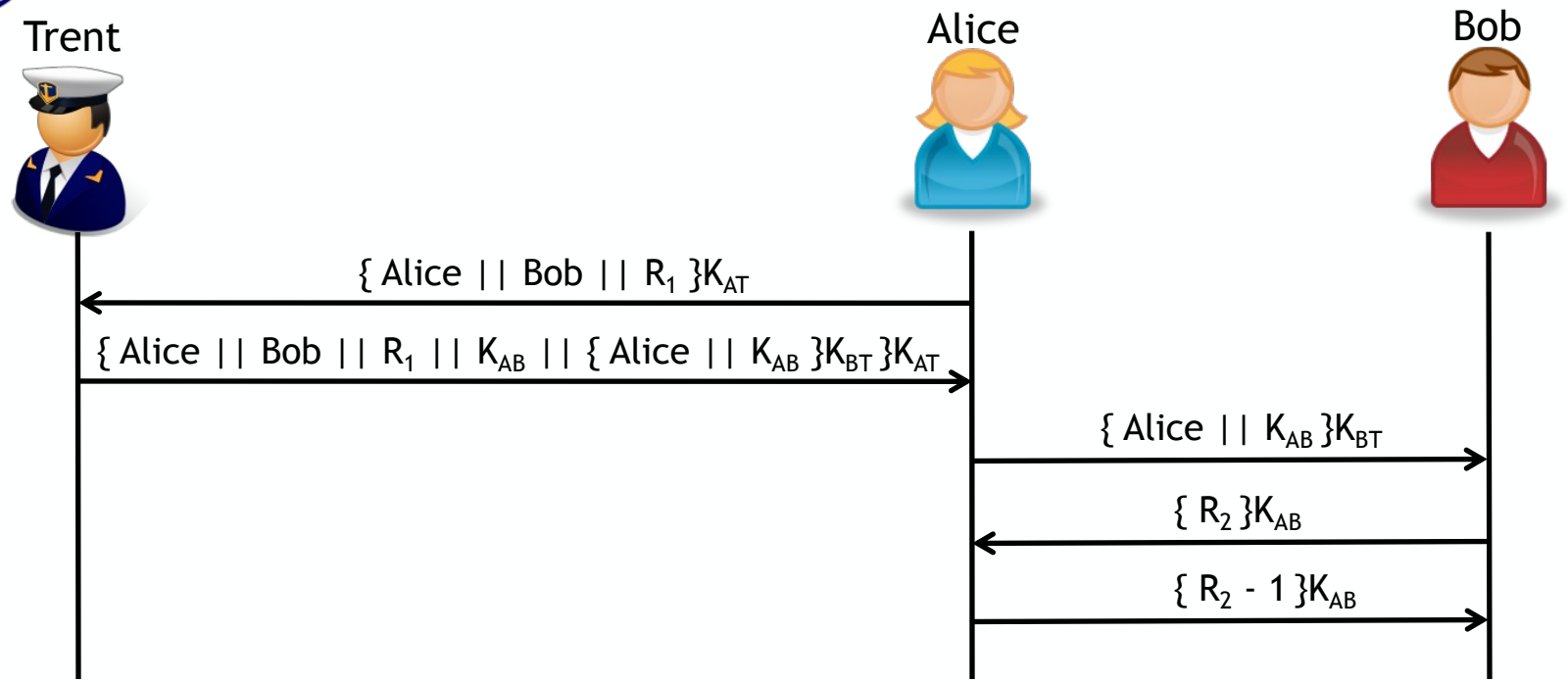
Mediated authentication protocols make use of a trusted mediator, or key distribution center (KDC), to make this possible!

The basics:



Note: This protocol is incomplete, as it does not authenticate Alice and Bob to one another. However, it is a good place to start…

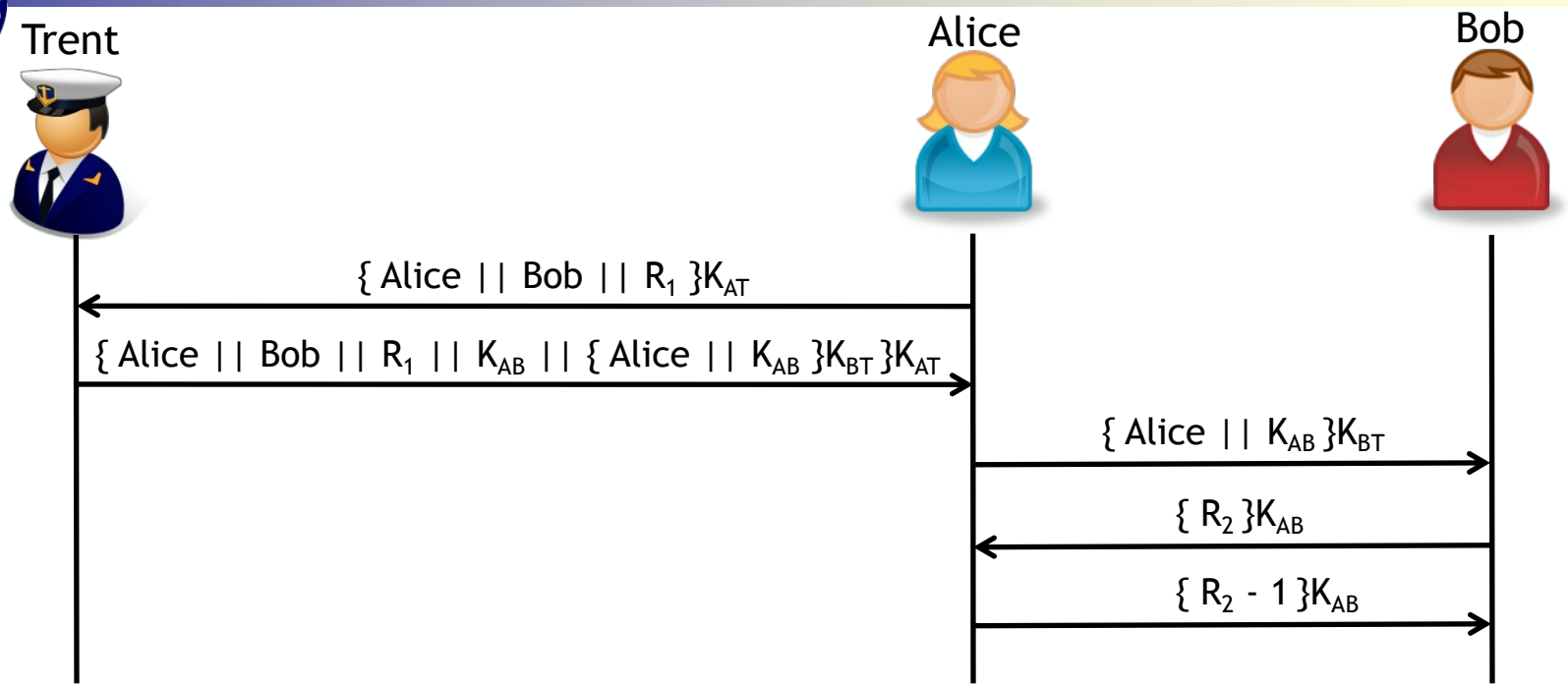
The Needham-Schroeder protocol is a well-known mediated authentication protocol



Note: R_1 and R_2 are called nonces

- Must be generated at random (unpredictable)
- Cannot be used in more than one protocol execution

Why does Needham-Schroeder work?



After message 2 Alice

- Knows that this message is fresh
- Knows that the session key is to be shared with Bob

After message 3, Bob knows that he has a shared key with Alice

After message 5, Bob knows that this key is fresh (Why?)

The Needham-Schroeder protocol assumes that all keys remain secret

Assume that Eve intercepts the message { Alice || K_{AB} }K_{BT} and later learns the session key K_{AB}

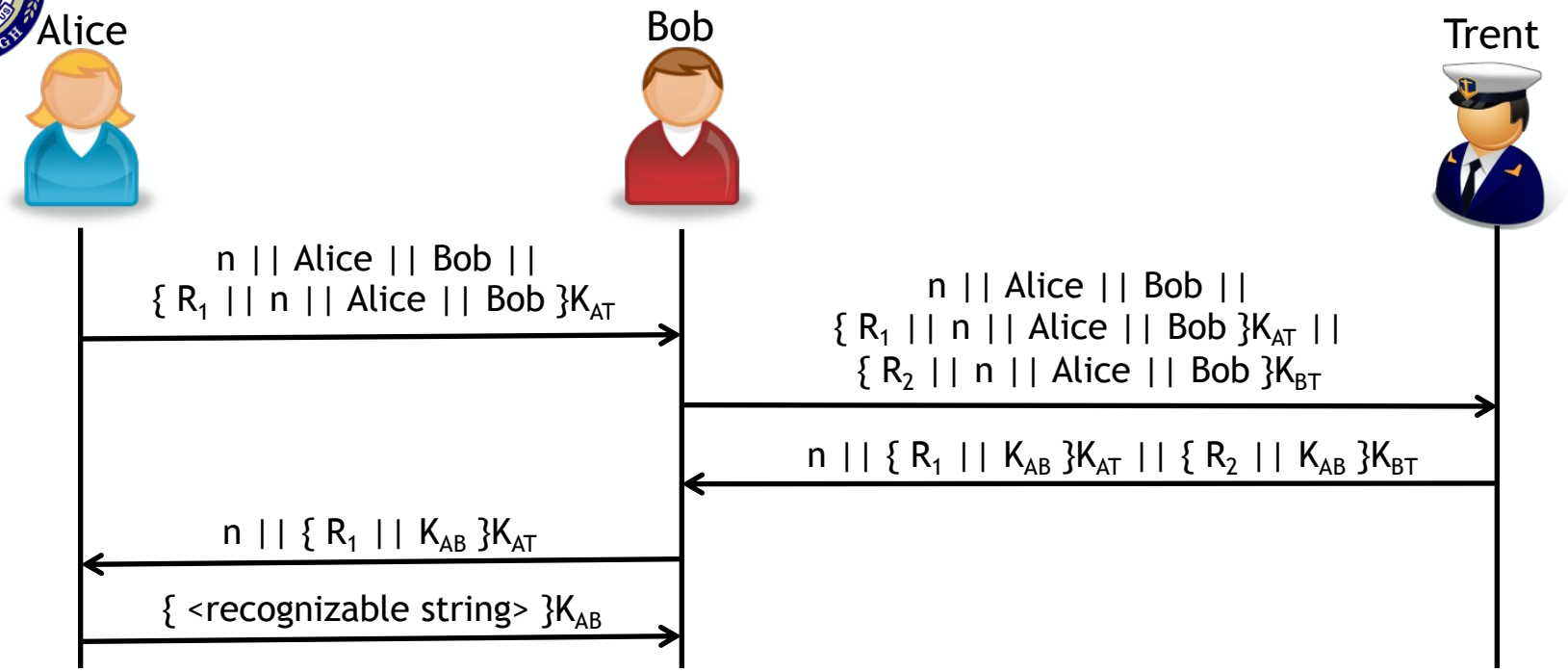
Question: *Why might a session key become compromised?*

Eve can now launch a replay attack!

- Eve can replay the message { Alice || K_{AB} }K_{BT}
- She can intercept Bob's response { R_3 } K_{AB} to Alice
- Since Eve knows K_{AB}, she can decrypt this message and reply { R_3 - 1 } K_{AB}

How can we defend against this type of attack?

The Ottway-Rees protocol prevents this attack



Properties of this protocol:

- After message 3, Bob knows that he has a fresh session key to share with Alice that was generated by Trent
- After message 4, Alice knows that she has a fresh session key to share with Bob that was generated by Trent
- After message 5, Bob knows that Alice received the shared key

Why doesn't a compromised session key subvert this protocol?

Assume that Eve:

- Records the message $n \,||\, \{ R_1 \,||\, K_{AB} \}K_{AT} \,||\, \{ R_2 \,||\, K_{AB} \}K_{BT}$
- Breaks the key K_{AB}

Now, say that Eve tries to forge a version of message 4 to Alice

- $n \,||\, \{ R_1 \,||\, K_{AB} \}K_{AT}$

If Alice does not have an ongoing exchange with Bob, this forgery will fail

- Why? No initial state saved

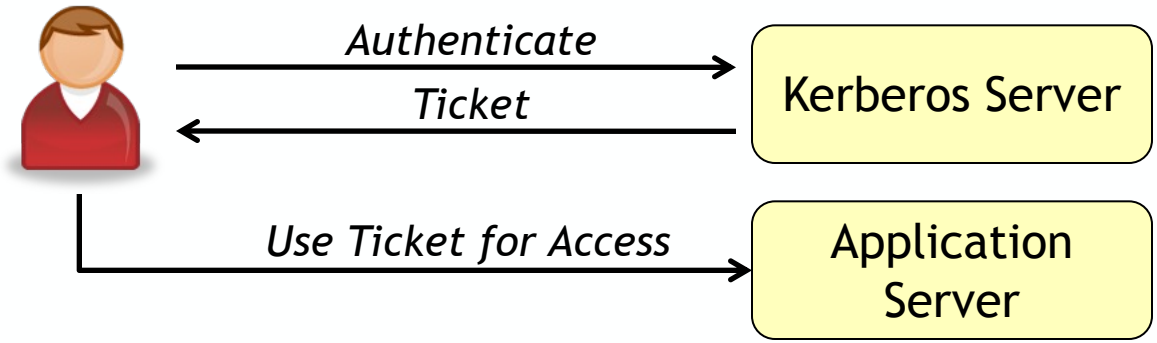
If Alice does have an ongoing exchange with Bob

- The forgery will fail if the number n does not match
- If n does match, the forgery will fail because Alice will be using a different nonce R_1

Kerberos Overview

Kerberos is a mediated authentication protocol

This is kind of like our term project, eh?



This protocol is based on the following assumptions:

- Server(s) used by Kerberos are highly secured (How?)
- Application servers are moderately secure, though may be compromised
- Client machines are untrusted

Kerberos uses secret key cryptography to allow users to authenticate to networked services from any location

Kerberos Design Goals

Main goal: Breaking into one host should not help the attacker compromise the overall security of the system

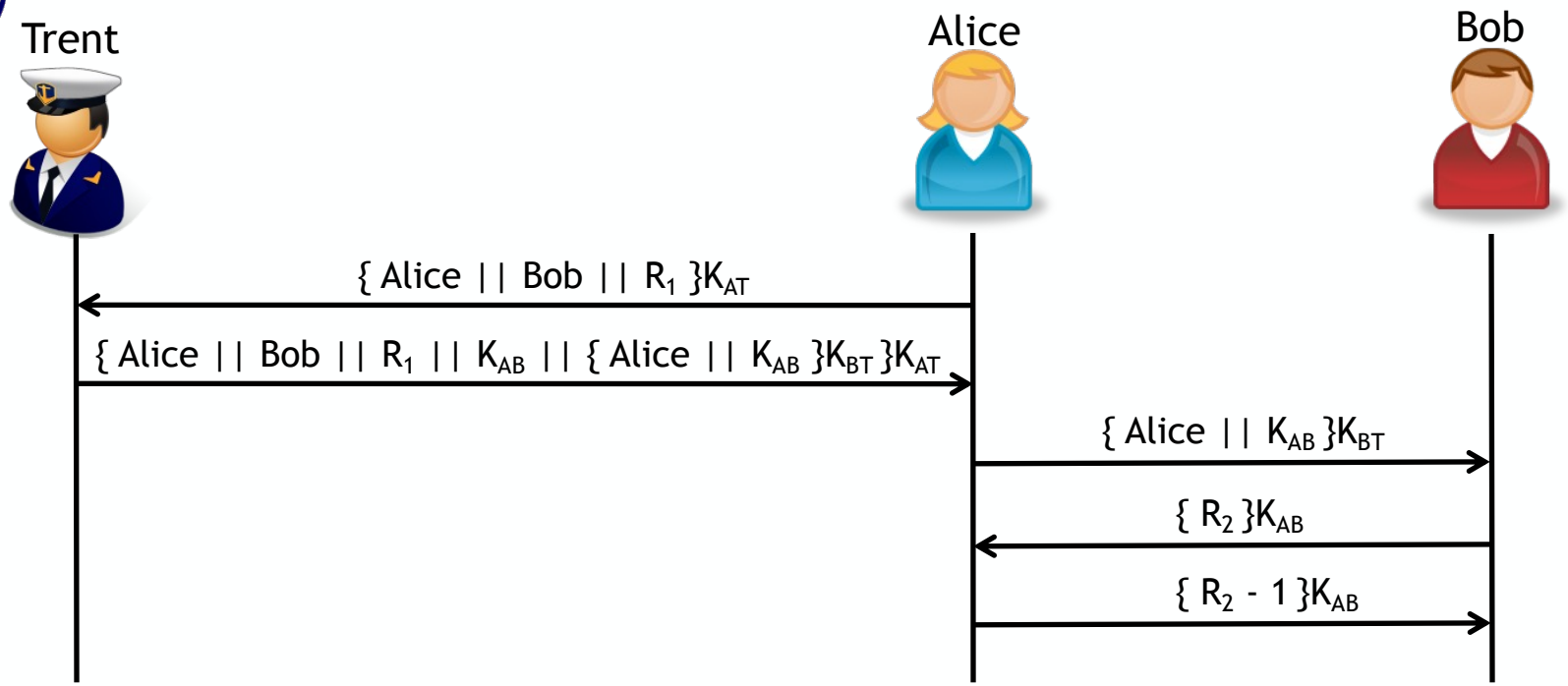
Client authenticator goals:

- Users cannot remember cryptographic keys, so keys should be derived from the user's password
- Passwords should not be sent in cleartext
- Passwords should not be stored on the server
- The client's password should be exposed as little as possible

The use of Kerberos should require only minimal modifications to existing applications

So, how does this work?

Kerberos is based on the Needham-Schroeder secret key authentication protocol



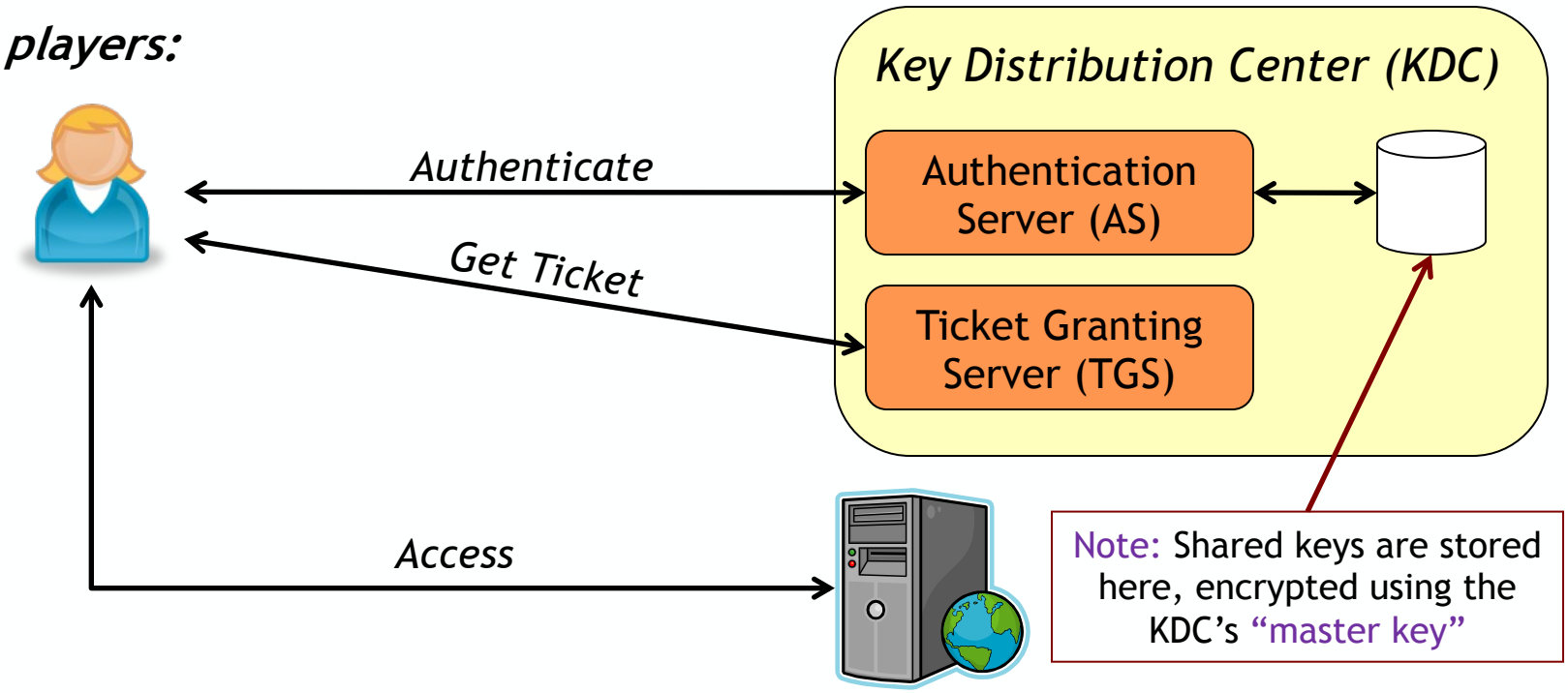
After message 2, Alice

- Knows that this message is fresh
- Knows that the session key is to be shared with Bob

After message 3, Bob knows that he has a shared key with Alice

After message 5, Bob knows that this key is fresh

Kerberos v4: The Basics

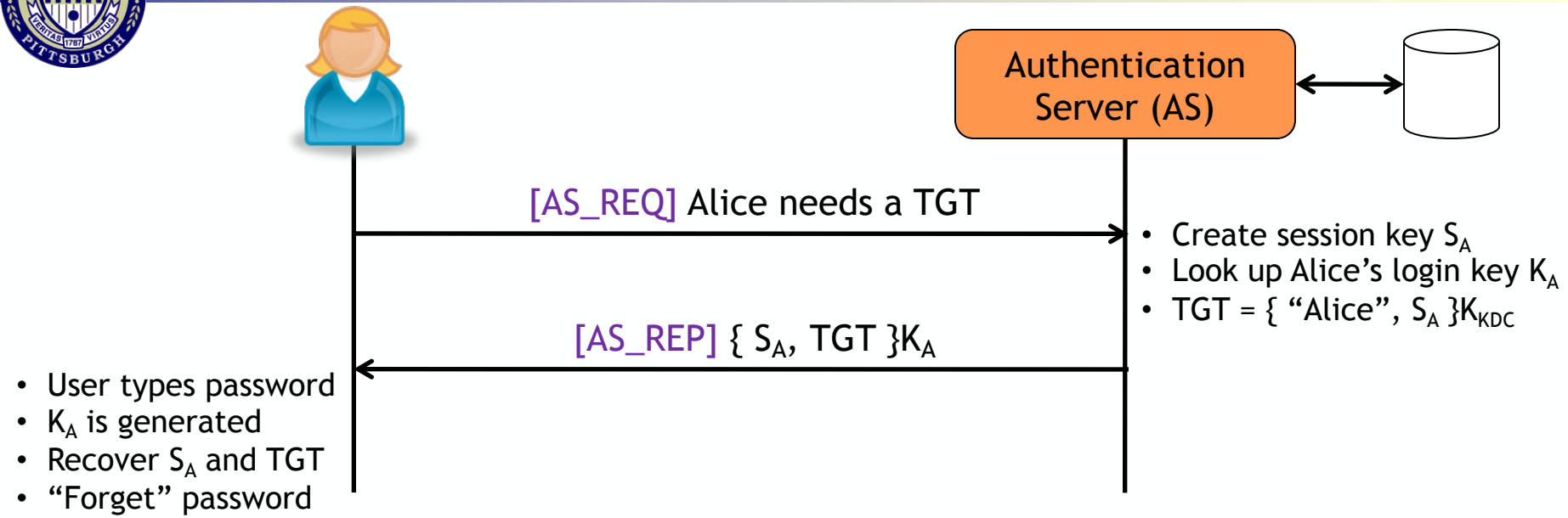


All principals in the system share a secret key with the AS

- User keys derived from their password
- Service keys are random cryptographic keys

The cryptographic algorithm used by Kerberos is (Triple) DES

Step 1: Obtaining a ticket-granting ticket (TGT)



The above process is used to initiate a login session

- Password used to initiate the session
- S_A is used for subsequent exchanges

Note that the user password is not needed until after the TGT is obtained

- Why is this a good thing?
- Why is this a bad thing?

What is the purpose of this process?

In a single session, a user may want to access many different machines

For example…

- Download a file from a secured web server
- SSH into another machine to compare results with experimental data
- Email a colleague to inquire about an oddity in the file
- FTP an updated file to the secured server
- …

By obtaining a single session key S_A, Alice only uses her password once!

- This minimizes the amount of time that the password is exposed
- If S_A is cracked, the password is still safe

Furthermore, the TGT frees the KDC from maintaining any state

- Recall that TGT = {"Alice", S_A }K_{KDC}
- No need to track S_A at the server, just ask Alice for her TGT

Obtaining a TGT: Message Detail

AS_REQ

# Bytes	*Content*
1	Version of Kerberos (4)
1	Message Type (1)
≤ 40	Alice's name
≤ 40	Alice's instance
≤ 40	Alice's realm
4	Alice's timestamp
1	Desired ticket lifetime
≤ 40	Service name (krbtgt)
≤ 40	Service instance

In multiples of 5 minutes (up to about 21.5 hours)

Helps Alice match request/reply pairs

Note: This message is sent unencrypted

Obtaining a TGT: Message Detail

AS_REP

Bytes	*Content*
1	Version of Kerberos (4)
1	Message type (2)
≤ 40	Alice's name
≤ 40	Alice's instance
≤ 40	Alice's realm
4	Alice's timestamp
1	Number of tickets (1)
4	Ticket expiration time
1	Alice's key version number
2	Credentials length
var	Credentials

Bytes	*Content*
8	S_A
≤ 40	TGS name
≤ 40	TGS instance
≤ 40	TGS realm
1	Ticket lifetime
1	TGS key version number
1	Length of ticket
var	Ticket
4	Timestamp
var	Padding of 0s

Note: The "Credentials" field is encrypted with the Alice's login key

Obtaining a TGT: Message Detail

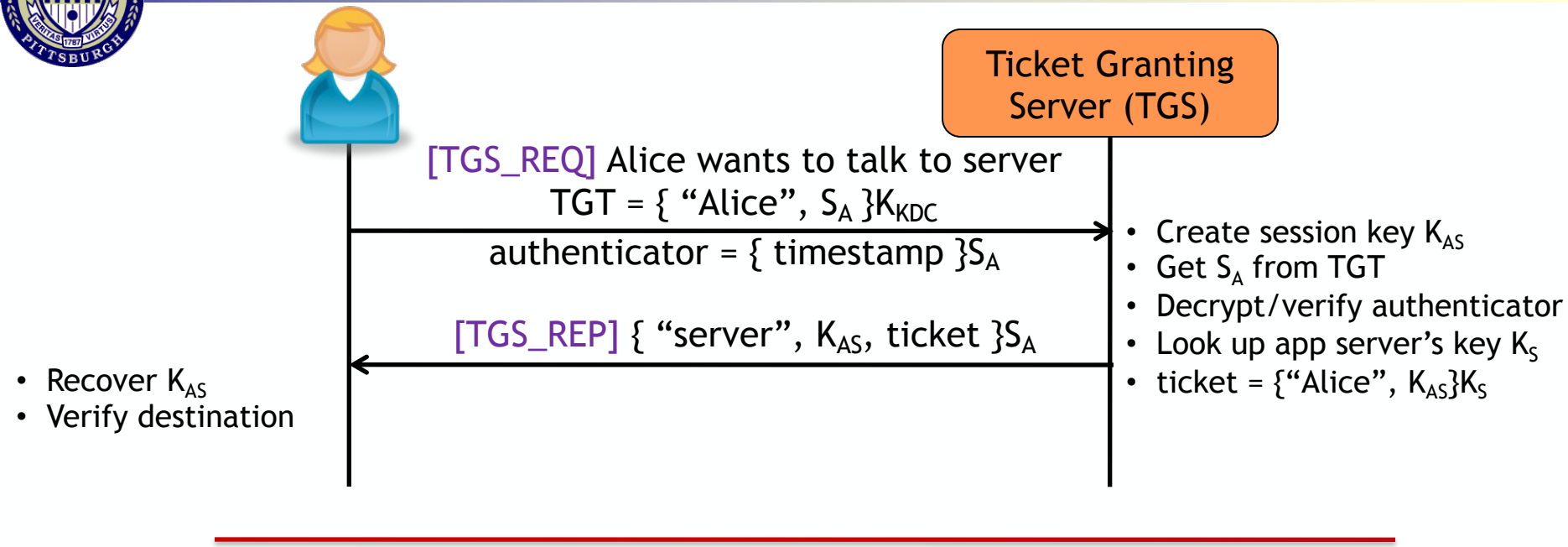
This is the credentials field

Bytes	*Content*
8	S_A
≤ 40	TGS name
≤ 40	TGS instance
≤ 40	TGS realm
1	Ticket lifetime
1	TGS key version number
1	Length of ticket
var	Ticket
4	Timestamp
var	Padding of 0s

Bytes	*Content*
≤ 40	Alice's name
≤ 40	Alice's instance
≤ 40	Alice's realm
4	Alice's IP address
8	Session key S_A
1	Ticket lifetime
4	KDC timestamp
≤ 40	TGS name
≤ 40	TGS instance
var	Padding of 0s

Note: The "Ticket" field is encrypted with the TGS's secret key

Step 2: Obtaining a service ticket



Interesting notes:

- Alice did not use her password to authenticate!
- The TGS did not need to maintain any state to verify Alice's identity

The authenticator attests to the freshness of the current exchange

- This means that Kerberos requires synchronized clocks (usually ~5 mins)
- Not actually needed in this exchange (Why?)

Obtaining a Service Ticket: Message Detail

TGS_REQ

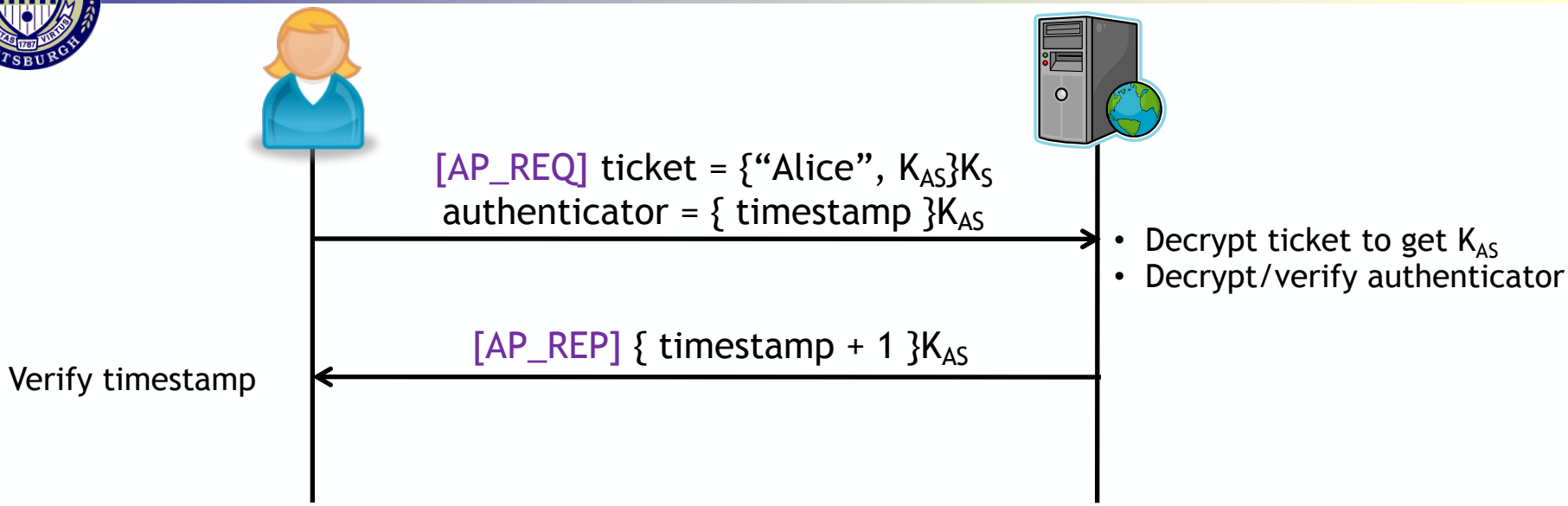
Bytes	*Content*
1	Version of Kerberos (4)
1	Message Type (3)
1	KDC key version number
≤ 40	KDC realm
1	Length of TGT
1	Length of authenticator
var	TGT
var	authenticator
4	Alice's timestamp
1	Desired ticket lifetime
≤ 40	Service name
≤ 40	Service instance

Copied from credentials field of AS_REP

Bytes	*Content*
≤ 40	Alice's name
≤ 40	Alice's instance
≤ 40	Alice's realm
4	checksum
1	5ms timestamp
4	Timestamp
var	Padding

Note: The TGS_REP message is the same format as AS_REP

Step 3: Using a service ticket



The AP_REQ message authenticates Alice to the server

- Only the KDC knows K_S, so the ticket for Alice is authentic
- If timestamp is recent, then this message is fresh and sent by Alice

The AP_REP message authenticates the server to Alice

- Only Alice, the server, and the KDC know K_{AS}
- If the timestamp is as expected, then this message is fresh

Question: How can we prevent replay attacks?

Using a Service Ticket: Message Detail

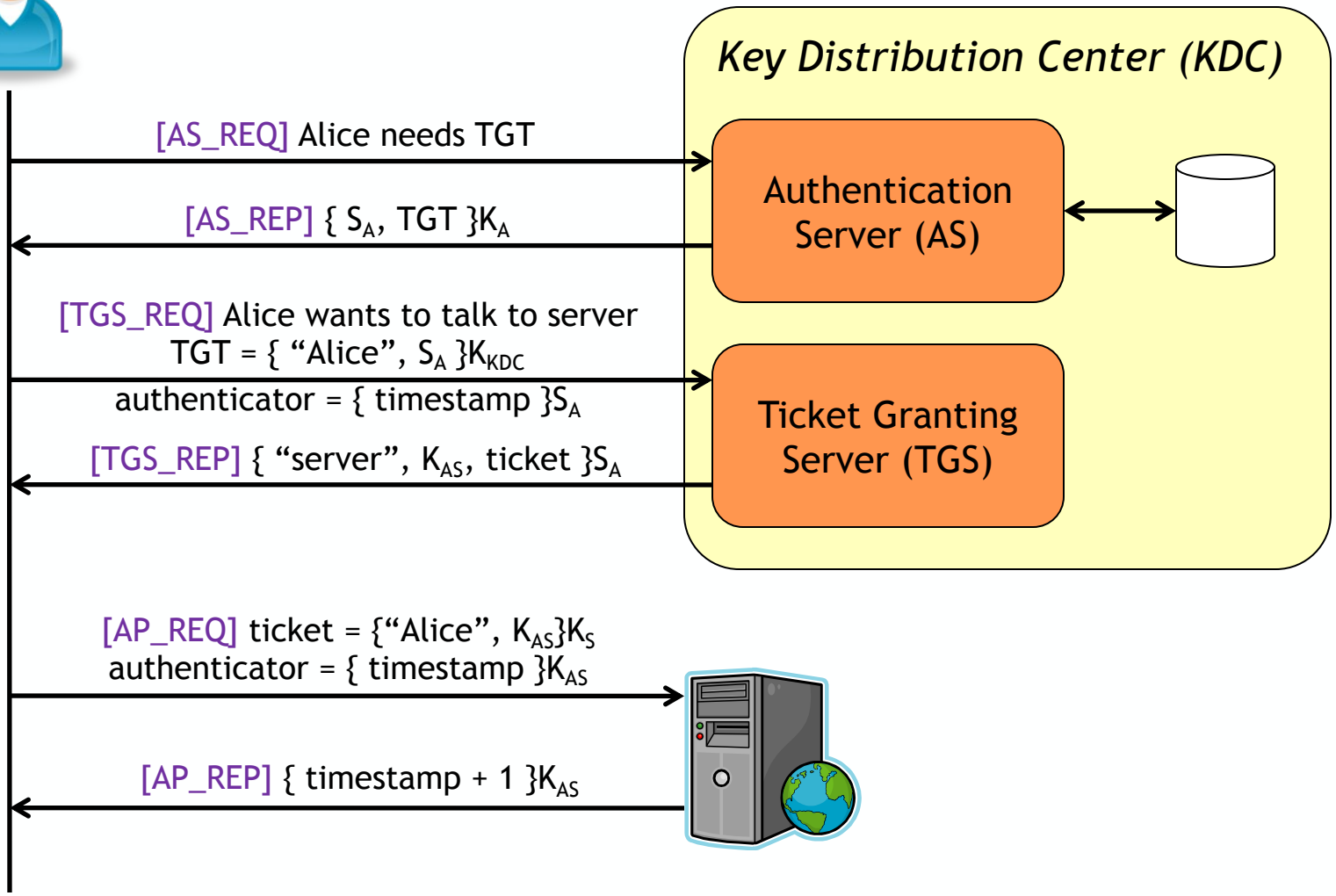
AP_REQ

Bytes	*Content*
1	Version of Kerberos (4)
1	Message Type (8)
1	Server's key version number
≤ 40	Server's realm
1	Length of ticket
1	Length of authenticator
var	ticket
var	authenticator

Copied from credentials field of TGS_REP

The ticket and authenticator follow the same format as in the TGS_REQ messages

Putting it all together…



…

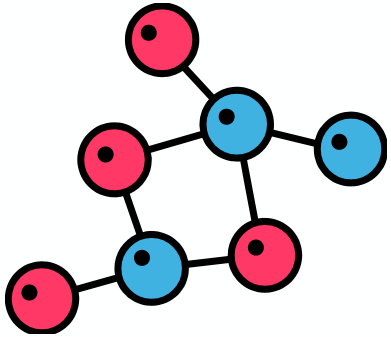
Is the assumption of a global KDC *really* realistic?

Problem: The KDC knows all keys!

- Probably not reasonable in mutually-distrustful organizations
- Scalability as number of users increases is poor
- Very valuable single point of attack

Problem: Reliability and fault tolerance

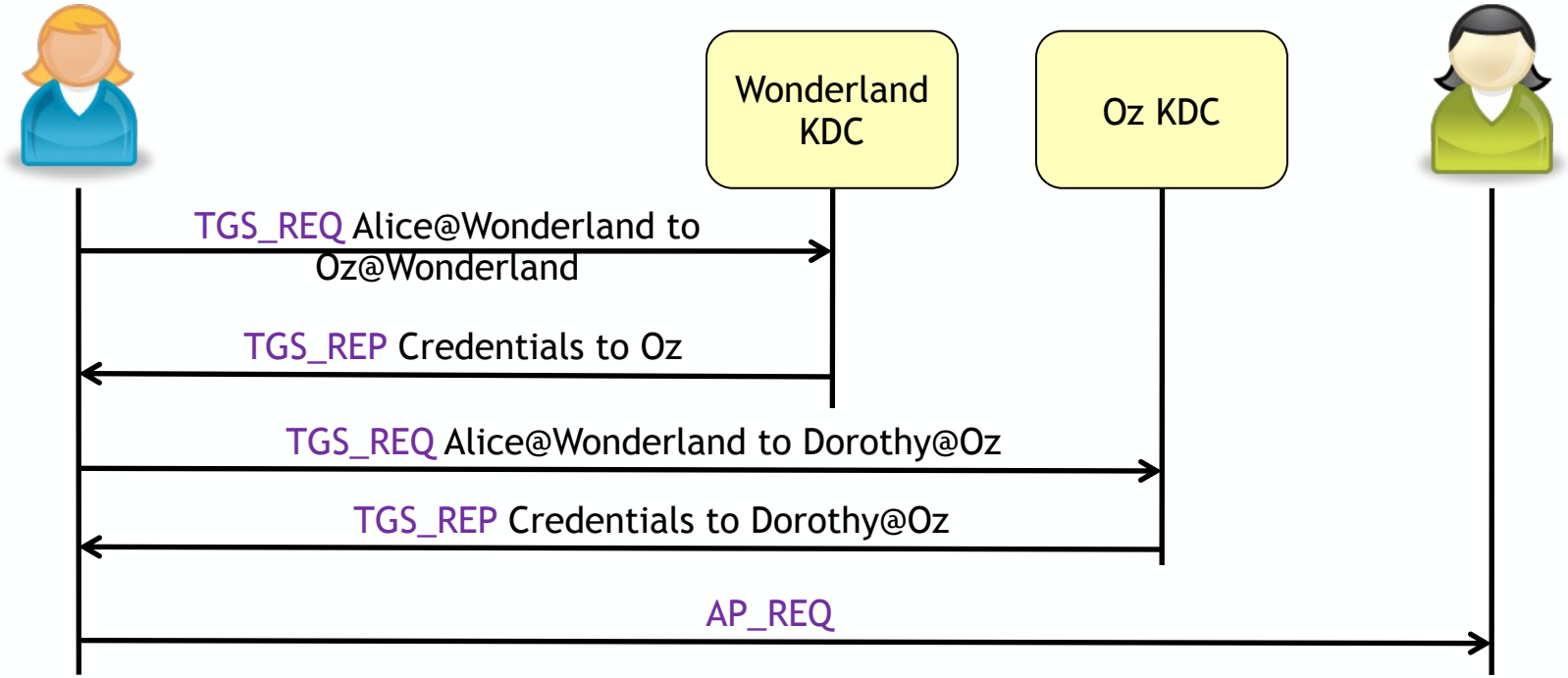
- A single KDC is a single point of failure
- If users cannot authenticate, they cannot work!
- Even if KDC is up all the time and everyone trusts it, it will probably no be able to serve all requests in a timely manner…



Kerberos is widely used, right? How does it address these problems?

Kerberos solves the untrusted KDC problem by allowing inter-realm authentication

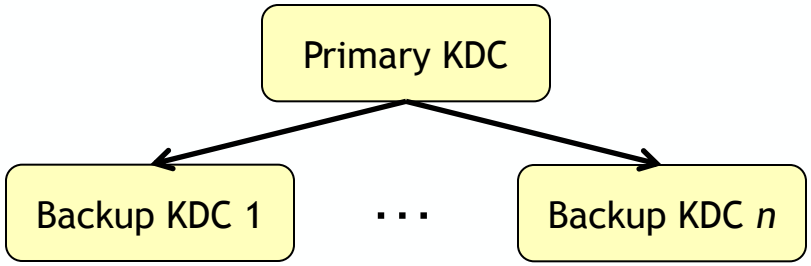
Example: Alice from the realm "Wonderland" wants to talk to Dorothy in the realm "Oz". Clearly, the Wonderland KDC knows nothing about principals managed by the Oz KDC. How do we proceed?



Note: For inter-realm authentication to work, the KDCs for each realm must agree to share keys a priori

Kerberos solves the failure and bottleneck problems by replicating the KDC's database

Goal: Any KDC should be able to service any client request



For this to work:

- The KDCs must all share the same key
- The databases managed by each replica must be consistent with the primary

How can the replicated databases maintain consistency with the primary?

- Primary DB contents are periodically downloaded by the backups
- Backups are used exclusively for read only operations (Is this a problem?)

How is the DB protected during transmission?

- Confidentiality: Provided "for free" since DB is stored encrypted
- Integrity: Keyed hash using shared secret key

Despite being a widely-deployed authentication solution, Kerberos v4 is far from perfect

Security

- Kerberos v4 is based on DES
- Integrity provided using non-standard techniques
- Does not support the use of other algorithms

Clocks and timestamps

- Maximum ticket lifetime is ~21.5 hours
- Cannot renew tickets
- Cannot obtain tickets in advance

Networks and naming

- 4 bytes used for network address
- What about IPv6??
- Names constrained to 40 characters

Kerberos v5 fixes these problems!

Supports extensible security suites. New algorithms can be added to the protocol as they are discovered. Standard techniques used for integrity protection.

Security

- Kerberos v4 is based on DES
- Integrity provided using non-standard techniques
- Does not support the use of other algorithms

ASN.1 used to encode names and addresses. Much more flexible.

Tickets have start and end times, can be renewed, and use a different timestamp format.

Clocks and timestamps

- Maximum ticket lifetime is ~21.5 hours
- Cannot renew tickets
- Cannot obtain tickets in advance

Networks and naming

- 4 bytes used for network address
- What about IPv6??
- Names constrained to 40 characters

Conclusions

Kerberos is a widely-used authentication paradigm based on the Needham-Schroeder authentication protocol

Authentication via Kerberos is a three-step process

1. Password-based authentication to the AS
2. TGT returned by the AS is used to request a service ticket from the TGS
3. Service ticket is used to mutually authenticate with a service

Inter-realm authentication allows users in different administrative domains to mutually authenticate

Many KDCs are replicated to prevent bottlenecks in the event of failure

Next time: Public key infrastructure (PKI) models